Final Report - Artificial Intelligence Project

Image Perspective Transformation using Autoencoders

Arnaud FRAMMERY, Ismail OUSSAID, Youssef AMMI,
Souhail HADGI, Soufiane JELOUANE, Yassir HANAFI

Special thanks to : Arnault Chazareix, Théo Estienne.

Summary:

CentraleSupélec

SICARA

I. Introduction

At Paris, one can find the Startup Studio M33 which is a startup incubator in the creation of digital solutions for diverse clients. The idea of our project has come into existence in the office of Sicara. Founded in 2016, it is part of M33 and a startup specialized in Image Recognition. Their R&D team of experts (Fig.1) provides measured added-value to many companies' image recognition projects on a daily basis.

Fig.1 Sicara team

Our client is Arnault Chazareix, who is a former Centrale Paris student in Software Engineering and a former intern in The Digital Lab of Paris and in Feedly *CA* as a Data Scientist and Software Engineer. Since 2018, Arnault is a R&D Data Scientist in Computer Vision. He has published 3 Artificial Intelligence & Deep Learning articles in Sicara's website and he is working on Few-Shot Classification Algorithms for Computer Vision.

The project named "*Réconcilier des prédictions sous plusieurs angles*" (*Reconciling predictions from multiple angles*) consists of developing an Artificial Intelligence capable of transforming images perspective (Left, Center and Right). For instance, if we capture a fast-food dish from the right side (one available camera) and we want to detect every element a consumer has taken to bill the client for his menu. At first sight, the best thing would be to add another camera in a different position that would take another perspective picture, which is costly. Then, our client has come with the idea of an IA that would generate the image (Fig.2) as if it was taken from the other side. This solution would be less expensive on a long term for companies and is useful to avoid frauds and articles hidden by mistake.

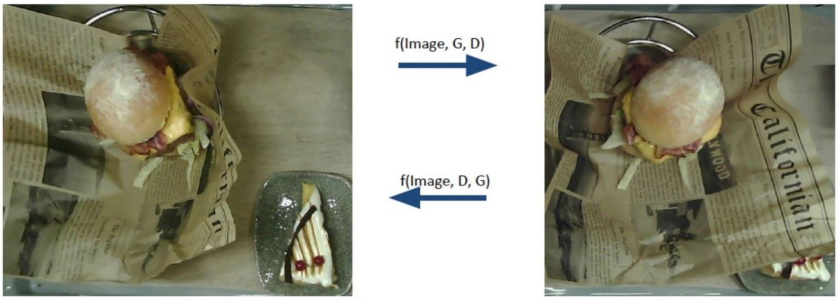


Fig.2 Image Transformation example (hidden pie for Left angle)

In the project, the client has some expectations. First, Mr. Chazareix wants a demonstration video made of images generated by the implemented Artificial Intelligence. Second, given highways surveillance video streams (Fig.3), the purpose is to be able to transform the perspective so well that a pre-trained Object Recognition Network can still recognize cars, trucks, etc. in transformed images.

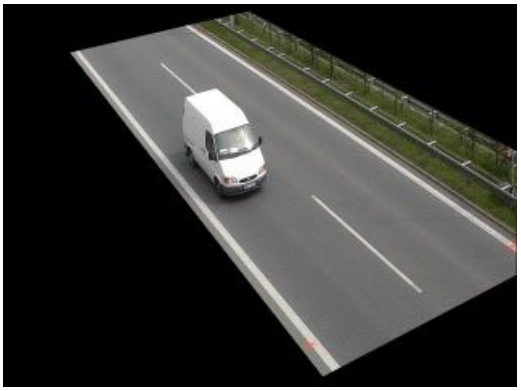


Fig.3 Typical input image

Finally, the client wants us to work on Auto-encoders that would change images point of view. Then, the client aims at benefiting from our work insofar as we could provide him a functioning model to operate the perspective transformation and thanks to a Deep Learning pipeline, we could build a record of unworthy ideas and useful ones.

In the following parts, we will discuss some state-of-the-art research papers dealing with Deep Learning methods, Image Transformation and Image Denoising. Then, we will present our achievement and further analysis of our results in three axes: Loss functions, Perspective Transformation & Image Denoising. Finally, we will conclude with presentation of our final AI and by presenting the mutual benefits we and the client have had from the project.

II. State-of-the-art

II.1. Perspective Transformation

When it comes to computer vision and image generation/classification, Deep Learning architectures have to be explored especially in the context of creating a working AI.

Deep Learning for image classification

Real-time object detection systems can be used to detect specific objects on images/videos and be used as a criteria for the quality of an image/video generator. YOLO is a fast and accurate state-of-the-art object detector that detects a lot of categories of objects, it uses a single neural network to get a global context of the image. Pre-trained models can be used to detect cars for example with a certain precision. These type of models are used for classification. To generate new image/videos we need to look at other architectures.

Generative models

Generative adversarial networks

One of the main approach to image generation is using Generative adversarial networks. [1]. GANs (Appendix.1) put two neural networks of any type one against the other to generate new data that can fake being the real data. One of the two neural networks is the generator, it generates new data that are fed into the discriminator which is the second neural network. The discriminator tries to detect which data is authentic and which is fake, the generator tries to generate data that can pass for being authentic.
For image generation, the initial input can be a noisy image (gaussian noise). The output produced by the generator is compared to the real image (the desired output) using an objective function: usually a minimax function since we're taking a game-theoretic approach.

$$objective\ function = min_G max_D E_{x \sim p_{data}}[logD(x)] + E_{z \sim p(z)}[log(1 - D(G(z)))]$$

This objective function can be minimized by the method of gradient descent.
Specific application of the GANs in image rotation like face rotation may require models more complex that takes into account the context of the application, and thus cannot be taken as state-of-the-art approach to image rotation.

