PaintsTorch: a User-Guided Anime Line Art Colorization Tool with Double Generator Conditional Adversarial Network

Yliess Hati, Gregor Jouet, Francis Rousseaux, Clément Duhart

PaintsTorch: a User-Guided Anime Line Art Colorization Tool with Double Generator Conditional Adversarial Network

Yliess HATI
yliess.hati@devinci.fr
Pole Universitaire Leonard de Vinci, Research Center
La Defense, France

Gregor JOUET
gregor@adaltas.com
ADALTAS
Meudon, France

Francis ROUSSEAUX
francis.rousseaux@univ-reims.fr
URCA CReSTIC, Moulin de la Housse
Reims, France

Clement DUHART
duhart@mit.edu
MIT MediaLab, Responsive Environments Group
Cambridge, USA

ABSTRACT

The lack of information provided by line arts makes user guided-colorization a challenging task for computer vision. Recent contributions from the deep learning community based on Generative Adversarial Network (GAN) have shown incredible results compared to previous techniques. These methods employ user input color hints as a way to condition the network. The current state of the art has shown the ability to generalize and generate realistic and precise colorization by introducing a custom dataset and a new model with its training pipeline. Nevertheless, their approach relies on randomly sampled pixels as color hints for training. Thus, in this contribution, we introduce a stroke simulation based approach for hint generation, making the model more robust to messy inputs. We also propose a new cleaner dataset, and explore the use of a double generator GAN to improve visual fidelity.

CCS CONCEPTS

• **Computing methodologies** → **Reconstruction**; **Image processing**; • **Applied computing** → **Media arts**; • **Human-centered computing** → *User studies.*

KEYWORDS

deep learning, neural networks, generative adversarial network, user-guided colorization, datasets

ACM Reference Format:
Yliess HATI, Gregor JOUET, Francis ROUSSEAUX, and Clement DUHART. 2019. PaintsTorch: a User-Guided Anime Line Art Colorization Tool with Double Generator Conditional Adversarial Network. In *European Conference on Visual Media Production (CVMP '19), December 17–18, 2019, London, United Kingdom.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3359998.3369401

CVMP '19, December 17–18, 2019, London, United Kingdom

ACM ISBN 978-1-4503-7003-5/19/12…$15.00
https://doi.org/10.1145/3359998.3369401

Figure 1: PaintsTorch guided colorization on line art. PaintsTorch takes two inputs: a grayscale lineart and a color hint. It outputs a colored illustration following the color hints and prior knowledge learned from a custom illustration dataset.

DISCLAIMER

The illustrations shown in this paper belong to their respective owners and are used purely for academic and research purposes. Some content may hurt the sensibility of the reader, but the images shown are representative of the dataset and communities they originate from.

1 INTRODUCTION

Line Art colorization plays a critical part in the artists, illustrators and animators work. The task is labor intensive, redundant, and exhaustive, especially in the animation industry, where artists have to colorize every frame of the animated product. The process is often executed by hand for traditional animation or via the use of image editing software such as Photoshop, PaintMan, PaintToolSai, ClipStudio, and Krita. Therefore, one can see automatic colorization pipelines as a way to improve the artist's workflow, and such a system has recently been implemented into ClipStudio.

Automatic user-guided colorization is a challenging task for computer vision as black and white line arts does not provide enough semantic information. As colorization represents an essential part of the artist process, and directly influences the final art piece, automatic approaches require to produce aesthetically pleasing and consistent results while conserving enough texture and shading material.

Previous contributions from the deep learning community have explored image colorization [4, 6, 7, 11, 20, 22, 25, 30, 31]. While first works focused on user-guided gray scale images colorization [7, 11, 31] and could not handle sparse inputs, others explored color strokes colorization [6, 20, 25]. Nevertheless, none of these methods yield suitable generalization on unseen images nor generate pleasing enough images. More recent works addressed these issues and enabled for the first time the use of such pipelines in production environments, providing qualitative results [4, 22]. The current state of the art [4] introduced a new model and its training pipeline. The authors also stated that, based on a previous paper, randomly sampled pixels to generate color hints for training is enough to enable user strokes input for inference. One issue is that this statement is based on gray scale images colorization [31], a task close yet far enough from the one of line art colorization.

Our contributions include:

- The use of stroke simulation as a substitute for random pixel sampling to provide color hints during training.
- The introduction of a cleaner and more visually pleasing dataset containing high-quality anime illustration filtered by hand.
- The exploration of a double generator GAN for this task previously studied by contributions for multi-domain training.

The name "PaintsTorch" has been chosen to refer to this work. "Paints" stands for painting and "Torch" for the Pytorch deep learning framework. The name analogous by the "PaintsChainer" tool [22], where "Chainer" refers to the Chainer deep learning library.

2 RELATED WORK

As previous works in the literature has exhaustively described none deep learning based approaches, these are not explained in this paper. Nowadays, deep learning approaches to the line art colorization problem have shown to be the trend and outperform previous methods.

2.1 Synthetic Colorization

Previous works have studied gray scale mapping [7, 11, 31]. It usually consists of trying to map gray scale images to colored ones using a Convolutional Neural Network (CNN) or a generative model such as GAN [8]. By using high-level and low-level semantic information, such models generate photo-realistic and visually pleasing outputs. Moreover, previous works also explored direct mapping between human sketches and realistic images while providing a way to generate multiple outputs out of one sketch.

However, as explained before, semantic information for black and white line art colorization is not conceivable, and these models do not explore all the entire output space.

2.2 Generative Adversarial Network

GAN models [8] are responsible for successful contributions in computer vision generation tasks such as super-resolution, high-definition image synthesis, image inpainting, and image denoising [14, 18, 19]. This architecture has often been described as one of the most beautiful ideas of the deep learning field. It consists of training two networks against each other, one being a discriminative model and the other a generative one. Hopefully, at some point, the discriminator is fooled by the generator, and we consider the model trained.

