

Assisted product design with SinGAN

Mahdi KALLEL: mahdi.kallel@ip-paris.fr | Yassine NAJI : yassine.naji@ip-paris.fr

Abstract

In this project we offer an assisted design module leveraging single image generative networks (SinGAN) for assisted car design. Lifting the burden of passing from sketch to concept and helping designers draw inspiration from previous models.

1. Introduction

GANs are a type of generative networks that can produce realistic images from a latent vector, which usually consists of a random noise. Typically, a GAN consist of two networks: A generator (G) whose purpose is to map latent code to images and discriminator (D) whose task is to evaluate if an whether if the image is real (from the training dataset) or fake (generated by (G)).

Nowadays generative models are able to generate realistic new images, such architecture require huge amounts of data (precise) and take a long duration of training before yielding satisfactory results.

In this work, we seek to investigate the use of SinGAN in the process of assisted car design. SinGAN is a good candidate for this mission as it offers relatively fast training and allows the user to hand pick the image(s) from which to draw inspiration.

1.1. Architecture

The main contribution of this paper is its training method for generative modelling by using a single training image. This is allowed by using pyramid of fully convolutional GANs capturing the patch distribution at different scales.

Both the generator and discriminator consist of stacked (Convolution, BatchNorm, LeakyRelu). There's no fully connected layer at the output of the discriminator.

This fully convolutional architecture allows to produce images from various shapes.

1.2. Training

We define a scale as a Generator / Discriminator couple. At a scale level, the generator is fed with the random noise along with the generated image from the generator below them (except for first, which is fed with random noise only) and is responsible for generating an image larger by a factor of R (usually 4/3). The source image is downsampled to the

respective sizes and fed into the discriminator along with the generator's output.

Each level is trained for N_{steps} each step consists of one back-propagation pass through the scale.

The different levels are responsible for generating different kind of details in the image, with the lowest level able to generate coarser details and the higher levels able to manipulate more finer details.

1.3. Paint to image:

We will be using SinGAN's in the paint to image use case. This is done by pre-training the network on a "real image", then we feed a painting into one of the many scales of the network. In order for this to work, the painting must have the same semantic / color association than the original image.

2. New objective:

In section 1.3 we saw that SinGAN trains a generative model by considering patches of images as data points and using the dual formulation of optimal transport to compute a distance between the datasets of real and fake patches.

The original formulation of optimal transport (OT) cost is that we are given two sets of points A and B and a cost matrix for the transport of each point from A to B. And we want to find the coupling that minimizes our overall cost. The cost of such a coupling is the **optimal transport cost** C_t .

Finding this coupling is computationally expensive to solve (in the order of N^3). This is why most works prefer working with it's dual which offers an easier to solve approximation.

However, recent works [5] show that we can efficiently solve a regularized version of it. By adding an entropy penalty to the objective, we restrict our search space to couplings having the "Scaling form". This form can be efficiently solved using the Sinkhorn algorithm.

2.1. Implementation:

Using the SinkHorn loss required a few modifications to the network:

Instead of returning a single value for each patch, the



Figure 1. Illustration of paint to image.

model now returns N_{embed} values for each patch. These values represent embeddings of the patches in a latent space.

These embeddings are then used to compute the cost matrix which is the the distance (in the embedding space) of each pair of patches in the "real" and "fake image". This cost matrix is then fed to the Sinkhorn algorithm to find the best coupling and thus the OT distance.

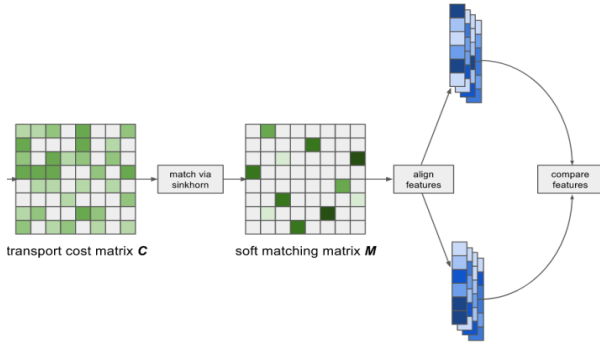


Figure 2. Illustration of OT cost computation.

