Transformation of Real to Cartoon Images

Rumana Akter *180104106*
Ashif Reza *180104107*
Asifur Rahim *180104109*
Ayesha Afroze *180104118*

***Abstract*—Cartoons are a type of illustration, usually drawn in a surreal or semi-realistic style and sometimes animated. They improve the quality of entertainment by making it more enjoyable and interesting. In this paper, we propose a model which transforms a real-world image into a cartoon-like image. For this purpose, we have used a generative adversarial network framework. Our model is trained using unpaired images of the real world and cartoons.**

I. Introduction

Entertainment is an essential part of our lives and cartoon occupies a large part of it. It is the most popular form of entertainment. Not only that is also an art form for expression from publication in printed media to storytelling. It takes a lot of work and requires significant artistic talent to painstakingly recreate real-world scenes in cartoon forms. A significant amount of time can be saved for artists by using specially created processes that can automatically convert real-world photos into high-quality cartoon-style graphics, allowing them to concentrate on more creative work. The currently available picture editing software and algorithms do not produce satisfactory results for cartoonization of real-world images. We propose our model in this paper which generates cartoon images from real-world images. This work will play a significant role in the entertainment industry. Our model contains a Generative Adversarial Network which is trained using unpaired dataset of real-world images and cartoon images. We have used the content loss as proposed in [9] and adversarial loss for maintaining sharp edges in cartoons.

II. Related Works

In the absence of paired examples, Jun-Yan Zhu *et al.* [1] provided a technique for understanding how to convert images from a source domain X to a target domain Y. The objective was to learn a mapping G: X–Y using an adversarial loss such that the distribution of pictures from G(X) is identical to the distribution Y. The tasks include assigning semantic descriptions to images from the Cityscapes dataset. A custom generative network was used which was adopted from Johnson *et al.* [2]. PatchGANs are employed for the discriminator networks in order to categorize whether overlapping image patches are real or fake. FCN score was used as a metric for evaluation.

Xinrui Wang *et al.* [6] proposed to separately identify three white-box representations from images by observing the cartoon painting behaviour and consulting artists: the surface representation, the structure representation and texture representation. The retrieved representations are learned and the cartoonized images are generated using a Generative Adversarial Network (GAN) architecture. They gathered 5000 photographs of the landscape and 10,000 images for the human face from the FFHQ datasets. Frechet Inception Distance (FID) has been used as evaluation metric.

CartoonGAN is a unique GAN-based technique to cartoon pictures that Yang Chen *et al.*[3] offer. Unpaired picture sets have been applied in the past. It utilizes a GAN architecture with two CNNs. A generator and a discriminator, respectively. The generator is trained to create output that deceives the discriminator, which categorizes whether the picture is synthetic or taken from the actual target manifold. Its technique may replicate the styles of specific painters when cartoon pictures from those artists are utilized in training. Real-world pictures and cartoon images are both included in the training data, however only real-world photos are included in the test data. Flickr is used to download real-world images. For training, four different cartoon picture types—Makoto Shinkai and Mamoru Hosoda, Miyazaki Hayao, and ”Paprika” style models—are employed. Images from a number of quick cartoon flicks are utilized to teach the Makoto Shinkai and Mamoru Hosoda style models. Images from the animated movies ”Spirited Away” and ”Paprika” are used to teach the Miyazaki Hayao and ”Paprika” style models.

Yang Chen's *et al.*[4] proposal for a network specifically for comics was rejected. The VGG16 network, a CNN with exceptional performance in classification tasks, was used to train the model. For content feature extraction, the same network architecture that was trained by [13] for object classification is employed. To train a comic-style network, images of real objects and real-world settings were combined with 10 different artists' comics. In the validation data and the test data, the network's classification accuracy is 99.25 and 99.5 percent, respectively.

Using the CycleGAN model, K. M. Arefeen Sultan *et al.* [5] suggested a method for converting cartoon pictures into photo-realistic ones. Spectral Normalization has been introduced to the model to increase efficiency and stability. The lowest

Frechet Inception has been attained by their suggested model.