While being able to generate good quality results, the vanilla implementation of GAN [8] suffers from mode collapse, vanishing gradient issues, and others. Improvement of the former model has been discussed in the literature introducing a gradient penalty [9] and a new loss based on the Wasserstein-1 distance [2]. When conditioned on external information such as class labels, the model referred to cGAN [21] can generate higher quality output as well as enabling natural controls on the generator.

The current state of the art for user-guided line art colorization [4] used such a model referred to cWGAN-GP to obtain their results. As well as introducing a deeper model compared to previous work, they introduce the use of a local feature network described in Section 3.3, thus providing semantic like information to the generator and the discriminator models. Furthermore, Their method manage to train a GAN with training data, illustrations, different from the inference one, line arts.

2.3 Double Generator GAN

The task of cross-domain training has been studied by previous works such as StarGAN [3] and DualGAN [27]. StarGAN translates an image to another domain using one generator inspired by the classical image-to-image GAN .[13] As their work handle discrete labels as target domains, our work considers hint matrices and features vector from a local feature network as continuous labels. This capacity is essential to the artistic field.

DualGAN goes a step further and introduces a double Generator. Their first Generator is used for translation and their second one for reconstruction. The two generators share only a few parameters. As this contribution allows better visually perceptive results, we consider the approach interesting enough to be explored for the task of line art colorization.

Table 1: The Table describes the dataset composition. The Paper Line Arts and Paper Colored lines refer to the dataset of the current state of the art while the Ours Colored refers to our image dataset adding up to the Total.

Source	Images
Paper Line Arts	2 780
Paper Colored	21 931
Ours Colored	**21 930**
Total Colored	43 861
Total	43 641

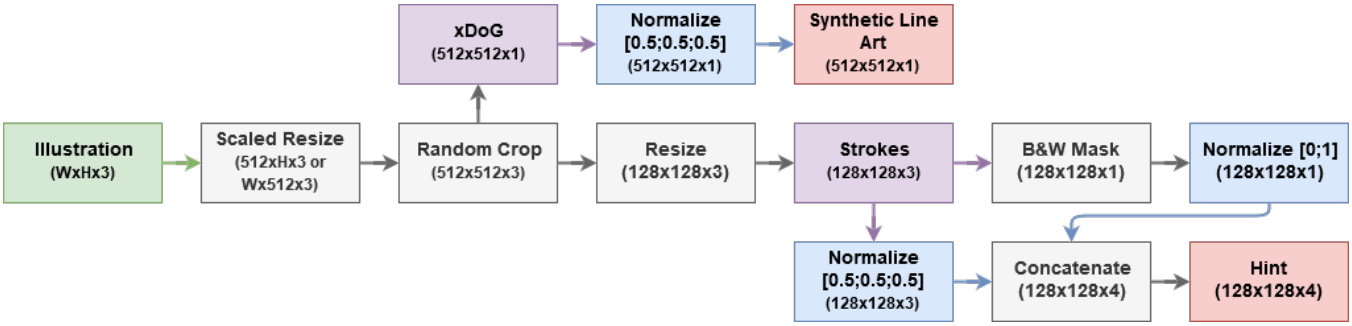

Figure 2: The illustration describes the entire transformation pipeline of the model's inputs. The pipeline outputs a lineart and corresponding color hint image from an input illustration. The process can be applied to any given illustration dataset.

3 PROPOSED METHOD

All the models presented in the paper are trained following the next instructions. In Section 3.1 we describe the dataset used for training and evaluation, Section 3.2 the prepossessing pipeline, Section 3.3 the model architecture, Section 3.3 the loss used to train the models, and Section 3.5 the training process.

3.1 Dataset

As stated by the current state of the art, datasets for colorization are available on the web. However, if we consider two of the most known and large datasets, Nico-opendata [7, 12, 16] and Danbooru2017 or Danbooru2018 [1], they both contain messy illustrations, and line arts are mixed with colored illustrations. In this sense, Ci et al. [4] gathered their custom dataset containing 21 930 colored illustrations for training and 2 780 high-quality line arts for evaluation.

Nevertheless, after investigation, we found images that cannot be qualified as illustrations, and the quality of the paintings is not consistent over the entire set of colored images. To this end, we collected a custom training dataset composed of 21 930 consistent, high-quality anime like illustrations. These illustrations have been filtered by hand to ensure some subjective quality standards. On the other hand, the line art set used for evaluation is not subject to these critics. The exact composition of the dataset can be found in Table 1.

3.2 Preprocessing

3.2.1 Synthetic Line Art. Illustrations do not often come with their corresponding high-quality line arts. To counter this issue, previous works use synthetics line arts to train their model. To generate high-quality fake line arts out of colored illustrations, Extended Difference of Gaussians (xDoG) [28] has proven to be one of the best methods. xDoG produces realistic enough sketches as it can be observed in Figure 3. In this work, we use the same set of parameters as the previous state of the art: $\gamma = 0.95, \phi = 1e^{9}, k = 4.5, \epsilon = -1e^{-1}$, $\sigma \in \{0.3; 0.4; 0.5\}$.

3.2.2 Simulated Hint Strokes. As mentioned, the current state of the art [4] stated that randomly sampled pixels hint during training is enough to enable natural interaction using user strokes as input

Figure 3: xDoG fake line art on the left generated out of the illustration on the right with parameters described in Section 3.2.1 and $\sigma = 0.4$

for inference. Their assumption is based on Zhang et al. contribution [31] which deals with gray scale image colorization. As their problem is not entirely the same, we explored the use of simulated strokes as a substitute for hint generation.