2.2. Cost matrix:

The cost matrix : $(C)_{ij} = \mathcal{L}(p_i, p_j)$ where p_i stands for patch i . As in 2 the distance is computed using the cosine distance between the patch embeddings of the discriminator.

To this term we experiment adding a pixel wise penalty term. It's aim is to encourage the transport between neighboring patches. We believe this will encourage more cohesion in the generated images.

$$(C)_{ij} = 1 - \frac{\nu(p_i) \cdot \nu(p_j)}{\|\nu(p_i)\| \cdot \|\nu(p_j)\|} + \frac{1}{2} \left(\frac{x_{p_j} - x_{p_i}}{\max(X)} + \frac{y_{p_j} - y_{p_i}}{\max(Y)} \right)$$

3. Automatic painting generation:

Creating at each time a new sketch for testing might be demanding, instead, we can get inspiration from existing car models to create sketches. In this section our objective

is to generate a new car inspired from two distinct cars. For that, we first had to generate a sketch from a real car image.

3.1. The network:

We designed a segmentation model based on FCN (fully convolutional network) [3], that we trained on a subset of PASCAL-Part Dataset, which is a dataset that contains the same images as Pascal VOC but with annotations for objects parts [4].

We use a pretrained FCN to delete the background, in order to focus on learning cars parts (**body , windows , headlights,wheels, mirrors and doors**). We also processed the annotations in order to keep only labels relative to cars parts then we trained our model on 1652 samples (and validate on 183) for 50 epochs using Adam optimizer with a decreasing learning rate starting from 10^{-4} . The figure 3 shows the segmentation output.



Figure 3. car parts segmentation

The next step is to generate a sketch based on the output of the segmentation but with the same color and orientation characteristics of the SinGAN training image. For this we perform segmentation on training image also and learn the colors of each part. Then we color our original sketch with the same colors as the training image. We can also predict the pose of the cars by identifying the position on the wheels. For example, using the training image (figure 7), we generate paint (figure 9). We summarize the whole procedure in the schema 6.

3.2. Results and discussion:

As we experimented 'paint to image' for sanity checks, we noticed that colors in the sketch seems to indicate the lo-

cation of features with the same color in the training image. Thus, in order to generate realistic car images, it's mandatory that the sketch matches (approximately) the same color distribution of the training image. In the case we are using segmentation for generating sketches, this task is relatively easy to preform. In fact, we can do segmentation to the training image and learn colors over different parts and then color the same parts in the sketch with the same colors (figures 7, 9). By using such painting as input for the model we get results similar to 4.



Figure 4. Output of singan, with 7 for train and 9 for input.

Unfortunately, even if our segmentation is precise enough to do color transfer for some parts (Wheels, windows, etc), but their are lacking cars parts which our segmentation model wasn't trained on (like the bumper for example), so the SinGAN can't cast those features in the sketch. Therefore we decided to pursue our experiments using hand picked paintings.

4. Generating car images:

4.1. Experiments :

SinGAN is able to capture texture information and even add new instances of objects in it's generated images.

The features extraction is well done by SinGAN due to it's hierarchical structures using discriminators which have access to image patches only (those patches are defined by the receptive filed of the discriminator output pixels) which allows generators to learn features of different sizes.

On the other hand, generating a car is quite a different task. Our main objective is to be able to match training car features with the sketch all while keeping a coherent image of a car. Which means that our model should learn car parts and try to cast them correctly in the sketch. From the figure 5 we see that, in the usual "Generate from noise" mode, it's impossible for SinGAN to reconstruct a car.

The images in higher scales being bases on the previous one, we can see that this problem starts from the initial scale who we couldn't get to generate coherent images of cars.