III. Dataset Description

Two datasets comprising both real-world and cartoon photos were utilized. An unpaired dataset has been used by us. Safebooru dataset [10] from Kaggle has been used. There are 10 lac cartoon images in it. 1000 photos were taken from them to train our model. We used the MS COCO dataset (Microsoft Common Objects in Context) [11], a large picture dataset with 328,000 photos of commonplace items and people, for real-world images. We have used 1000 images from it. The model is trained using an edge-smoothed version of each cartoon image.

IV. Formulation

The approaches that we have taken to convert a real-world photo to a cartoon-like photo are discussed in this section.

A. Loss Function

Loss functions indicate the variation between the distribution of data produced by the GAN and the distribution of the real data. The loss function that we used from [4] to train the discriminator and the generator is

$$L(G, D) = L_{(adv)}(G, D) + \omega L_{con}(G, D)(1)$$

This loss function contains two parts. Adversarial loss, $L_{(adv)}(G, D)$ and content loss, $L_{con}(G, D)$. These two loss functions are added together. We have used a weight ω equal to 10 to balance out the losses. The loss is computed from the discriminator output and not from the generator output. The loss from the discriminator output is backpropagated through the discriminator model and generator model to the photo image input data.

1) Adversarial Loss : The adversarial loss causes the generator to transform a realistic photo into a cartoon-like photo. Its value denotes whether the generated image appears like a cartoon image or not. The adversarial loss is used in both the generator and discriminator. Generative Adversarial Networks were introduced in [9], which mention that the following function is one that the generator seeks to minimize and the discriminator tries to maximize.

$$\mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))](2)$$

While the depiction of neat edges is a crucial element of cartoon images, their percentage in the overall composition is often deficient. A discriminator gets confused when the output image has correct shading but unclear edges. To solve this problem, we have trained with cartoon images along with their smooth-edged versions so that the discriminator can differentiate between clear and vague edges. We produce smooth-edged cartoons from normal cartoons by detecting the edges, dilating them and then applying Gaussian smoothing on them. Instead of utilizing the aforementioned equation (2), we used the following equation from[4]:

$$L_{adv}(G, D) = \mathbb{E}_{ciS_{data}(c)}[logD(c_i)]$$

$$+\mathbb{E}_{ejS_{data}(e)}[\log(1 - D(e_j))]$$

$$+\mathbb{E}_{pkS_{data}(p)}[\log(1 - D(G(pk)))](3)$$

This is the loss function formula for the discriminator because its output has no bearing on the content loss component of the loss function. For the initialization phase of the generator, this part of the formula is not used as described in [4]. The generator can affect only

$$\mathbb{E}_{pkS_{data}(p)}[\log(1 - D(G(pk)))](4)$$

part of the loss function. Hence, only this part is used in the generator training phase.

2) Content Loss : The content loss function makes sure that both the produced picture and the content image have identical activations of the upper layers. During transformation, the content loss Lcon(G,D) preserves the semantic content. We have used VGG network to calculate the content loss.The layer l output of the created picture is subtracted from the layer l output of the original photograph. The L1 spare regularization :

$$L_{con}(G, D) = \mathbb{E}_{piS_{data}(p)}[||VGG_1(G(pi)) - VGG_1(pi)||_1](5)$$

This formula from [4] is used in the loss function for the generator. Cartoon images differ greatly from photographs in many ways. These variations frequently focus on local areas where representation and regional traits alter drastically. The traditional L2 norm cannot handle such modifications as well as L1 sparse regularization can. To calculate our semantic content loss, the layer "conv4" uses feature maps.

B. GAN

A promising approach is to use Generative Adversarial Networks (GANs) to synthesise the image. Impressive outcomes in picture production and translation have been demonstrated. The Generator and Discriminator are two important parts of GANs. The generator plays the part of a robber by creating phoney samples based on genuine samples and duping the discriminator into thinking the fake samples are real. On the other hand, a Discriminator is similar to a police officer whose job it is to spot irregularities in the samples produced by the Generator and determine whether they are fake or real. The two components continue to compete until perfection is reached, at which point the Generator triumphs and makes the Discriminator fall for bogus data.