We simulated human strokes using the PIL drawing library with a round brush. To this end, we define four parameters: the number of strokes $n_{strokes} \in [0; 120]$, the brush thickness $t_{brush} \in [1; 4]$, the number of waypoints per stroke $n_{points} \in [1; 5]$ and a square of width $w_{range} \in [0; 10]$ which defines the range of movement of one brush stroke. By doing so, we aim to make the model robust to messy user inputs. An example of a brush stroke simulation generated from an illustration can be found in Figure 5.

3.2.3 Model Inputs Transformations. To be handled by the deep learning model, all inputs are preprocessed and follow certain transformations. First, the illustration input is randomly flipped to the left or the right. Then, the image is scaled to match 512 pixels on its smaller side before being randomly cropped to a 512 by 512 pixels image. The obtained resized and cropped illustration is then used to generate the synthetic gray scale line art (512x512x1). The same transformed illustration is then resized to a 128 by 128 pixels and used to generate a stroke simulated hint and its corresponding black and white mask to finally obtain the hint image used for training (128x128x4). All pixel data except the one coming from

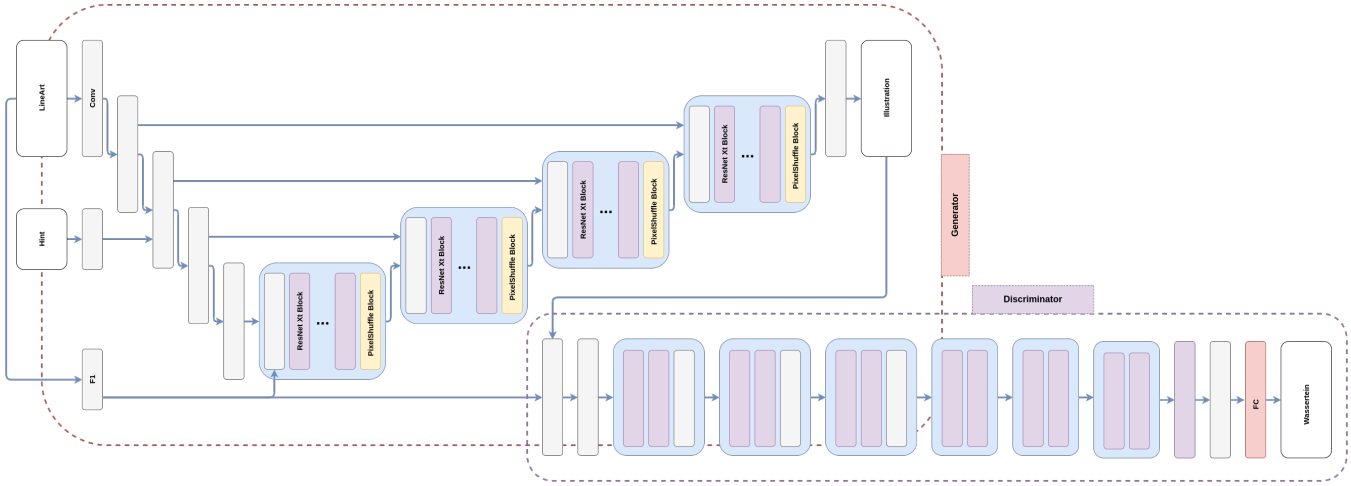

Figure 4: The Figure shows the Generator and Discriminator Convolutional Neural Network architectures. Both are composed of ResNet Xt Blocks (violet) and Pixel Shuffle Blocks (orange). The Generator uses residual connections in the form of a UNet.

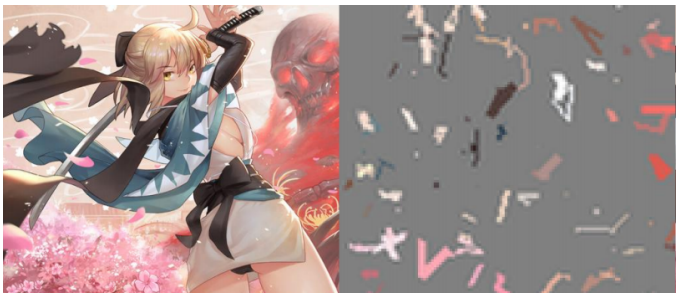

Figure 5: Example of a simulated brush stroke hint on the right generated from the left illustration left. The strokes are supposed to represent a normal usage of the software. Their density and thickness varies randomly to make the model more robust int its usage.

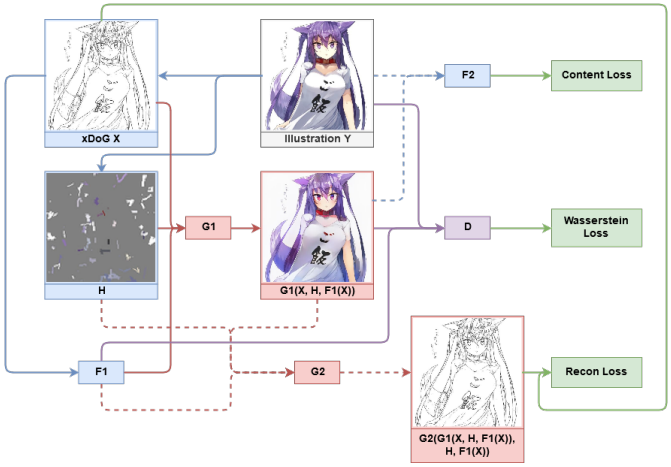

Figure 6: The illustration describes the overall model architecture from a higher perspective. Arrows describe the path of the data through the models to the losses. The colors allow distinguishing between each path and each piece of the architecture. Green arrows refer to connections with the loss modules, plain blue ones for the input modules, red ones for the generators, and violet for the discriminator.

the black and white mask is normalized with an [0.5,0.5,0.5] std and mean, and the mask is normalized to map the [0;1] range. The entire transformation pipeline can be observed in Figure 2.