From figure 5 we see that with the "Paint image" as input to one of the scales the model is able to pick up the global structures and successfully add details.

4.2. Fine tuning SinGAN for cars:

On average training SinGAN (on a 250x150 image) for 7 scales using GPU requires 20 minutes. Generating images whether that be from random noise or a painting is quasy instantaneous.

In order to establish the best SinGAN parameters for our task, we fixed a single training image (figure 7) and vector image (figure 8) as a painting. In order to compare the outputs of SinGAN objectively rather than visually. We came up with a new metric using the SIFID score [1] which computes the Frechet inception distance between the features of two images. Let's denote our metric by M, we have:

$$M(output) = S_p + S_t + |S_p - S_t| \quad (1)$$

Where:

$$S_t = SIFID(output, training\ image)$$

$$S_p = SIFID(output, paint)$$

We choose this metric because our aim is to minimize both SIFID scores S_1 and S_2 but without overfitting on one single image (this is captured by the term $|S_1 - S_2|$).

Since we focused on learning the features of the training images (cars parts) more than the global structure. We down scaled the image into 44x44 instead of 25x25 as in the original paper. Which allowed us to train the SinGAN for only 5 scales instead of 8. This made the training faster but didn't improve much the results. We also modified the generator convolutions kernel sizes in other to change the receptive field (and therefore modify the size of patches that the discriminators see), we found that the default parameter kernel size = 3 is optimal for our task. However, we noticed that increasing the number of steps per scale to 2500 (default 2000) improved slightly the results with respect to our metric. For each of the previous experiments we set all parameters to default except the target one. We summarized our best experiments in the table 1 .

5. Conclusion:

In this work we studied deeply the working of SinGAN's, we implement a new paradigm for training such network using the SinkHorn algorithm. We implemented, from scratch, a module to generate paintings to be fed to the network from real car images. We studied fine tuning the network for assisted car design. The quality of the generated images was assesed using a variation of the SIFID metric.

In conclusion, we found out that the original SinGAN parametrs were best suited for this task. Using Sinkhorn algorithm yields quite close results. Car segmentation needs some improvement to include finer parts of the cars in order to yield result similar to hand picked paintings.

References

- [1] **SinGAN: Learning a Generative Model from a Single Natural Image** , Tamar Rott Shaham, Tali Dekel, Tomer Michaeli [3](#), [4](#)
- [2] **Our code** : <https://github.com/kallel-mahdi/SinGAN2>
- [3] **Fully Convolutional Networks for Semantic Segmentation** , Jonathan Long, Evan Shelhamer, Trevor Darrell
- [4] **Link to PASCAL-Part Dataset**: <http://roozbehm.info/pascal-parts/pascal-parts.html>
- [5] **Learning Generative Models with Sinkhorn Divergences** Aude Genevay, Gabriel Peyré, Marco Cuturi [2](#), [4](#)
- [6] **Improving GANs using optimal transport**: <https://arxiv.org/abs/1803.05573>
- [7] **SinkHorn code**: <https://github.com/audeg/Sinkhorn-GAN> [2](#), [4](#)
- [8] **SinGAN code**: <https://github.com/jonasgrebe/pt-singan-single-image-gan> [1](#)

6. Work repartition :

Mahdi KALLEL : Borrowed the code for SinGan from [8](#) (we found it to be a clearer version than original implementation). We had to modify the code to work more like the original (Multiple passes for the discriminator ..). We then integrated the code for the Sinkhorn algorithm from [7](#) to train the network. And implemented the pixel wise penalty term. Implemented a script to train SinGAN on multiple images but the results were too bad to report.

Yassine NAJI : Worked on the automatic painting generation using segmentation. Adapted the Pascal Parts Dataset [\[4\]](#), and fine tuned the segmentation model FCN [\[3\]](#) to generate semantic labels for cars parts. Implemented the algorithm for generating painting from semantic labels. Worked on the SinGAN hyper-parameters tuning and designing the evaluation metric based on SIFID [\[1\]](#).