Autoencoders and specifically convolutional autoencoders CAEs can also be used for image generation. [2]

Convolutional autoencoders

The CAE (Appendix.2) uses several convolutions, pooling and symmetric deconvolutional layers for image reconstruction purposes such as denoising, super-resolution, deblurring. The output in a CAE is often of the same dimensionality as the input, but the hidden layers have different dimensionalities. The encoder part tries to learn a new representation for the data which is the bottleneck part, this bottleneck is then fed to a decoder that outputs back to the original dimensionality.
The number, size and type of the layers needs to be determined depending on the application. But it usually follows a certain logic: the decoder part is symmetric to the encoder, max pooling layers often follow convolutional layers in the encoder part and Batch normalization or Upsampling is used after the deconvolutional layers in the decoder, the activation functions used are often relu for the hidden layers and sigmoid for the output layer but custom activation functions can be tested. The size of the filters increases with the depth of the encoder and decreases in the decoder.
The initial architecture is basic (such as the one used for the MNIST) and must be improved to reach the desired output. The search for the optimal model needs a test driven approach with multiples architectures tested following a certain logic (increasing complexity by increasing size of filter, reducing the training time by changing the batch size...).
The CAE may still not be enough to reach the desired output, some solutions such as Joint Auto-encoder can be explored.

Data Augmentation

Data augmentation can be used to avoid overfitting and artificially increase the amount of data [3].
There is no generalizable and safe augmentation policy, it is domain/context dependant. Generated images are expected to vary with the data input and thus require specific handling. In general, geometric and lighting augmentations are used such as flipping, rotation, color space, blurring and noise injection.
Generators models such as GANs and CAEs can also be used to do data augmentation by generating new input data, we can then mix those models to have a stronger architecture overall.

These state-of-the-art approaches are already quite robust for basic datasets such as MNIST, it's the context or domain of application that requires adding more complexity to the already existing models.

II.2. Image Denoising

Once our autoencoder was ready to produce its first outputs. We noticed that the results were showing some inaccuracy in the reproduction of the cars appearance. Some images were in fact blurry and suggested that our architecture wasn't robust enough. The problem was clearly due to the existence of some disturbing noise that jeopardizes our architecture. We then decided to use a denoising neural network architecture to try and fix these inconsistencies.
Denoising Autoencoders solve this problem by corrupting the data on purpose by randomly adding gaussian distributed variable. In general, the percentage of input nodes which are being set to zero is about 50%. Other sources suggest a lower count, such as 30%. It depends on the amount of data and input nodes you have. In order to apply this new architecture in our work, we looked up a variety of articles to finally settle for one very interesting paper **Deep Learning for Image Denoising: A Survey** [4].

After analysing these articles, we conclude that the denoising autoencoder can help perform two interesting features, starting with trying to encode the input and thus preserving important informations about the input, and trying to undo the effect of a corruption process applied to the input of the autoencoder. The latter can only be done by capturing the statistical dependencies between the inputs. On a technical level we used at first, an implementation of typical CNNs, then we proceeded to adding ReLu to the architecture in order to increase the computation speed for image denoising we got inspired from the DnCNN layer discussed in the paper to come up with a similar layer combination. We might add that this new denoising architecture would theoretically help us not only on image denoising but also in super resolution image tasks. We made, then, research on the net to understand the task of each kind of layer. In the encoder, a Conv2D creates a feature map that summarizes the presence of detected features in the input and the following MaxPool2D extracts sharp and smooth features. In the decoder, a Conv2DT generates a feature map from an input feature map with greater dimension to get from latent space representation to reconstructed image and the following Batch Normalization standardizes input layers which has a regularizing effect, reducing generalization error much like the use of activation

regularization and "*has an effect on the training of the model such as dramatically reducing the number of epochs required (...)*", according to *Deep Learning (2016)* by Ian Goodfellow, Yoshua Bengio & Aaron Courville.

II.3. Loss function & Metrics

After building and testing our first architectures, it turns out that the outputs represent the image visualized from the wanted point of view, but the quality of the image, especially the quality of the vehicles was not satisfying. And so we had two options to improve our model: either use a denoising process at the output of our model, or train our model using some more sophisticated loss functions. When it came to loss functions, we have looked up a large scope of articles presenting mathematical formulas for a new loss function such as **Loss Functions for Neural Networks For Image Processing.** [5].

Some selected approaches worked well with no complications for us. First the modified Mean Squared Error, where we put weights on the pixels corresponding to vehicles, this way when minimizing the loss function, the autoencoder would emphasize on minimizing the pixels representing vehicles, which would ideally lead to a better quality for the vehicles. A different approach was proposed to us by Theo, its an idea he encountered in this following article: Learning **Deformable Registration of Medical Images with Anatomical Constraints** [6]. This idea is one we did not encounter when doing our own research for loss functions. IT consists of building a new autoencoder structure, independent of our main structure. Train this new structure on our dataset using the classical Mean Square Error, and then define a new loss function between image 1 and image 2 which is equal to the Mean Square Error between the encoding of image 1 and image 2, meaning that the loss function which we would use for our main autoencoder would be equal to $MSE(\ Encodage(image_1)\ ,\ Encodage(image_2)\ )$. This idea makes sense, since it compares two images in the latent space, after extracting the most prominent features, which would make the comparison more meaningful. We have tried this approach on smaller datasets, the results depend a lot on the structure of the new autoencoder. There were some computational complications to generalize it to our problem, especially that this approach is one we started late in the process.

III. Description of the work

III.1. Perspective Transformation

In this section, we will focus on the change of angle itself, which includes: (i) the creation of the dataset from videos, (ii) the model to transform images from one angle to another and (iii) the model learning.

III.1.i. Dataset generation

The raw dataset provided is composed of several videos of different roads seen from three different angles (left, center, right). The raw dataset also includes a mask (to hide off-road

elements) and a pickel file containing information about the videos (including the time code of the passing vehicles).

From these videos, we extract only images where at least one vehicle is present thank to the pickel file, apply the mask and reduce the size of the images (from HD to 240*320), all thanks to the OpenCV library. The images are then saved in different folders depending on the type of road and angle. Then comes a verification step to make sure that there is the same number of images for each angle and that an image from a certain angle corresponds to an image from another angle. In the end, the dataset consists of 7383 images per angle.

In addition, YOLO is used to create a dataset containing vehicle positions for use in the custom MSE (see section III.3.i). For each image of the main dataset, an xml file is created which contains the position of the vehicles, the type of vehicle and the confidence probability of YOLO.

