CSCI 631: Project – Manga Colorization

Group Name: The Boys

Group Members: Vidit Kothari (vk3080), Karamcheti Pritham (kp5764)

Introduction

Manga, the esteemed form of Japanese comics, has traditionally been rendered in a monochromatic palette. This distinctive characteristic presents an intriguing challenge for colorization. In our current endeavor, we aim to infuse these classic black and white illustrations with vibrant hues through the application of Conditional Generative Adversarial Networks (cGANs). These advanced networks are adept at learning colorization from a solitary reference image. To augment the efficacy of this process, we incorporate supplementary image processing techniques, such as image segmentation, saturation enhancement, and color quantization. The forthcoming sections will guide the reader through the foundational concepts and elaborate on the intricacies of the neural network model employed in our project.

Generative Adversarial Networks (GANs)

A Generative Adversarial Network (GAN) is an innovative neural network architecture poised for data generation, modeled closely on the distribution of training data. This architecture bifurcates into two fundamental components: the generator and the discriminator.

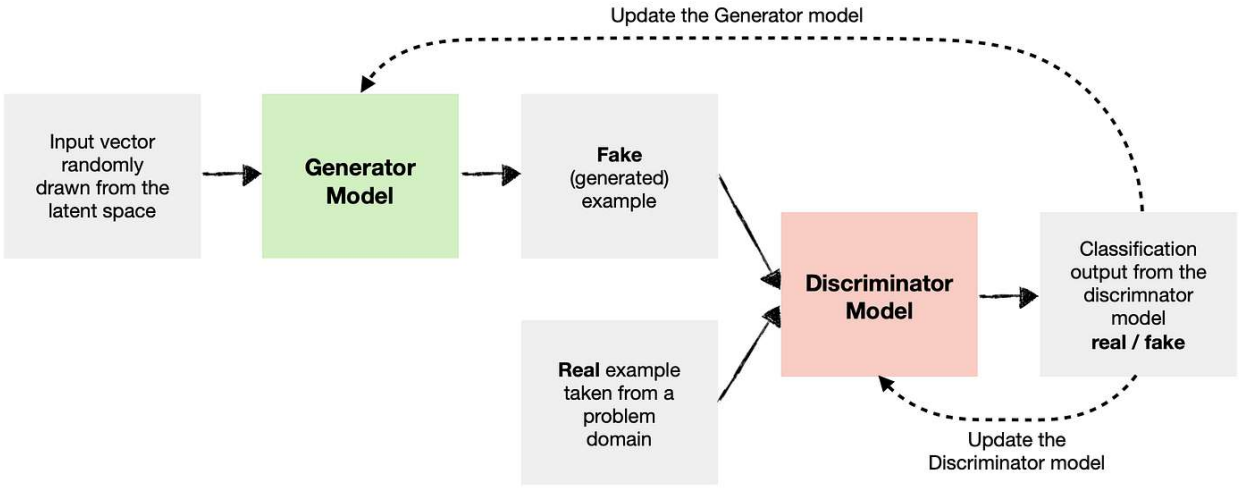
The generator assumes the role of an artist, crafting counterfeit data with the intent to mirror the genuine. In parallel, the discriminator operates as a discerning critic, tasked with differentiating the authentic from the forged. This dynamic establishes a competitive training environment where both entities continually evolve — the discriminator sharpening its acuity for real versus fraudulent data, and the generator striving to perfect its artistry. As training progresses, the generator advances, culminating in the production of data that is virtually indiscernible from the original dataset.

Conditional Generative Adversarial Networks (cGANs)

Expanding upon the foundational framework of Generative Adversarial Networks (GANs), Conditional Generative Adversarial Networks (cGANs) introduce a nuanced layer of specificity to the generative process through conditionality. These conditions may encompass a range of auxiliary information, such as categorical labels or tags, which steer the generative process

towards a targeted outcome. The objective transcends mere data generation; cGANs are orchestrated to fulfill explicit conditions, thereby tailoring the output more precisely.

The utility of cGANs becomes apparent in scenarios that demand a heightened level of direction in the creation process. They excel in sophisticated tasks that require a fusion of context and creativity, such as converting sketches to detailed images or weaving complex narratives into visual art. Their application in image-to-image translation and text-to-image synthesis underscores their transformative capability, making them invaluable tools in the domain of controlled, conditional data creation.



Implementation Overview

The implementation framework is composed of two primary components: image processing and image generation. The initial outputs from the CGAN (Conditional Generative Adversarial Network) are often marred by color inaccuracies. The image processing phase aims to enhance these raw images, focusing on improving saturation and achieving color uniformity.