Hyper parameters	Best start scale / Total num of scales	SIFID wrt train image	SIFID wrt painting	Custom metric 1	Best results
Number of steps per scale = 2500 (default 2000)	5/7	0.38	0.43	0.86	
Defaults SinGAN parameters	5/7	0.44	0.34	0.88	
scale factor = 0.9 (default 0.75)	14/17	0.38	0.43	0.90	
Using Sinkhorn loss	7/7	0.12	0.59	1.06	
Training without gradient penalty	5/7	0.52	0.13	1.04	
Discriminator convolution kernel size = 4 (default 3) the number of scales = 4	4/4	0.64	0.43	1.28	

Table 1. Experiments

7. Appendix original Loss functions

As we are dealing with a single image, we consider the patches of the training image as our dataset. This is done by setting the last convolution of the discriminator to have only "ONE" channel and considering it as our discriminating function.

- **Adversarial loss:** WGAN is used for a stabilized training where final discrimination score is the average over the patch discrimination map. Also the loss is defined over the whole image rather than over random crops.

This loss comes in different flavors, in the original paper the authors use the dual objective of this optimal transport loss:

$$W(P_r, P_g) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim P_r} [f(x)] - \mathbb{E}_{z \sim Z} [f(g_\theta(z))]$$

Where $\|f\|_L \leq 1$ stands 1-Lipschitz continuous functions. The constraint is enforced by a **Gradient normalisation** term in the generator loss.

- **Reconstruction loss:** Ensures the existence of noise that can be used to generate original training image. So basically simultaneous to what we are have been doing(adding noises at various levels as we have seen till now) we try to see whether the initial input noise can be used to generate the original image.

To do so, we are gonna take zero noise in the later levels. After this, we just calculate the squared difference loss between the generations at each level(with zero noise being assigned to every level except the lowest) and the image at the size scale. This also serves another purpose to determine standard deviation of noise Z_n at each scale, as we take the std to be proportional to RMSE between one-step upsampled version of generated x_{n+1}^{rec} and x_n , giving an indication of amount of information to be added at each scale(which is added as noise).



Figure 7. SinGAN training image



Figure 8. Car painting (used for image generation and fine tuning)



Figure 9. Example of painting obtained by segmenting an image of car.