III.1.ii. Network Architecture Design

At the beginning, we hesitated between an architecture close to GAN or an auto-encoder (Generator AE to be precise, which is a Symmetric Generator Autoencoder). In the end, considering the difficulties to make a GAN converge during training, we started with an autoencoder to change the angle. The idea is to give as input an image from a specific angle (e.g. center), to pass it through an encoder to keep only the important information in the image. Then to use this encoded image in a decoder in order to reconstruct the image in another angle (for example left). To be able to build and train this kind of network, we used the TensorFlow 2.0 framework. For the network parameters we used the Adam optimizer, an MSE (Mean Square Error) loss function coupled with L2 loss to avoid potential instabilities during training:

$$Loss = MSE + \beta.L_2 \quad (with\ \beta\ a\ hyperparameter)$$

Appendix 3 and 4 show the chosen architectures for the encoder and decoder. To create them, we followed what was generally done for auto-encoders by adapting certain parameters according to the specificities of our problem and the results of our trainings. Thus, this Generator AE is the culmination of a selection on several architecture tests that we carried out throughout this project.

III.1.iii. Model Learning

First, we load the data via TensorFlow into a tf.data.dataset object, applying various methods to mix the images, separate them into batches (batch size of 8), divide into a train set and a validation set (5% validation). Second, we train the Generator AE with the parameters already seen above for about 40-50 epochs in Graph mode. Third, the evolution of the training between each epoch is recorded by recording one or two validation images. At the end, the change of angle of the test images and the loss curve can be recorded in order to check the good progress of the training.

In terms of results (from images not used during training), the road is well rendered and the vehicles are present. However these are blurred, probably due to the MSE loss function, and

sometimes to the wrong colour (white instead of red). In addition, dark vehicles (of a colour close to the road) are often less well reconstructed than light ones.

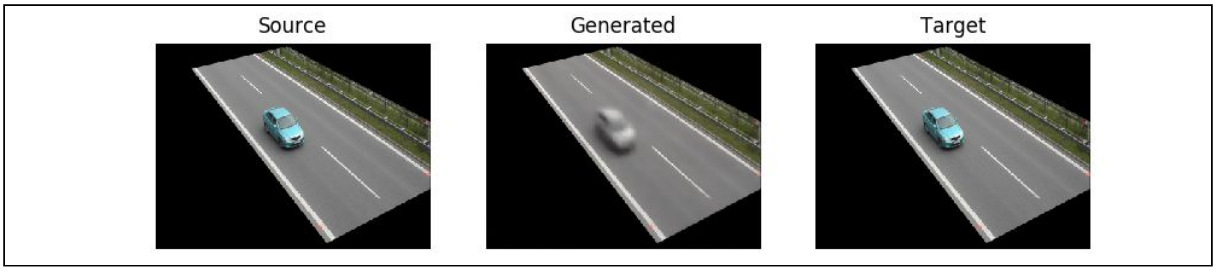


Fig.4 illustration of the colour problem

Finally, we have developed a model that can handle several angles at once (Multi Task Learning Generator AutoEncoder). To do this, we always use a common encoder, but one decoder for each angle, i.e. three decoders (the architectures remain the same). Thus, we find ourselves with three possible angles at the input and three at the output, that is nine possible combinations. All these combinations are used during training to make the model as robust as possible. Thus the new loss function is :

$$Loss = [MSE(decoder\ right) + MSE(decoder\ center) + MSE(decoder\ left)] / 3 + \beta.L_2$$

The results are similar to those from a single angle, however the vehicles appear a little fuzzier.

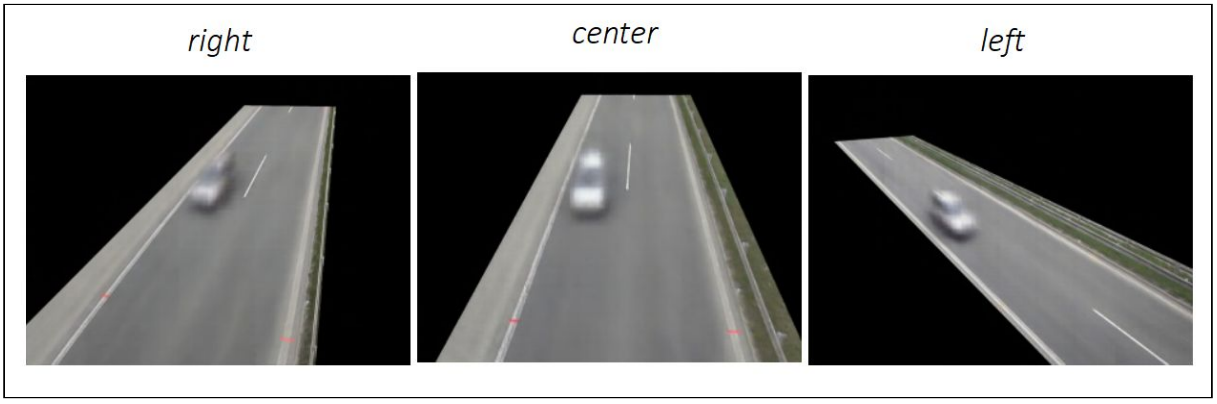


Fig.5 Generation of the three angles from the right view

Concerning training, losses stagnate around 40-50 epochs and training takes a few hours. There is an automatic saving of the best model during training (from the loss point of view on the validation data). You can see on the Appendix 5 the evolution of the loss function according to the number of epochs. You can see that the three decoders learn in similar ways, and that the model suffers very little overfitting. The difference between the decoder loss and the general test loss comes from the term ß*L2.

In the end, therefore, we are able to provide the customer with a ready-to-use dataset, the means to train the Generator AE and the already recorded weights directly usable. However, the results for the model with a single angle change are generally better (the vehicles are less blurred) than for the model with all three angle changes.

III.2. Image Denoising

In this section, we present a potential Deep Learning Denoising method we can use to improve our perspective-transformed images. Generally, training a deep model needs two major steps: (i) network architecture design and (ii) model learning from training data. For network architecture design, we are inspired by the generator. We modify it into a N-DAE which is a Denoising AutoEncoder with N blocks in Encoder and N blocks in Decoder (Appendix.6) to make it suitable for image denoising by choosing some fixed parameters.

III.2.i. Network Architecture Design

Appendix.6 illustrates the architecture of the proposed Denoiser to potentially improve the quality of generated images out of the generator. We have chosen to work on a symmetric Autoencoder :

- Encoder: N couples Convolutional Layer (Conv2D) - Max Pooling (MaxPool2D) & a final Convolutional Layer

- Decoder: N couples Transpose Convolutional Layer (Conv2DT) - Batch Normalization (BatchNorm) & a final Transpose Convolutional Layer with 3 channels to get colored output

Deep architecture:

For simplicity's sake, each encoder's Max Pooling has a size of (2,2) and each Conv2D has a stride of (2,x) with x= 2 if width size is odd and 1 otherwise. Every filters in Conv2Ds and Conv2DTs have the size (3,3) as we deem such parameter has no significant impact.
As for the Transpose Convolutional Layers, their strides are "symmetrical" to Convolutional Layers' strides since the k-th Conv2D has the same stride as the (N-k)-th Conv2DT's of the decoder for $k \leq N$ so the input shape and output shape are the same.