3.3 Model

Regarding the model architecture, the GAN we used is similar to the one used by Ci et al. [4]. This model is shown in the Figure 4 but a more detailed explanation of the model can be found in their paper. The Generator $\mathcal{G}_1$ is a deep U-Net model [23] composed of ResNetXt [10] blocks with dilated convolutions to increase the receptive field of the network. LeakyReLU [29] is used as activation with a 0.2 slope except for the last layer using a Tanh activation. The Discriminator is inspired by the SRGAN one but upgraded with the same kind of blocks as the generator without any dilation and using more layers. They also introduced the use of a Local Feature Network $\mathcal{F}_1$. This network is an Illustration2Vec [24] model able to tag illustrations. These tags are passed through the Generator and the Discriminator along with the hint image as a way to condition and give semantic information to the GAN.

Our contribution introduces the use of a second Generator $\mathcal{G}_2$ using the same architecture as $\mathcal{G}_1$. This second Generator is responsible for the generation of a synthetic line art out of the fake illustration inferred by the first one. This kind of approach has been used for cross-domain training [27]. By doing so, we aim for improving the overall perceptive quality of the generated illustration as well as giving $\mathcal{G}_1$ further insight and better training objective. A schematic of the whole architecture can be found in Figure 6.

3.4 Loss

As indicated in Ci et al. paper [4], the loss functions are a combination of all GAN's improvements described earlier in the paper. We want the second Generator to back propagate its signal to the first one, so we add a new term to the Generator loss, which relies on a simple MSE. In this Section, we describe each loss used to train the model.

First, we define the global Generator loss as a combination of three components: a content component, an adversarial one, and a reconstruction one.

$$\mathcal{L}_{\mathcal{G}} = \underbrace{\underbrace{\mathcal{L}_{cont}}_{\text{content loss}} + \underbrace{\lambda_1 . \mathcal{L}_{adv}}_{\text{adv loss}} + \underbrace{\mathcal{L}_{recon}}_{\text{recon loss}}}_{\text{total loss}} \tag{1}$$

The adversarial part is computed thanks to the local feature network $\mathcal{F}_1$ used as conditional input, and WGAN-GP [9] used to distinguish fake examples from real ones. The component is weighted by parameter $\lambda_1 = 1e^{-4}$

$$\mathcal{L}_{adv} = -\mathbb{E}_{\mathcal{G}_1(X,H,\mathcal{F}_1(X))\sim\mathbb{P}_g}[\mathcal{D}(\mathcal{G}_1(X,H,\mathcal{F}_1(X)),\mathcal{F}_1(X))] \tag{2}$$

A perceptual loss is used for the content loss, which relies on an L2 difference between generated output and target CNN feature map coming from the fourth convolution activation of a pretrained VGG16 [26] on ImageNet [5].

$$\mathcal{L}_{cont} = \frac{1}{chw} \|\mathcal{F}_2(\mathcal{G}_1(X,H,\mathcal{F}_1(X))),\mathcal{F}_2(X)\|_2^2 \tag{3}$$

The loss signal we call reconstruction loss describes the ability of generator $\mathcal{G}_2$ to produce a fake line art out of the fake illustration generated by $\mathcal{G}_1$ as close from the xDoG synthetic line art used for training. As the output does not contain multi-channel information, the difference is computed with a mean squared error.

$$\mathcal{L}_{recon} = MSE\ [\mathcal{G}_2(\mathcal{G}_1(X,H,\mathcal{F}_1(X)),H,\mathcal{F}_1(X)),\ X] \tag{4}$$

Concerning the Discriminator loss, it is a combination of the Wasserstein loss and the penalty loss.

$$\mathcal{L}_{\mathcal{D}} = \underbrace{\underbrace{\mathcal{L}_w}_{\text{critic loss}} + \underbrace{\mathcal{L}_p}_{\text{penalty loss}}}_{\text{total loss}} \tag{5}$$

The critic loss is described in the WGAN paper [2].

$$\mathcal{L}_w = \mathbb{E}_{\mathcal{G}_1(X,H,\mathcal{F}_1)\sim\mathbb{P}_g}[\mathcal{D}(\mathcal{G}_1(X,H,\mathcal{F}_1(X))),\mathcal{F}_1(X)] - \mathbb{E}_{Y\sim\mathbb{P}_r}[\mathcal{D}(Y,\mathcal{F}_1(X)))] \tag{6}$$

The penalty term, as described in the current state of the art [4], is composed of two components, a penalty term and an extra constraint from Karras et al [15] The two parts are weighted by parameters $\lambda_2 = 10$ and $\epsilon_{drift} = 1e^{-3}$.

$$\mathcal{L}_p = \lambda_2 . \mathbb{E}_{\hat{Y}\sim\mathbb{P}_r}[(\left\|\nabla_{\hat{Y}}\mathcal{D}(\hat{Y},\mathcal{F}_1(X)),\mathcal{F}_1(X)]\right\|^2 - 1)^2] + \epsilon_{drift} . \mathbb{E}_{\hat{Y}\sim\mathbb{P}_r}[\mathcal{D}(Y,\mathcal{F}_1(X))^2] \tag{7}$$

Table 2: The Table compares the Frechet Inception Distance (FID) of multiple models trained over 100 epochs. [Paper] refers to the colored images used for training by Ci et al. [4], [Custom] to the images dataset we collected, and [Custom + Paper] to the combination of both. Lower value is better. STD stands for standard deviation to the mean.