1) Discriminator: It indicates data is predicted to be bogus or authentic using a straightforward classifier. It receives training from actual data and offers feedback to a generator. The discriminator receives training to improve its capacity to discriminate.

2) Generator: The generator, on the other hand, is taught to produce higher-quality samples that are identical to genuine ones. Using the original (actual) data as a foundation it produces phoney data. Its goal is to trick the discriminator into thinking it cannot predict a fake image by creating the fake image depending on feedback. And when the generator succeeds in tricking the discriminator, training comes to an end, and we can declare that a generalized GAN model has been developed. It uses neural networks which include hidden layers, activation, and loss functions.

3) Training and prediction of generative adversarial networks: :

1. Train discriminator on real dataset
Discriminator is currently being trained on a real dataset. In the training of the discriminator across n epochs, there is only a forward path; backpropagation is not present. Additionally, the given data is noise-free and exclusively contains real photographs; when dealing with fraudulent images, the discriminator employs instances generated by the generator as negative output. What occurs now during discriminator training. It categorizes data that is authentic and fraudulent. When a discriminator incorrectly labels anything as real or phoney, it suffers from the discriminator loss, which helps it perform better. Through discriminator loss, the discriminator's weights are updated.

2. Train generator
The generator samples random noise while it is being trained and creates an output from that noise. The discriminator is idle when Generator is being trained, and the reverse is true when Generator is being trained. It aims to turn any input random noise into useful data while the generator is being trained. The generator runs over several epochs and requires a lot of time to produce significant data. The following are the steps for training a generator. To assess whether the generator output is genuine or not, a discriminator is first used to construct a generator output on a noise sample. The discriminator loss is calculated. To calculate gradients, backpropagate through a discriminator and a generator. Gradients will be used to update the generator weights.

3. Train discriminator on fake data:
The samples produced by the generator are sent to the discriminator, which determines if the data is true or fake and returns feedback to the generator.

4. Train generator with the output of discriminator:
During the second time, the generator will be trained using the criticism provided by the discriminator in an effort to perform better.
GAN is utilized in our work due to its picture creation and translation capabilities. Real-world photos and cartoon pictures are required for training the GAN. The model is trained using an edge-smoothed version of every cartoon and real-world image in order to help the GAN learn to create clean edges in cartoon images more effectively. Real-world photos are supplied as input to the discriminator while cartoon images are given as input to the generator. Cartoon and generated cartoons are consecutively supplied to the pretrained content network VGG-16 after being transferred from the generator. The following step involves extracting three patches from a produced picture and a real sample, respectively, and feeding them to the local discriminator. Three areas of the resulting image are made credible and photorealistic by the discriminator. Additionally, the global discriminator urges the generator to convert the cartoon picture into the final image produced.

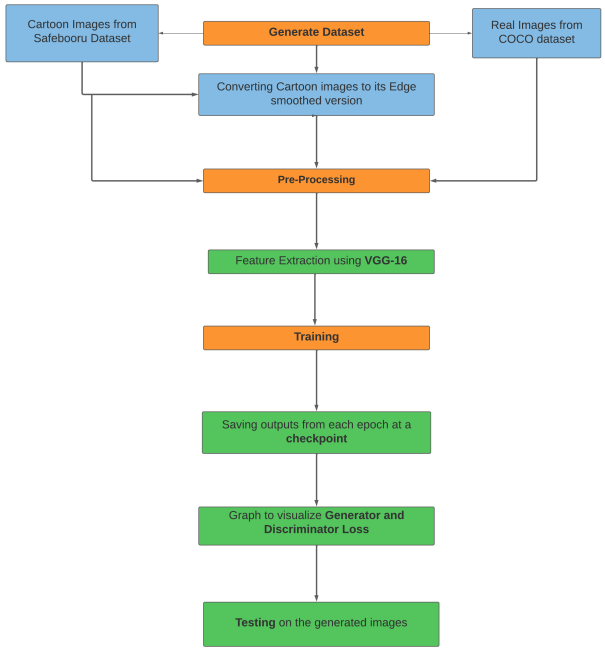

Fig. 1. Workflow Diagram

V. Implementation

The application of our strategy is covered in this part, along with examples of the outcomes.