Relu function is the only function used in our Conv2Ds & Conv2DTs (except the final layer). Such function was chosen among others because of the non-saturation of its gradient which accelerates the convergence of stochastic gradient.

III.2.ii. Model Learning

The training dataset of our denoiser are made from K-SGAE generated images dataset and target dataset. We have chosen to work on two possible datasets to train our below model: complete dataset of generated images (D100%), tuned dataset made of half generated images and half gaussian-noised images from target dataset (D50%) and complete gaussian-noised dataset (D0%).

To build D50%, we take first half of generated images. Then, we take the second half of target images and add gaussian noise multiplied by coefficient uniformly distributed such as:

$$x_{noisy} = x_{target} + \lambda \cdot noise$$
$$with: \ \lambda \sim U(0, 0.2), \ noise \sim N(0, 1)$$

To build the D0%, we do the same process (Appendix.7) as above on complete target dataset. To build the D100%, we generate images from the generator.
Each dataset was made of 7383 images from which 5% is used as test set so the autoencoder doesn't give a look at it and from the lasting 95% of the dataset: 10% is a validation set (~9.5% dataset) and 90% (~85.5% dataset) is a training set.

To train the DAE, I train it for 50 epochs and our method consists of taking benefits from Keras CallBacks not to lose time for the training and we have a precise monitor for that: Validation Loss which represents the Generalization Error. So, we have used Early Stopping with a patience of 5 to stop training if for 5 consecutive epochs, validation loss has stopped decreasing. Also, we have exploited ReduceLROnPlateau with a patience of 3 and a factor of 0.2 to reduce learning rate by multiplying it by 0.2 if for 3 consecutive epochs, validation loss has stopped decreasing. Finally, to save the best model for a fixed architecture out of the 50 epochs of training, we have used a checkpointer which saves the model minimizing the monitor. Also, other options, we have tried to prevent overfitting were adding Dropout Layers with different Dropout rates (0.2, 0.3, 0.4 & 0.5) but this was ineffective and we have tried to add some fully connected layers which were, as expected, very computationally expensive to the the point that making 2 or 3 blocks with $f_o \leq 64$ was raising memory error so we gave up on the idea which is not even so much adapted to identify images patterns.

Some of our previous ideas were skipped, we omitted the completely noised dataset as it was working well for images with Gaussian noise but not our blurred images. So, we decided to work only with half-noised dataset for the rest of the experience. Also, for the loss function, we tried mixed loss with form $\alpha.Loss_1 + (1-\alpha).Loss_2$ with MSE, custom MSE with bigger weights on cars and even, MAE as $Loss_1$ or $Loss_2$. But none of the results were conclusive. Then, we have exclusively worked with MSE.

During the time we had, we could scrutinize the effect of the following hyperparameters: Batch Size, f_0 (first layer's number of filters) & N (number of couples) on the reconstruction process on output images from the Generator AE. The main point of this is to find the best tradeoff between these three to minimize the Validation Loss.

First, we study the effect of f_o on the denoising process by choosing f_o from 8 to 1024 (until we reach error memory). As for the Batch Size for the training, throughout more than 50 tested architectures, the best Batch Size remains 4 for each trained model when we tried the following Batch Sizes : 4,8,16,32 (until run out of memory). Then, I take a fixed Batch Size = 4 and all the N values I studied and I vary f_o so I can get this very comparison below for N=6 :

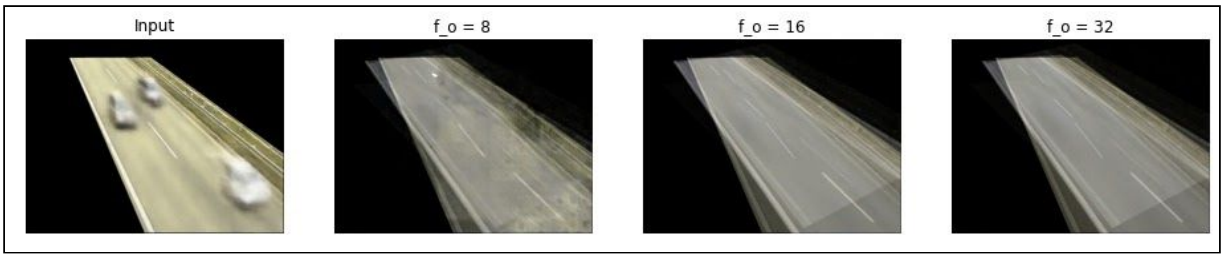


Fig.6 Effect of f_o on test image for N=6 and Batch Size = 4

It is clear that increasing f_o improves the image quality of the output but not the quality of vehicles and the road is not straight anymore. Later, we have trained a N-SDAE for N from 2 to 6 with fixed Batch Size and the optimal f_o for each N, which is called ${f_{max}}^N$.

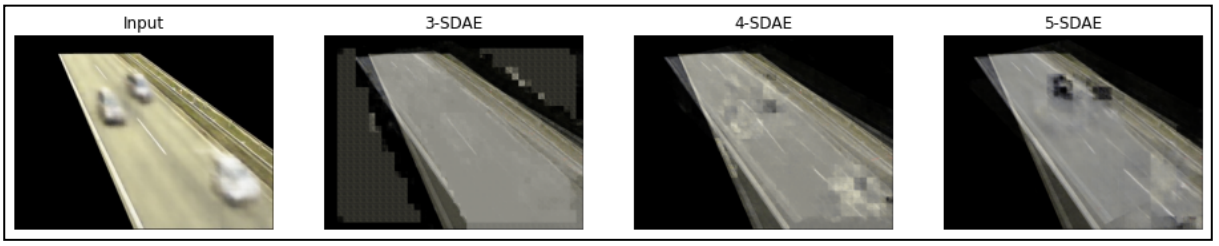


Fig.7 Effect of N on test image for Batch Size = 4 and $f_o = {f_{max}}^N$

Here, it is visually obvious that improving N increases the road quality and some blurred surfaces on the positions where cars are supposed to be.