Model and Options	FID	STD
(Paper) Random, Simple	104.07	0.016
(Paper) Strokes, Simple	**68.28**	**0.048**
(Paper) Strokes, Double	83.25	0.019
(Custom) Random, Simple	82.23	0.022
(Custom) Strokes, Simple	**64.81**	**0.035**
(Custom) Strokes, Double	**65.15**	**0.006**
(Custom + Paper) Strokes, Double	75.71	0.032

Table 3: The Table compares the FID of our model trained over 100 epochs for different batch sizes: 4, 16, and 32. A higher batch size returns lower FID values. Lower value is better. STD stands for standard deviation to the mean.

Batch Sizes	FID	STD
4	74.53	0.003
16	**64.35**	**0.061**
32	**65.15**	**0.006**

3.5 Training

The models are trained on an Nvidia DGX station using four V100 GPUs with 32Go of dedicated RAM each. The ADAM optimizer [17] is used with parameters: learning rate $\alpha = 1e^{-4}$ and betas $\beta_1 = 0.5$, $\beta_2 = 0.9$. The same training pipeline as previous work has been applied. One gradient descent step is first applied to train the Discriminator $\mathcal{D}$, then to train Generator $\mathcal{G}_1$ and finally $\mathcal{G}_2$. For comparison, all models have been trained for 100 epochs. However, the final one is trained on 300 epochs.

4 RESULTS

In our contribution, we trained and experimented different model pipelines. To evaluate and compare these models, we realized multiple evaluations. In this Section, we describe our results.

4.1 FID Evaluation

Peak Signal-to-Noise Ratio (PSNR), as stated by Ci et al. [4], does not assess joint statistics between targets and results. Moreover, our dataset does not provide colored illustrations with their corresponding line arts. In that sense, measuring similarities between the two data distributions, synthetic colorized line arts, and authentic line arts is more appropriate to evaluate the model's performances. The FID can measure intra-class dropping, diversity, and quality. A small FID means that two distributions are similar. The FID evaluation is performed the same way as Ci et al. [4] between the colored illustrations train set and the line arts test set. It extracts features

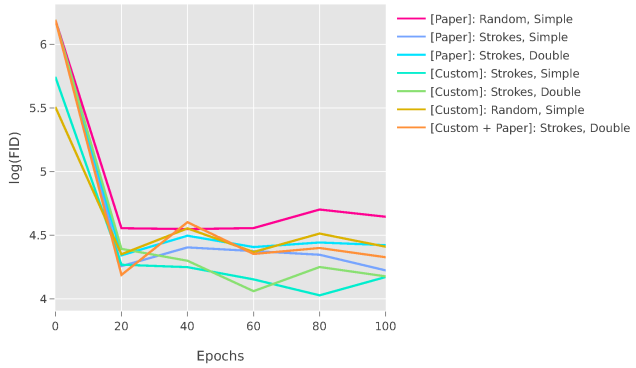

Figure 7: The graph compares the FID on a logarithmic scale of every model trained over 100 epochs.

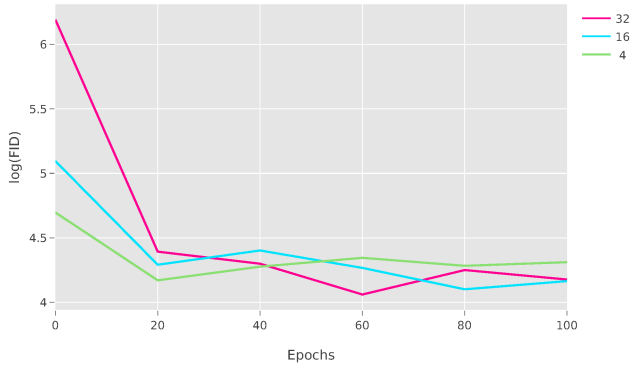

Figure 8: The graph compares the FID on a logarithmic scale for different batch size, 32, 16 and 4 over 100 epochs.

from intermediate layers of a pre-trained Inception Network to model the data distribution using a Multivariate Gaussian Process.

Results of the FID evaluations can be found in Tables 2, 3 and Figures 7, 8. This objective evaluation allows us to infer some insight about the different model training pipelines we tried during our experimentation. Stroke simulation, instead of randomly sampled pixels for hint generation provides the most notable positive impact on the FID value. The overall quality improvement of the training illustrations also yields better results. Though, we cannot deduce if the double generator visually improves the output with this kind of evaluation as art is a completely subjective matter. Finally, greater batch size does not have that much of an impact on the FID value, but allows faster training.

Table 4: The Table describes the Mean Opinion Score (MOS) for every model we compared in our study. PaperRS stands for current state of the art illustration data with randomly sampled pixels and one generator, CustomSS for our illustration data with stroke simulation with one generator and CustomSD for double generator. STD stands for Standard Deviation to the mean

Model	MOS	STD
PaperRS	1.60	0.85
CustomSS	2.95	0.92
CustomSD	**3.10**	**1.02**

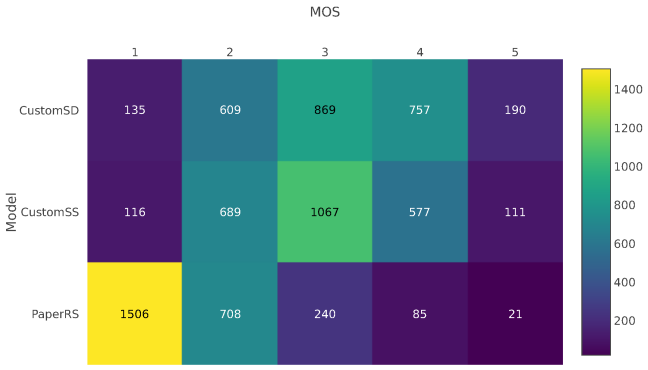

Figure 9: The heatmap compares the MOS ratings for every model we study. PaperRS stands for current state of the art illustration data with randomly sampled pixels and one generator, CustomSS for our illustration data with stroke simulation with one generator and CustomSD for double generator.