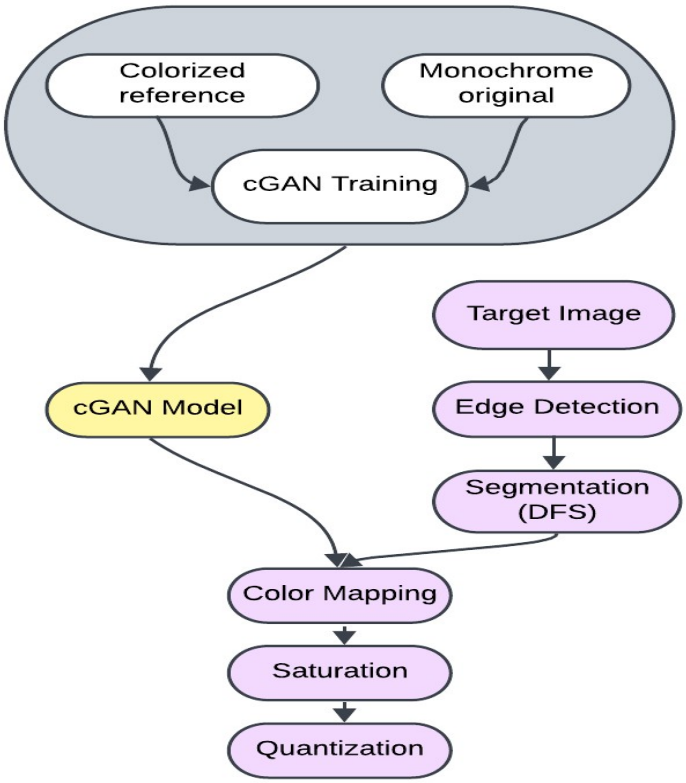


Image Generation

In our project, we aimed to understand the cGAN architecture in detail by implementing it from scratch. We first went through various online resources to understand the intricacies of the model. The cGAN model comprises of essentially 2 different parts, the generator and the discriminator.

Design Decisions

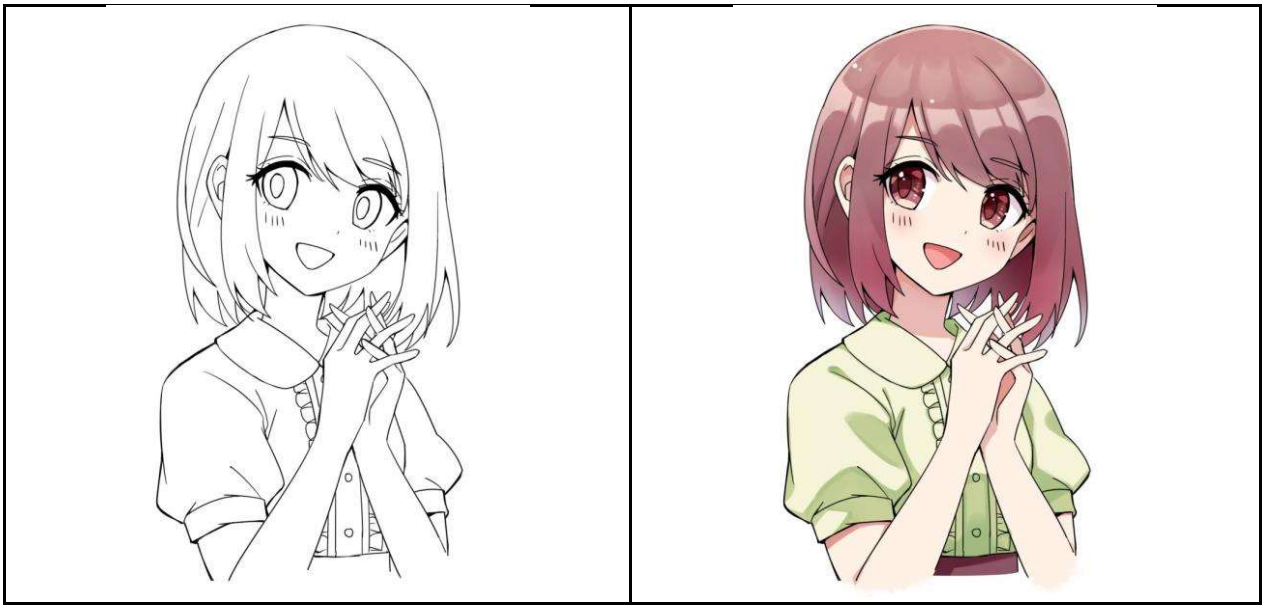
One of the first design decisions we had to make was to decide the inputs to the generator and discriminator. In the reviewed literature online, there were different implementations of the generator, where in some sources, the generator was taking an embedded label, which helped the generator create an image which corresponded to the label. This was more for use cases where there are multiple labels, like data generation using MNIST. Since our use case was simply to color an image given a sketch and a colored image, the black and white image was enough to

work as an input to the generator. In an extension, where the model can be used to colorize multiple characters, the architecture could be modified to specify the character they are interested in coloring.