Thanks to many attempts, we found that N = 6 and f_o = 32 for Batch Size = 4 is the optimal model. The best model for this very architecture is reached at epoch = 34 (Appendix.8). The validation loss has an order of magnitude of 10^{-4} while training loss is about 10^{-2}. Among the reasons one can stress out, the validation set could be easier to denoise (it is not the case here since it is built randomly) but we concluded that it might be due to the fact this loss is computed at the end of an epoch where the model learned sufficiently well to have such performance on validation test.
Unfortunately, all architectures taken into account, the DAE is not adequate and we can't consider that such structure is apt to detect patterns in the image and it considers cars as a noise and worse, don't keep the road as well-oriented as it should be. Indeed, even the best architecture gives bad results (Appendix.9) where cars are completely erased. This very work doesn't correct the outputs of the generator but has contributed to the deliverable in the way that the client knows that denoising is not useful to solve our problem and other ways have to be scrutinized. With a further analysis, we consider that deblurring would have been more interesting. In fact, we could have applied an average filter to blur the images of the input dataset or apply such filter only on the surface where we can find the vehicle(s). Moreover, we could have worked on cropped versions of the image where we can focus only more on cars. Also, we could have added few random cars images in the dataset to make the DAE more able to identify the presence of cars on generated images. Once, this part was over, we understood that we needed to check another element which can influence our generator

results. The following part will, in part, introduce a way to study the performance of our Autoencoders and possible loss function to exploit for the training.

III.3. Loss function & Metric

Perhaps one of the most crucial parts of learning the model and testing it is a well-chosen loss function to train our architecture and a metric to assess our work.

III.3.i. Loss functions

At first, we have trained our autoencoder model based on the classical MSE (Mean squared error), the results were acceptable in terms of recognizing that generally the angle of view has been changed, but the quality of the outputs, especially the quality of the vehicles on the output was not very satisfying. From there on, we had two possible roads to explore, one of which is proposing some new loss functions to train our model, and the other one being improving the quality of our outputs using denoising autoencoders or other denoising strategies. When it came to proposing new loss functions, we have spent some time looking for some mathematical formulas for loss functions on images, but we ended up not using them, because they were very general and not specific to our project. Some other approaches made the cut. The first approach was to modify the basic MSE by introducing weights for the pixels representing the vehicles, this way when minimizing the loss function, we will give more importance to the vehicles pixels, and therefore the output vehicles would be of better quality. In addition, we also tested an NCC (Normalized Cross Correlation) loss function using convolution to compare two images: unfortunately the results were not conclusive. Another approach which was proposed a little late in the process, is that of considering a loss function which would be equal to the MSE at the output of the encoding part of an autoencoder. Intuitively, this is in hopes of comparing two images in the latent space where more critical and decisive features are concerned by the comparison. This idea was inspired by an article about "Learning Deformable Registration of Medical Images", from which the process graphic (Appendix.10) is taken.

This graphic (Appendix.10) proposes computing the loss function at the output of an encoder. And so what we do concretely, is build an new autoencoder, independent of the main one (the one giving predictions), we train this autoencoder using our dataset. We then define a new loss function which is the MSE at the output of the encoding part of our new architecture. The sole purpose of this autoencoder is to produce the new loss function through its Latent Space, it is not directly involved in our final prediction. We have first tried to code this process on some more basic datasets found on the Internet, just to get a grasp of the commands. The only challenging part is accessing the output of an intermediary layer (the encoding). The results were not always satisfying, we believe that this depends on the architecture we chose for our new autoencoder. As for our dataset, we have decided to copy the architecture of our (main) predicting autoencoder for our new autoencoder, we had some problems running the code, we believe this is due to dimension problems. We didn't succeed in finalizing this approach in the remaining time. To conclude as per this approach, it makes perfect sense why this approach would work, but its complexity (building a new architecture) as well as the fact that we started it quite late in the process were obstacles for us to use it.

III.3.ii Metrics

As for the metrics to assess our work, we have decided to use the ability to detect vehicles in our output as a metric of success. Initially we had used a customized YOLO for object detection where threshold is smaller to recognize moving objects more often in our videos. We have limited the classes of identifiable objects to include only: cars, trucks, and motorcycles, and since we had difficulties detecting vehicles viewed from center, we have decided to lower the threshold for detecting vehicles.

$$metric = \frac{number\ of\ recognized\ vehicles\ on\ the\ generated\ images}{number\ of\ recognized\ vehicles\ on\ the\ initial\ images}$$

But even with these changes, identifying vehicles by YOLO in our outputs did not yield very satisfying results, as the outputs were too blurry, which shows the importance of the deblurring process. The problem is that YOLO can't recognize the presence of vehicles in images out of the DAE while it recognizes 5% vehicles out of the Generator AE, which happens to be inefficient as it denoises and not lower blurr. This very aspect of our project has contributed as it gives the client a possible idea to exploit in the future.

IV. Conclusion

Finally, we have worked on three different autoencoders for three differents tasks. the perspective transformer, the denoising autoencoder and the loss encoder.

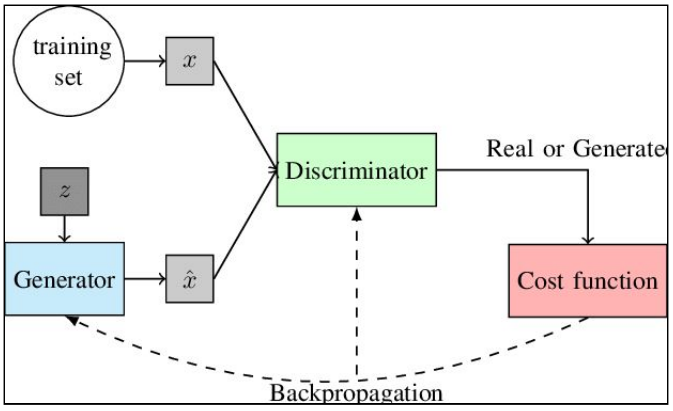
In terms of results, we succeeded in carrying out the task we were asked to do, i.e. the change of angle. To be exact, we are able to change the perspective of an image from three defined angles. However, this success must be qualified by the fact that the vehicles (elements having the most importance on our dataset), remain too blurred to be detected by object detection. The track on denoising, although promising but unfortunately not having borne fruit, could be replaced by deblurring to try to fully meet the objectives we have been given and on a more advanced technical aspect, it might quite interesting to look at how the encoder loss could affect the results. Also, to generalize the task of such autoencoder, one can use transfer learning to benefit from the knowledge gained while learning how to change roads + cars image perspective as it can apply to change other images perspectives.