A. WorkFlow

Fig. 1 depicts our workflow for our work.

1) Generate Dataset: Safebooru dataset [10] is the one that was utilized to create the cartoon images. The model is trained using an edge-smoothed version on each cartoon picture in order to help the GAN learn to create sharp edges in the cartoon image. In our implementation, canny edges detection, the dilation and the gaussian blur with openCV have

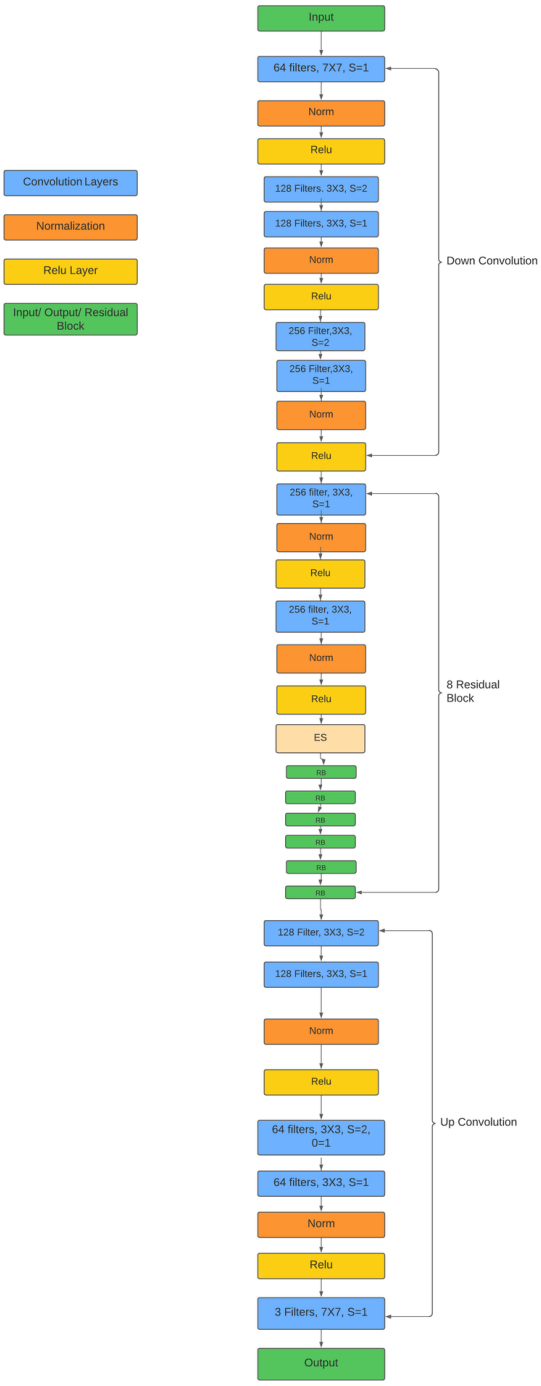

Fig. 2. Generative Network.Here s means stride and O indicates output padding and norm and ES indicates Batch-Normalization layer and Element-Wise sum respectively

been applied . Pillow has been used to paste the edges back onto the original image after making the white background transparent. For real-world images, COCO dataset [11] has been used. 1k images of each type have been used for training.

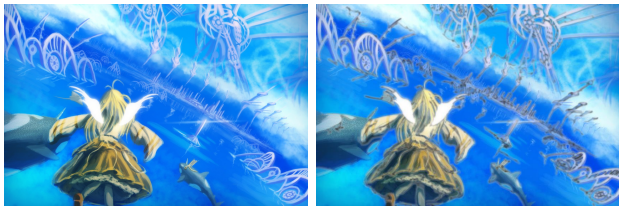

Fig. 3. Unsmoothed vs Smoothed Image

2) Pre-Processing: For every type of input picture (cartoons, smoothed cartoons and real-world photos), a data loader is constructed to process the images and divide the sets into training and validation sets by a 90/10 ratio. The image is 256x256 pixels in size.