The discriminator design was more or less in sync with most of the guides we referred to. The discriminator took the sketch and the generated image, as well as the sketch and the colorized image. This helped the discriminator to understand which image 'fake' and which image was 'real'.

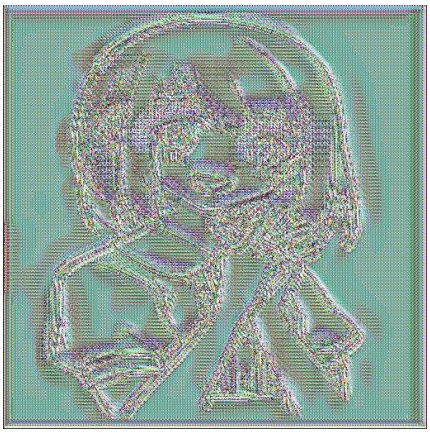
We also had to decide the loss functions and the optimizers to be used. This was given to us by our reference paper and their citations. We used the Binary Cross Entropy Loss for the discriminator, and a combination of L1 loss and Binary Cross Entropy Loss for the generator. By using two loss functions for the generator, we were essentially 'penalizing' the generator based on the discriminator's output of the generated image, as well as how different the generator's image was from the original. This was aimed at training the generator to make images that could 'fool' the discriminator, while being close to the ground truth. The optimizer used for both the generator and discriminator were Adam optimizers.

We also had to decide on the data we would use for the training. Since our reference paper uses just one training image to train their model, we decided to try to replicate their approach. However, obtaining images was a little bit of a challenge. We then found an open-source anime website which had line arts as well as colored images of the same characters in the same image. We identified three pairs of images for our training. Out of the three, we decided to select the simpler one, given the complexity and hardware constraints of the project. The sketch and colorized image we decided to use are as follows.



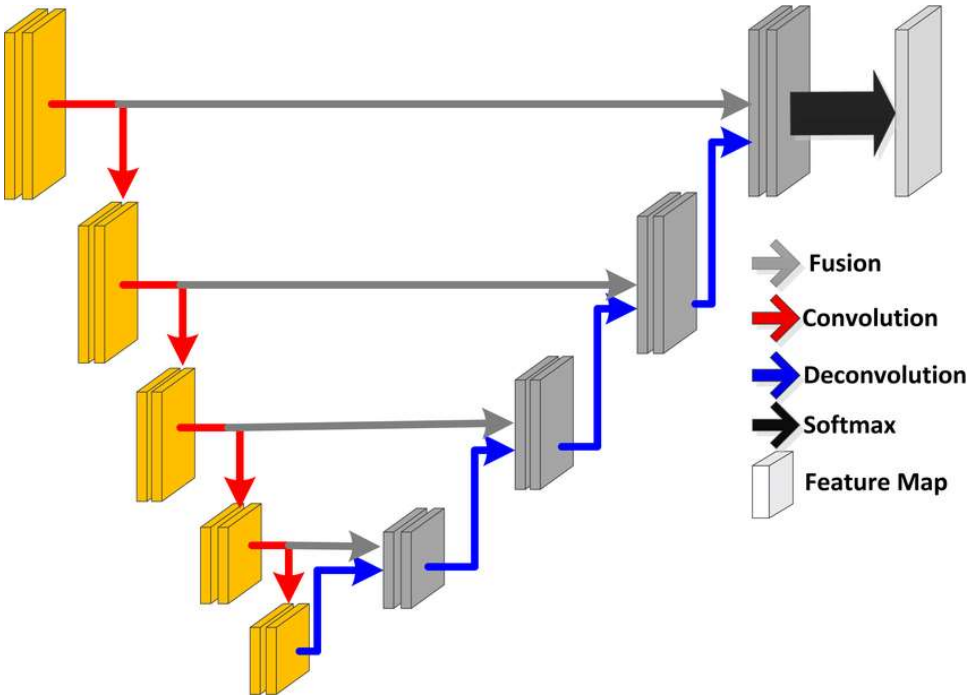
The Generator Implementation

To start with, we used a simple architecture of the generator, which involved an encoder and a decoder. The encoder aims to understand the features of the input image, while the decoder will try to reconstruct the image given the encoded output. The resulting simple encoder decoder architecture was not performing well, and we were getting a non-deterministic output. A sample of the output using a simple architecture can be seen below.



As you can see, the generator could not fill the color. It was giving an image filled with artefacts and was not learning well. On further research, we realized that the model was too simple to achieve a task like colorization. We discovered that the encoded features were not being decoded well using a simple architecture. We needed to map the feature representations in the encoder to the decoder to help the decoder generate a good image.