Through this long-term project, we have had an opportunity to learn how to code a Deep Learning model in Tensorflow/Keras on Python, process images & videos in CV2/PIL Image and work on a technical project in team with more experienced researchers. Also, for all of us, this was our first experience working on a State-of-the-art Research Project and we found it particularly exciting and educational to the point that we have become sure to work in Data Science & Artificial Intelligence. Furthermore, throughout the project, we have deepened our knowledges in Machine Learning methods such as Hyperparameters Tuning, the importance of quantifying an Artificial Intelligence model performance and having a significant metric to do so.

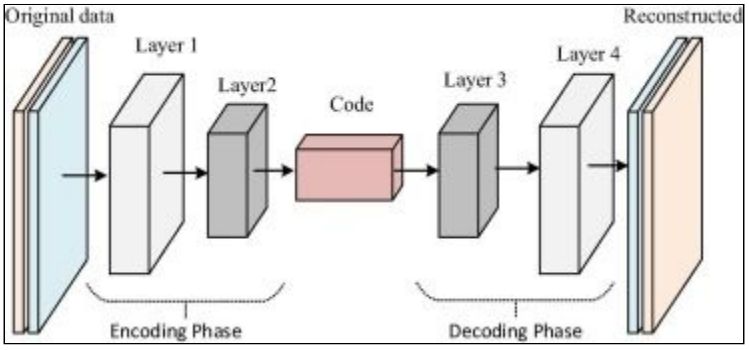
For the client, we have been a considerable help in the exercise of reconciling predictions from multiple angles. Indeed, we have trained multiple symmetric autoencoders whether it is

to transform perspective or denoise images so he can have a report of dysfunctional and functional models, which is a major boost to progress on the topics later with other teams. Moreover, we have build easily usable models to change colored images perspective from one angle to another with three different choices (Left, Center, Right). In the end, we have written commented codes and exploit documented ideas on Gitlab to make the client experience using our work as comfortable as possible.

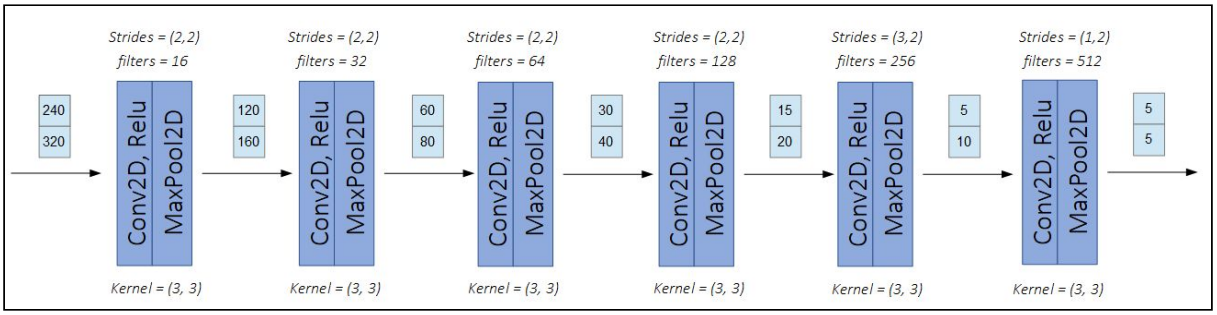
V. Appendix



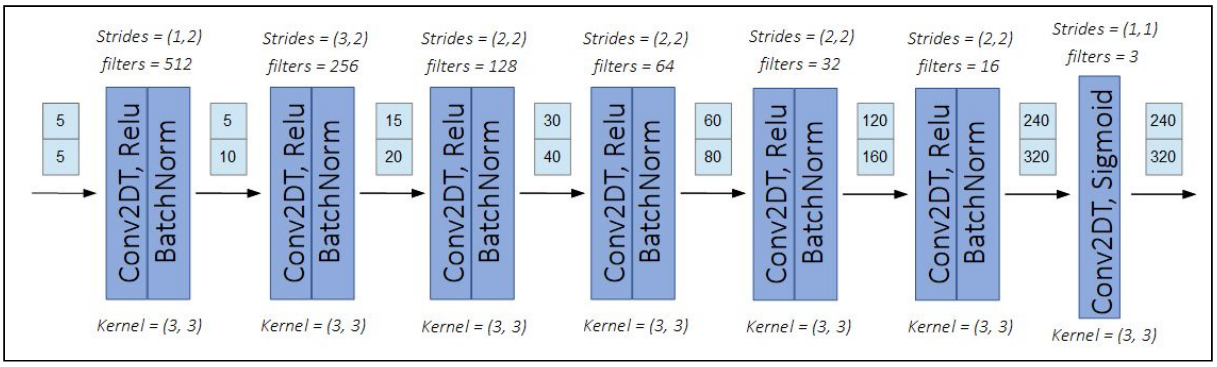
Appendix.1 GAN Process
(fig 15 from [Generative Adversarial Networks Framework.](#))