3) Define Model: The Models for discriminator and generator have been defined.

4) Feature Extraction using VGG-16: A spatial constraint between our results and the extracted structure representation by has been established using high-level features retrieved by a pre-trained VGG16 network.

5) Training: Adam Optimizer has been used for training. The hyperparameters which are used are batch size=16,lr = 0.0002,beta1 = 0.5,beta2 = 0.999. The model has 210 epochs where initial weights are 10. The model has been trained using different initial weights and epochs. But the best result has been got after using 210 epochs which are shown in the Fig. 3 and Fig. 4.

6) Saving Output: After every training round the outputs are saved in a Tensorboard.

B. Network Architecture

Here in the Fig 2,the generative model is given. Here For the generative networks ,the architecture which has been implemented starts with a flat convolution stage and continues with two down-convolution blocks.In the flat convolution block the size is same as the input size by using stride=1.On the other hand ,in the down convolution, the size is halved by using stride of 2. Here Conv2d have been used for down-sampling.The following formula has been used to calculate image-sizes after each layer. Here Conv2d have been used for down-sampling.The following formula has been used to calculate image-size after each layer:

$$ImageSizeOutput =$$

$$(ImageSizeInput - KernelSize + 2 * padding)/Stride + 1 \quad (6)$$

Here for 1st convolution layer:

$$ImageSizeOutput =$$

$$(256 - Kernel + 2 * Padding)/Stride + 1$$

(7)

Here, kernelsize(7*7),padding(3*3) and stride is 1. So,image-size is same after 1st convolution.But After the down-sampling, As the value of stride is 2,According to the formula the image-size is halved.After that ,8 residual blocks with the same layout consists of 2 convolution layer followed by two batch-normalization layers are then utilized to build the content and create various feature suggested in [3].After using the residual blocks the image sizes become 1/4th of the input image-size. To rebuild the output cartoon-style images, two up-convolution blocks with 3*3 kernels and two stridden convolution layer with stride of 2 and 1 respectively adding 1 outer-padding are utilized. For up-sampling the following formula is used . Imagesizeoutput equal to stride*(Imagesizeinput-1+kernalsize+2*padding).For example in the 8th convolution layer, according to the formula output will be

$$2 * (128 - 1) + 3 - (2 * 1) = 255 + 1$$

(outer-padding).

For the Discriminative networks, the architecture implemented by Yang Chen et al. [3] has been used.Here ,At first one flat convolution followed by a leaky Relu is used.After that ,two strided convolution layers are used to reduce the size of the input images.And At last ,convolution layer with a kernel size of 3*3 are used for classification.

C. Result Analysis

The safebooru cartoon dataset [10] of cartoon images, the smoothed version of the images along with real photos from the coco dataset [11] has been trained for about 210 epochs.

The real photo training dataset [11] along with the safebooru cartoon dataset [10] yeilds a specific cartoon stylization. Based on different art styles, this catoonization may be effectively learned through a similar process.

The first round of training for 210 epochs yielded different results in each epoch. Directly after the start of training the very first result shows a grid like image with no presence of clear or distinct shapes of the corresponding real image. However after around 10-20 epochs, the generated cartoon images start showing more characteristics similar to a cartoonized image as shown in fig 5.

For the first round of training the value of ω was set to 10 which is the weight value for content preservation from the real images to the comic style of the generated cartoon images. However this shows cartoon images that are less cartoonized and more realistic. Thus it fails to fulfill the purpose of creating cartoon images as shown in figure 4.