4.2 MOS Evaluation

It is generally challenging to evaluate art results as the artistic perception is different for every person, and the FID is not good at assessing the model's quality in this context. Thus, as previous works did, we also conducted a MOS evaluation of the different model's pipelines. This evaluation aims at quantifying the reconstruction of perceptually convincing line art colorization. To do so, we asked 16 users to score synthetic illustrations between one and five corresponding to bad quality and excellent quality. The evaluation is composed of 160 randomly selected line art from the validation set. Corresponding hint images have been created by hand by non-professional users and used to generate 160 corresponding illustrations for each of the three models we compared. Thus, the overall number of images to rate per user is 480. Examples of the illustrations shown to the users are available in Figure 10.

Our results show in Table 4 and Figure 9 that our models are perceptively better when compared to the previous state of the

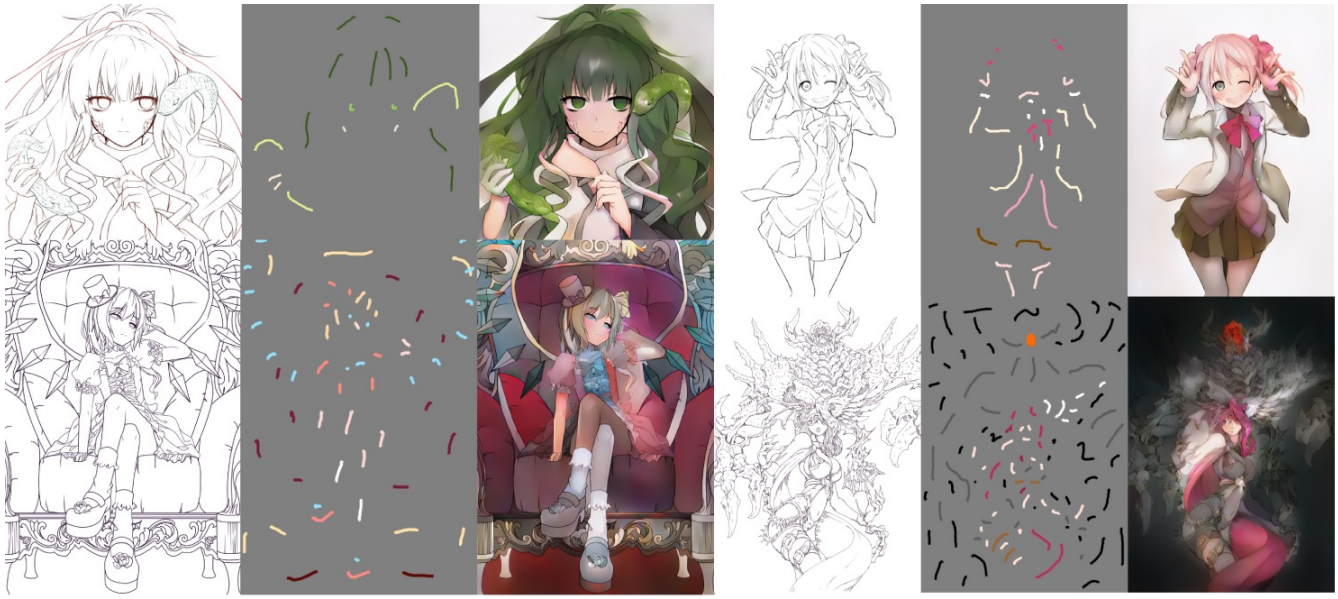

Figure 10: The illustrations shows examples generated from our contribution model using the line art on the left and the hint in the middle.

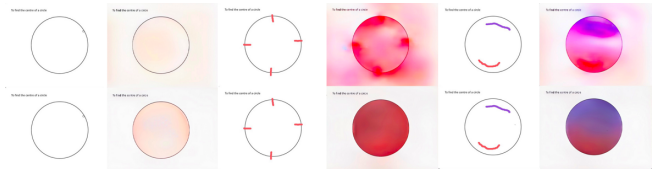

Figure 11: The illustrations compare the previous state of the art (top) to our results (bottom) in coloring a 3 simple tasks: filling a circle without hint, filling a circle with on the edge brush stokes, and performing gradients using messy inputs.

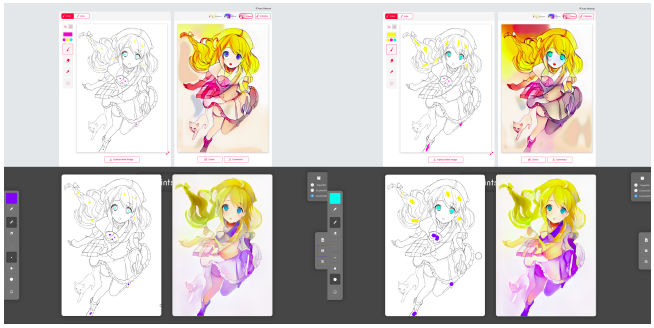

Figure 12: The illustrations shows the differences between PaintsChainer and our model when the brush strokes are thin versus when the hint is made out of thicker strokes.

art. We realized a unilateral Student test with a significance level $\alpha = 0.001$ to compare the MOS mean of our model to the current state of the art with a sample size $n = 16$. We obtained a t value of 4.525, approximately equivalent to a $p - value$ inferior to 0.001. Statistically, our study validates that our contributions improve the model described by Ci et al. [4]. This evaluation score also allows us to conclude on the use of a double generator. The dual generator seems to slightly improve the illustration quality and provides higher contrast with misplaced colors.