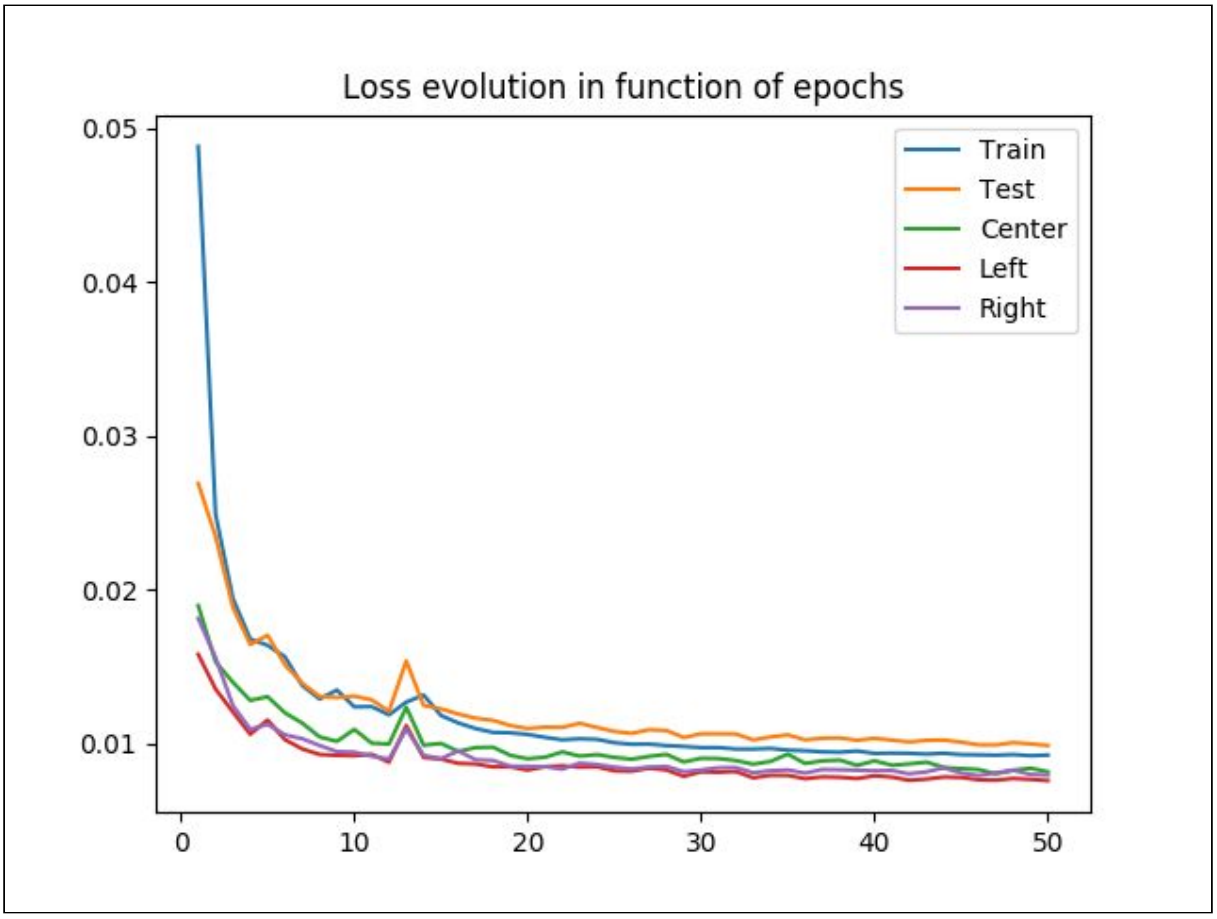
Appendix.2 Architecture of a CAE
(fig 2 from [https://www.sciencedirect.com/science/article/pii/S1389041718302730](#))