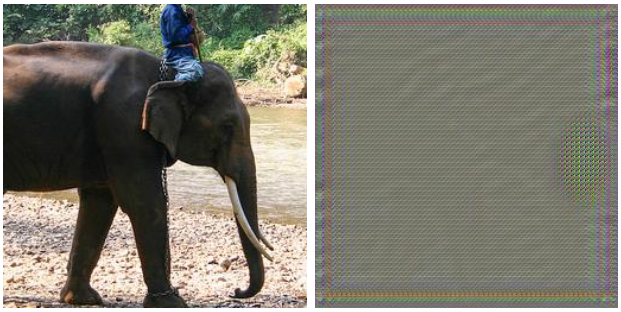

Fig. 4. Real Image Vs Generated Cartoon Image (Epoch 1 for ω=10)

Fig. 5. Real Image Vs Generated Cartoon Image (Epoch 210 for ω=10)

It Seems that the lower the value of ω actually is, the more balance is observed in the resultant cartoon images. For the next round of training, the value of ω was set to around 0.000005. Initially it showed similar results as figure 3. But later on, around 100 epochs, the generated images showed distinct shapes that mirrored the original or real images.

Fig. 6. Real Image Vs Generated Cartoon Image (Epoch 100 for ω=0.000005)

In the final stages of the training process, the images show much needed balance between the real and generated cartoon images. A comic style effect is observed in the cartoon images around 210 epochs.

The graph displays the relation between epochs and training loss for Generator and the discriminator. For better visualization, the x axis has been scaled to a logarithm of the epoch values. The Discriminator shows little loss throughout the training process. However, the Generator loss produced during training fluctuates throughout the epochs. For each iteration, a checkpoint is generated to make the training resumable for

Fig. 7. Real Image Vs Generated Cartoon Image (Epoch 210 for ω=0.000005)

Fig. 8. Real Image Vs Generated Cartoon Image (Epoch 210 for ω=0.000005)

more rounds of training with the saved weights and biases from previous rounds.

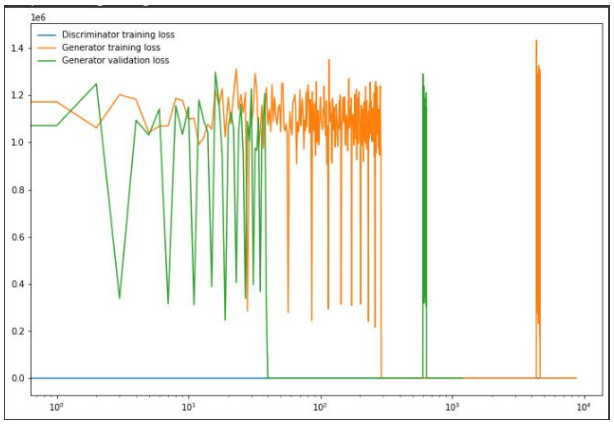

Fig. 9. Training and Validation Loss

VI. Conclusion and Future Work

In this paper, we suggested a Generative Adversarial Network that can convert real-world pictures into cartoon-style visuals. The results would have been better with the use of cartoon dataset of the same style. But this cannot be done due to the unavailability of datasets containing cartoons of a certain style. Furthermore, in future, we would like to look into how to better stylize cartoons of human faces by utilizing local facial traits. We would employ Conditional-GAN to accomplish this.

References

[1] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired imageto-image translation using cycle-consistent adversarial networks. In International Conference on Computer Vision, 2017

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] Chen, Yang, Yu-Kun Lai, and Yong-Jin Liu. "Cartoongan: Generative adversarial networks for photo cartoonization." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.

[4] Y. Chen, Y. -K. Lai and Y. -J. Liu, "Transforming photos to comics using convolutional neural networks," 2017 IEEE International Conference on Image Processing (ICIP), 2017

[5] Sultan, KM Arefeen, et al. "toon2real: Translating Cartoon Images to Realistic Images." 2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI). IEEE, 2020.

[6] Wang, Xinrui, and Jinze Yu. "Learning to cartoonize using white-box cartoon representations." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020.

[7] Black Mamba " Generating fake faces using GAN "

[8] Forrester Cole. " Cartoon Set "

[9] Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets." Advances in neural information processing systems 27 (2014).

[10] Alex Lamson "Safebooru: Anime Image Metadata", 2020

[11] Tsung-Yi Lin "Common Objects in Context",2021