4.3 Visual Improvements

The differences in our approach results in visibly perceptive improvements. As it can be observed in Figure 11, training the models using simulated strokes improves the general ability to fill the inner part of forms as well as allowing the user's inputs to be messier. When the user's strokes exceed the outer part of the area slightly, the current state of the art fails at capturing the user's will only to fill the inner part. As shown in Figure 11, color gradients are also visually more pleasant using our contribution.

5 APPLICATION

In this Section, we discuss the possible use of this kind of application and the web app we developed to ease the experiments.

In order to use the models, we developed a web application to allow real-time user interactions with our contribution with simple tools such as a brush pen, an eyedropper, and an eraser with various brush sizes. Visual of the app can be found in Figure 13. It has been created using a dockerized flask rest API to serve the PyTorch models on the DGX station we used for training. The web application performs API calls each time the user's touch to the canvas end. To ease the production of the Figures for this contribution, and allow

Figure 13: The Figure is a screen-shot of the web app environment created to use our model. On the left, there is a tool bar with a color selector, an eyedropper, a pen, an eraser, and sizes for the brush. On the right the tool bar allows to import a line art, save the colored illustration and save the hint canvas. The top right tool box is used to select a model to use. These models are the one described in the study. The left canvas is the one the user can draw on whereas the right one displays the illustration results.

Figure 14: The illustration shows the appearance of artifacts on the output image of our model in the presence of highly saturated colors or densely populated hint images.

users to share their creations, we also provide some tools to save the illustration and the hint.

As shown in Figure 15, this kind of tool can be included in an artist's workflow saving time and providing new types of creativity. While the contribution can be useful to professionals illustrators, it can also leverage the power of digital colorization to the beginners through natural interaction. As it can be used for finalized art pieces, it is also a way to prototype quickly. The field of animation could also benefit from this kind of application if the model can allow temporally stable outputs.

6 LIMITATIONS

PaintsTorch enables natural interaction when painting illustrations. Even though our solution provides visually pleasing colored illustrations, it sometimes fails to some extent.

When too many strokes are included in the hint image, or when colors are highly saturated, the network tends to produce artifacts, as Figure 14 shows. These artifacts can be the result of the dataset color distribution. It could be resolved by introducing data augmentation on the source and hint images such as changing the hue, saturation, and contrast, but also by allowing more strokes per hint map in the stoke simulation.

Moreover, in some cases, our network does not always apply the exact same colors as the given ones. As it can be observed in Figure 15, it failed to capture the artist's intent to make the eyes pinkish.

Finally, our pipeline does not use any layer system like painting software such as Photoshop, Krita, and others do. Digital artists usually work with multiple layers. Some are used for the sketch, others for the lineart, and colors. PaintsTorch only delivers a final illustration, including the lineart. The colors cannot be separated by the artist afterward and force him to paint directly on the produced colored illustration.

7 CONCLUSION

Guided line art colorization is a challenging task for the computer vision domain. Previous works have shown that deep learning yield better results than previous methods. In our contribution, we propose three changes that can improve the current state of the art's results. Our first contribution is the introduction of stroke simulations as a way to replace random pixels activation to generate the hint used for training. Our second contribution is the use of a custom, high resolution, and quality controlled dataset for training illustrations. Our third contribution is the exploration for the use of a second generator, which is in charge of generating synthetic lines art based on the produced artificial illustrations. These three contributions, as the study shows, allow for improved perceptive results compared to previous works.

Our results allow to produce quality illustrations on unseen line arts and the used of different input stroke sizes. However, the model still suffers from small artifacts. Moreover, it does not always seem to use the exact color information provided by the user's hints. The model could also provide increased robustness to thinner or thicker line arts and color strokes by changing few training parameters.

One extension of this work could be studying the impact of using a more massive and diversified dataset. We are also planning to make the model stable temporally to be used for animation purposes.

ACKNOWLEDGEMENTS

We would like to thanks the De Vinci Innovation Center for providing an Nvidia DGX-station, as well as all the users who took the time to participate to our study, and the Pytorch team for providing such a useful and accessible framework for deep learning on multi-GPU machines.

Figure 15: The Figure is a representation of how an artist would naturally embed our contribution in his workflow.

REFERENCES

[1] Gwern Branwen Aaron Gokaslan Anonymous, the Danbooru community. 2019. Danbooru2018: A Large-Scale Crowdsourced and Tagged Anime Illustration Dataset. https://www.gwern.net/Danbooru2018. https://www.gwern.net/Danbooru2018 Accessed: DATE.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein Generative Adversarial Networks. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, International Convention Centre, Sydney, Australia, 214–223. http://proceedings.mlr.press/v70/arjovsky17a.html

[3] Yunjey Choi, Min-Je Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. 2017. StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation. *CoRR* abs/1711.09020 (2017). arXiv:1711.09020 http://arxiv.org/abs/1711.09020

[4] Yuanzheng Ci, Xinzhu Ma, Zhihui Wang, Haojie Li, and Zhongxuan Luo. 2018. User-Guided Deep Anime Line Art Colorization with Conditional Adversarial Networks. *CoRR* abs/1808.03240 (2018). arXiv:1808.03240 http://arxiv.org/abs/1808.03240

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

[6] Kevin Frans. 2017. Outline Colorization through Tandem Adversarial Networks. *CoRR* abs/1704.08834 (2017). arXiv:1704.08834 http://arxiv.org/abs/1704.08834

[7] Chie Furusawa, Kazuyuki Hiroshiba, Keisuke Ogaki, and Yuri Odagiri. 2017. Comicolorization : Semi-automatic Manga Colorization. *CoRR* abs/1706.06759 (2017). arXiv:1706.06759 http://arxiv.org/abs/1706.06759