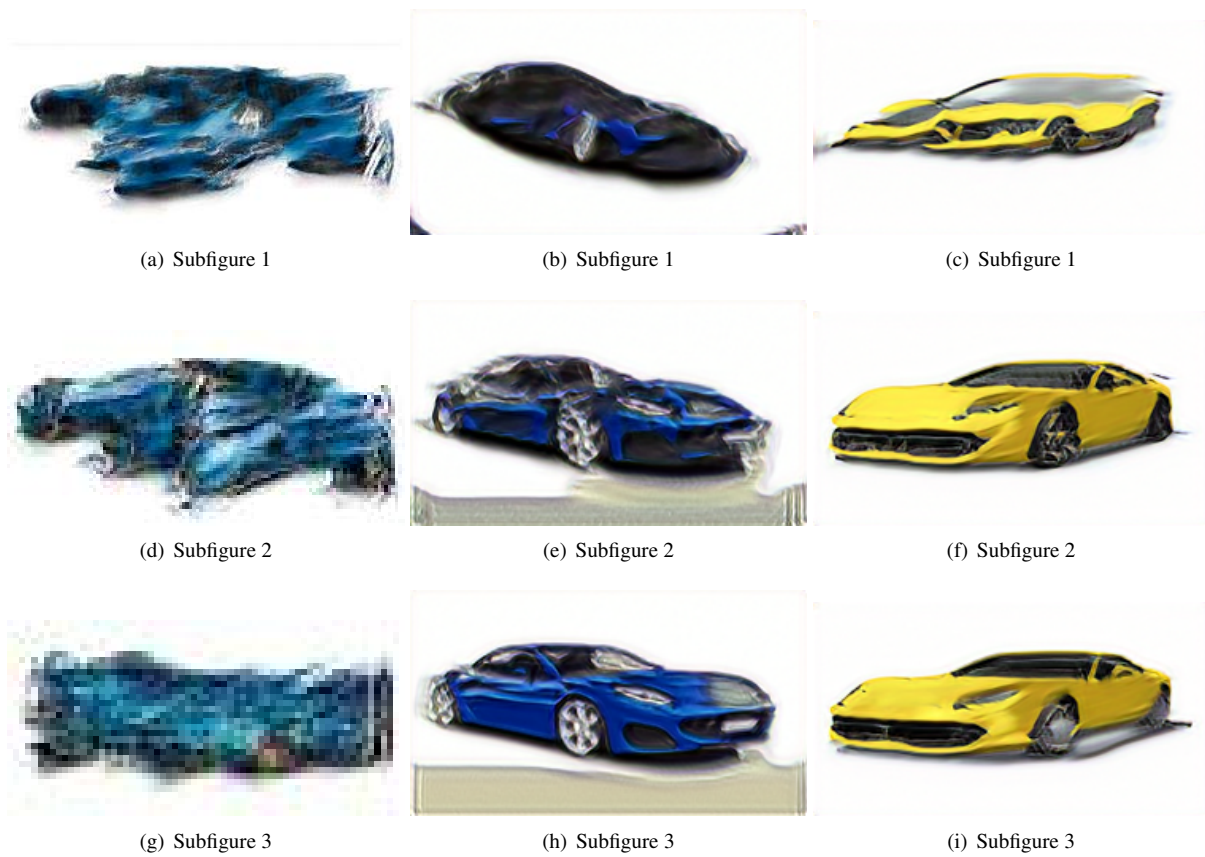


Figure 5. Images generated : From left to right : 1. Random noise output, 2. Output when feeding paint 8 , 3. Output when trained on another image.

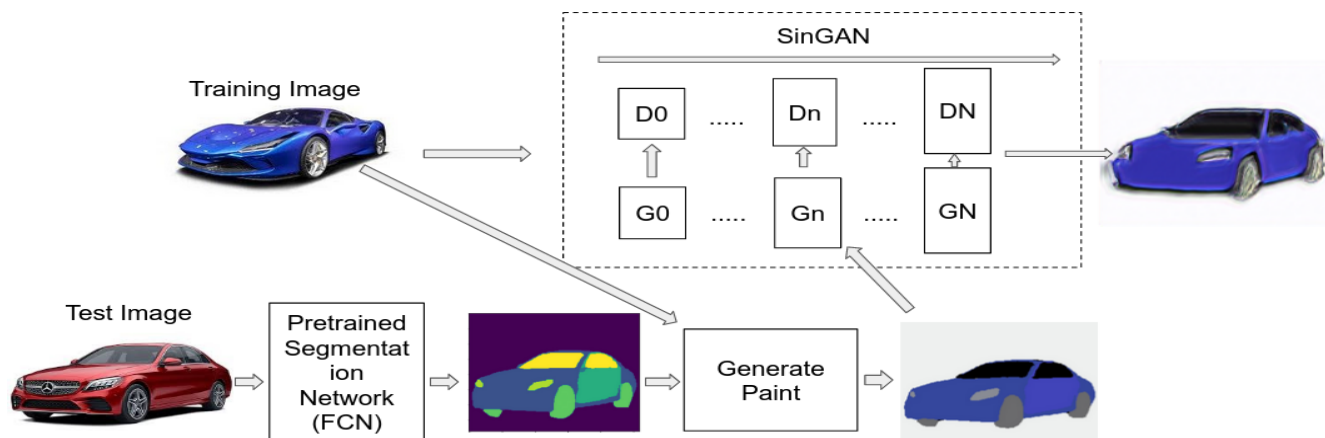


Figure 6. Pipeline of SinGAN to generate car sketch