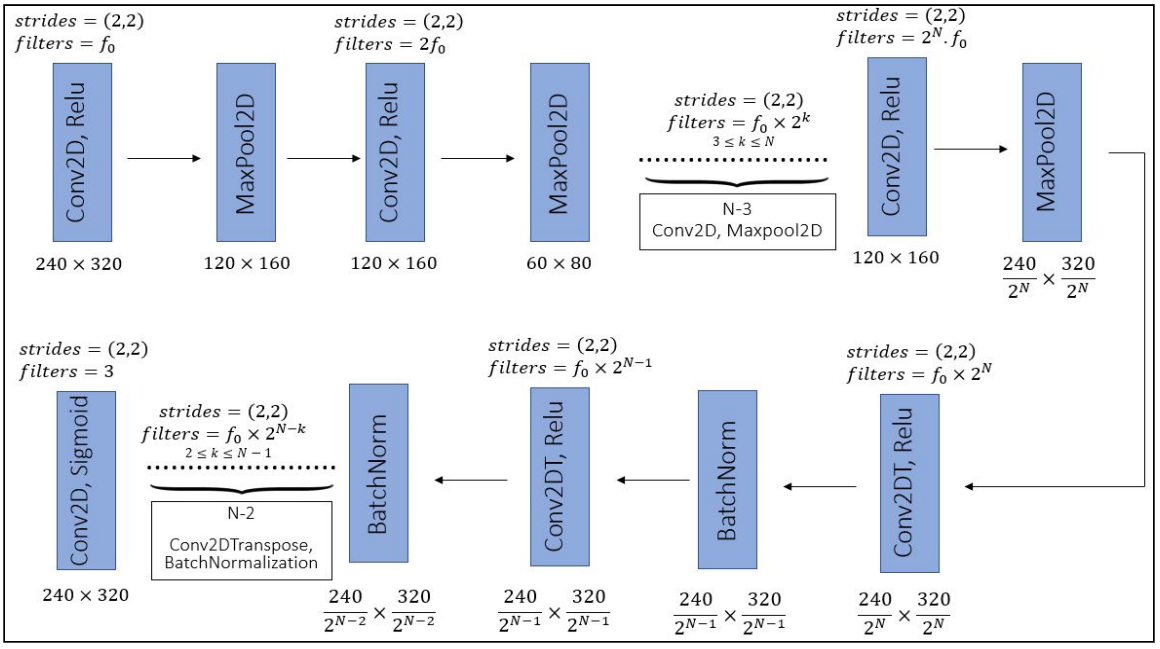
Appendix.3 Encoder of the Generator AE



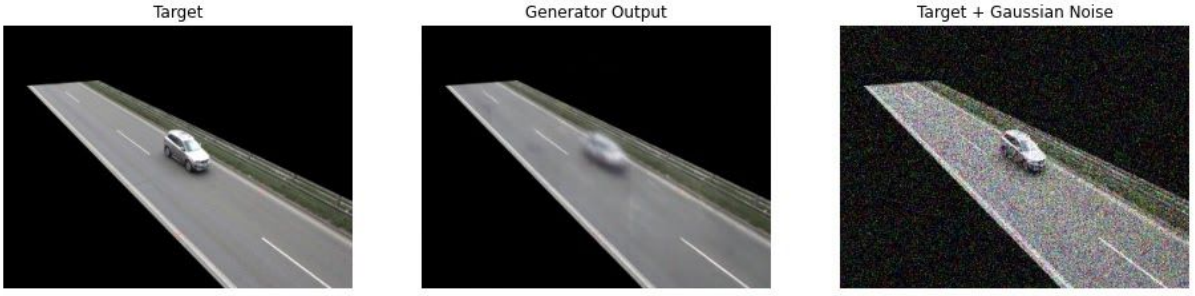
Appendix.4 Decoder of the Generator AE



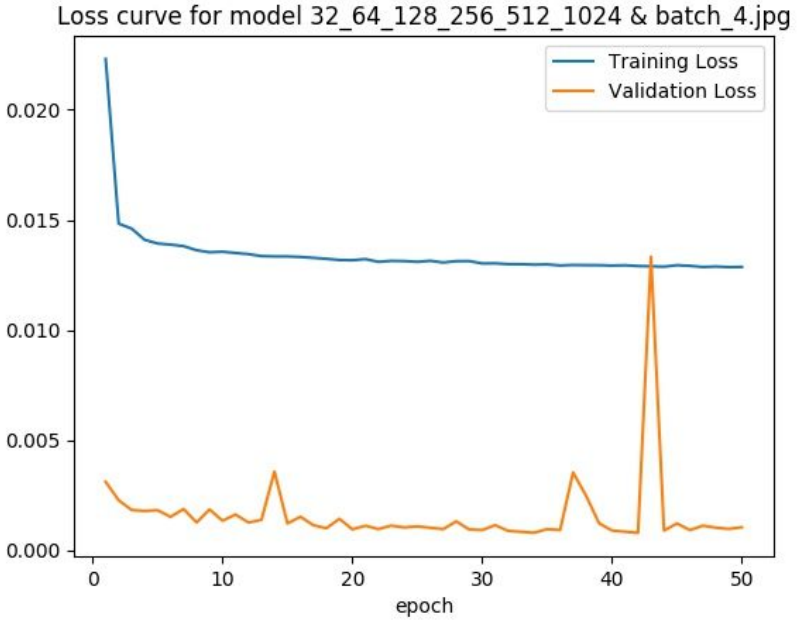
Appendix.5 Loss curve (batch size 8, epochs 50, Beta 0.000005



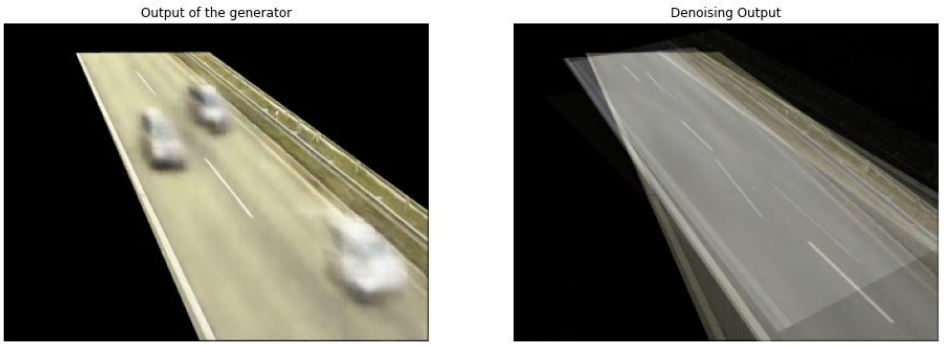
Appendix.6 Architecture of the DAE



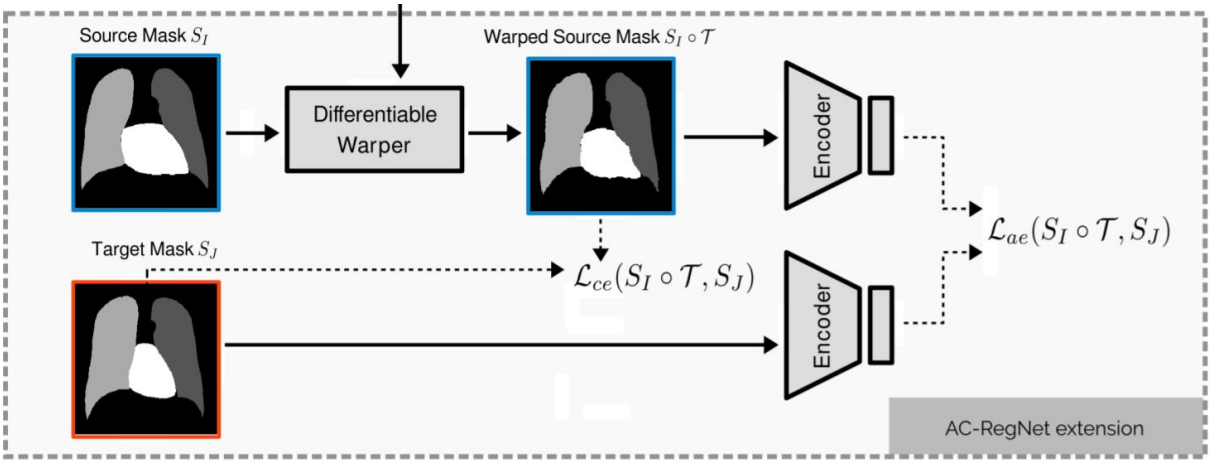
Appendix.7 Comparison between target image, generated image and noisy version of the target



Appendix.8 Loss curve for N = 6, f_o = 32, Batch Size = 32 during 50 epochs



Appendix.9 Impact of the optimal DAE



Appendix.10 Process to compute the loss function
(Fig.1 one from https://arxiv.org/pdf/2001.07183.pdf)

VI. Bibliography

- [1] *A Survey of State-of-the-Art Ganbased Approaches to Image Synthesis* ; Shirin Nasr Esfahani and Shahram Latifi ; https://pdfs.semanticscholar.org/eda0/6ef7d3938b343d3985584f1954a5bd5c88c3.pdf

- [2] *Image Restoration Using Convolutional Auto-encoders with Symmetric Skip Connections* ; Xiao-Jiao Mao, Chunhua Shen, Yu-Bin Yang ; https://arxiv.org/pdf/1606.08921.pdf

- [3] *A survey on Image Data Augmentation for Deep Learning* ; Connor Shorten & Taghi M. Khoshgoftaar ; https://link.springer.com/article/10.1186/s40537-019-0197-0

- [4] *Deep Learning for Image Denoising: A Survey* ; Chunwei Tian, Yong Xu, Lunke Fei, and Ke Yan ; https://arxiv.org/pdf/1810.05052.pdf

- [5] *Loss Functions for Neural Networks for Image Processing* ; Hang Zhao, Orazio Gallo, Iuri Frosio, Jan Kautz ; https://arxiv.org/abs/1511.08861

- [6] *Learning Deformable Registration of Medical Images with Anatomical Constraints* ; Lucas Mansilla, Diego H. Milone, Enzo Ferrante ; https://arxiv.org/pdf/2001.07183.pdf