[8] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'14)*. MIT Press, Cambridge, MA, USA, 2672–2680. http://dl.acm.org/citation.cfm?id=2969033.2969125

[9] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. 2017. Improved Training of Wasserstein GANs. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 5767–5777. http://papers.nips.cc/paper/7159-improved-training-of-wasserstein-gans.pdf

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). arXiv:1512.03385 http://arxiv.org/abs/1512.03385

[11] Paulina Hensman and Kiyoharu Aizawa. 2017. cGAN-based Manga Colorization Using a Single Training Image. *CoRR* abs/1706.06918 (2017). arXiv:1706.06918 http://arxiv.org/abs/1706.06918

[12] Hikaru Ikuta, Keisuke Ogaki, and Yuri Odagiri. 2016. Blending Texture Features from Multiple Reference Images for Style Transfer. In *SIGGRAPH ASIA 2016 Technical Briefs (SA '16)*. ACM, New York, NY, USA, Article 15, 4 pages. https://doi.org/10.1145/3005358.3005388

[13] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. 2016. Image-to-Image Translation with Conditional Adversarial Networks. *CoRR* abs/1611.07004 (2016). arXiv:1611.07004 http://arxiv.org/abs/1611.07004

[14] Justin Johnson, Alexandre Alahi, and Fei-Fei Li. 2016. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. *CoRR* abs/1603.08155 (2016). arXiv:1603.08155 http://arxiv.org/abs/1603.08155

[15] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2017. Progressive Growing of GANs for Improved Quality, Stability, and Variation. *CoRR* abs/1710.10196 (2017). arXiv:1710.10196 http://arxiv.org/abs/1710.10196

[16] Y. Kataoka, T. Matsubara, and K. Uehara. 2017. Automatic manga colorization with color style by generative adversarial nets. In *2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. 495–499. https://doi.org/10.1109/SNPD.2017.8022768

[17] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. http://arxiv.org/abs/1412.6980 cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[18] Orest Kupyn, Volodymyr Budzan, Mykola Mykhailych, Dmytro Mishkin, and Jiri Matas. 2017. DeblurGAN: Blind Motion Deblurring Using Conditional Adversarial Networks. *CoRR* abs/1711.07064 (2017). arXiv:1711.07064 http://arxiv.org/abs/1711.07064

[19] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. 2016. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. *CoRR* abs/1609.04802 (2016). arXiv:1609.04802 http://arxiv.org/abs/1609.04802

[20] Yifan Liu, Zengchang Qin, Zhenbo Luo, and Hua Wang. 2017. Auto-painter: Cartoon Image Generation from Sketch by Using Conditional Generative Adversarial Networks. *CoRR* abs/1705.01908 (2017). arXiv:1705.01908 http://arxiv.org/abs/1705.01908

[21] Mehdi Mirza and Simon Osindero. 2014. Conditional Generative Adversarial Nets. *CoRR* abs/1411.1784 (2014). arXiv:1411.1784 http://arxiv.org/abs/1411.1784

[22] Preferred Networks. 2017. paintschainer. https://paintschainer.preferred.tech/.

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. *CoRR* abs/1505.04597 (2015). arXiv:1505.04597 http://arxiv.org/abs/1505.04597

[24] Masaki Saito and Yusuke Matsui. 2015. Illustration2Vec: A Semantic Vector Representation of Illustrations. In *SIGGRAPH Asia 2015 Technical Briefs (SA '15)*. ACM, New York, NY, USA, Article 5, 4 pages. https://doi.org/10.1145/2820903.2820907

[25] Patsorn Sangkloy, Jingwan Lu, Chen Fang, Fisher Yu, and James Hays. 2016. Scribbler: Controlling Deep Image Synthesis with Sketch and Color. *CoRR* abs/1612.00835 (2016). arXiv:1612.00835 http://arxiv.org/abs/1612.00835

[26] K. Simonyan and A. Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014).

[27] Hao Tang, Dan Xu, Wei Wang, Yan Yan, and Nicu Sebe. 2019. Dual Generator Generative Adversarial Networks for Multi-Domain Image-to-Image Translation. *CoRR* abs/1901.04604 (2019). arXiv:1901.04604 http://arxiv.org/abs/1901.04604

[28] Holger Winnemöller, Jan Eric Kyprianidis, and Sven C. Olsen. 2012. XDoG: An eXtended difference-of-Gaussians compendium including advanced image stylization. *Computers & Graphics* 36, 6 (2012), 740 – 753. https://doi.org/10.1016/j.cag.2012.03.004 2011 Joint Symposium on Computational Aesthetics (CAe), Non-Photorealistic Animation and Rendering (NPAR), and Sketch-Based Interfaces and Modeling (SBIM).

[29] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. 2015. Empirical Evaluation of Rectified Activations in Convolutional Network. *CoRR* abs/1505.00853 (2015). arXiv:1505.00853 http://arxiv.org/abs/1505.00853

[30] Lvmin Zhang, Yi Ji, and Xin Lin. 2017. Style Transfer for Anime Sketches with Enhanced Residual U-net and Auxiliary Classifier GAN. *CoRR* abs/1706.03319 (2017). arXiv:1706.03319 http://arxiv.org/abs/1706.03319

[31] Richard Zhang, Jun-Yan Zhu, Phillip Isola, Xinyang Geng, Angela S. Lin, Tianhe Yu, and Alexei A. Efros. 2017. Real-Time User-Guided Image Colorization with Learned Deep Priors. *CoRR* abs/1705.02999 (2017). arXiv:1705.02999 http://arxiv.org/abs/1705.02999