On exploring the citations in our reference paper, we came across a generator architecture called U Net. The U Net architecture introduces skip connections between the encoder and decoder, with a set of upsampling layers between the encoder and decoder to map the features detected in the encoder. Here is a diagram of the U-Net architecture.



U-Net Architecture

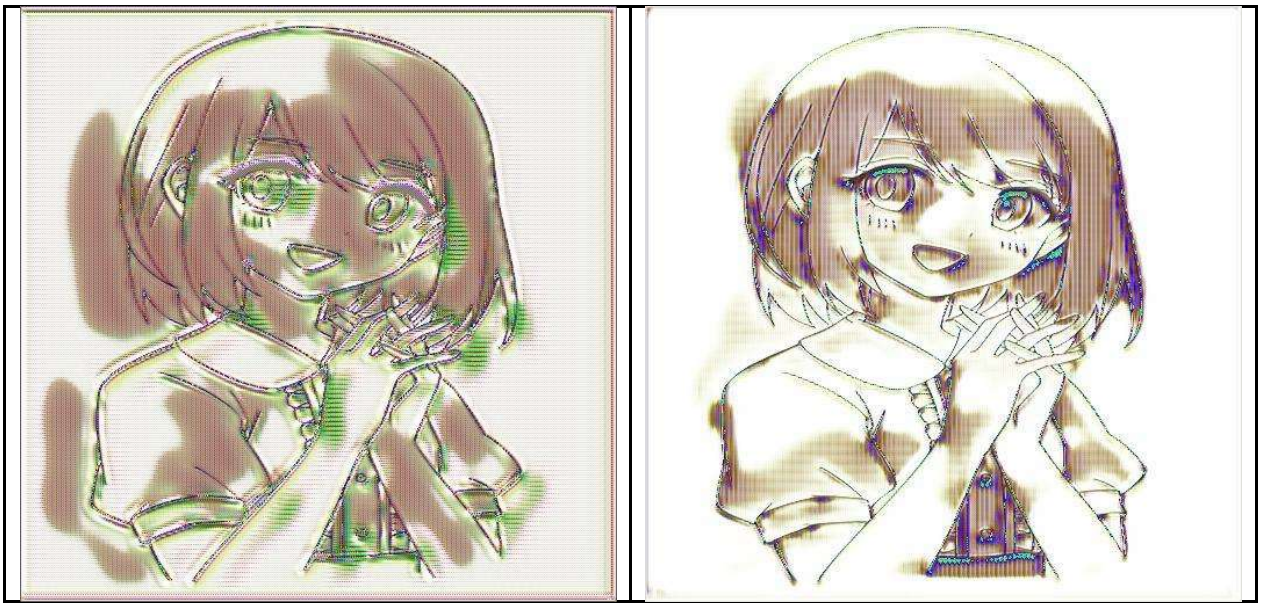
In the diagram, the red arrows represent the convolution layers, using the nn.Conv2d layer. The grey arrows represent the skip connections, that map the features detected by the decoder to the same level in the encoder. The blue arrows are deconvolution operations, or upsampling operations, which are done by a combination of nn.ConvTranspose2d and MaxPooling layers. Since the architecture resembles a U shape, it is called the U net architecture.

The Discriminator Implementation

The discriminator was a simple convolution network, with two inputs, the generated or true image, and the black and white image. The discriminator's task was to identify the real and fake images. The output of the discriminator was a single value, depicting its confidence of the realness of the image. This was done by mapping the convolutional outputs to a fully connected layer, and then applying a sigmoid function on it.

Results

The results we got were not as expected. The model was able to identify the colors in the true image but was not able to fill the image with the colors. Here are a few results of our model with various hyperparameters.



Hurdles, bugs, and setbacks

Since we were implementing the cGAN architecture from scratch, apart from the architecture decision outlined in the previous section, our team had to solve a lot of bugs, as well as change the model training method a few times.

The first hurdle we came across was creating the network itself. Since the network contains a lot of convolution and deconvolution layers, we had to carefully map the output of one layer to the output of another, especially when defining the generator. Initially, following the reference papers, we created the architecture using their given definition, later to realize that the layers were not mapping to each other. After this, we decided to step by step calculate the output dimensions of each layer(convolution, deconvolution and skip connections) so that the training can go smoothly. We used the formulae in the official pytorch documentation to calculate the dimensions of each layer.

We also had to maintain a standard size of the output image, which threw a lot of errors. Since designing a convolutional layer to exactly output the dimensions we needed was difficult, our team used the F.interpolate method to resize the image to the desired size.

Our major setback was the inaccurate results. We believe that the inaccuracies arised from two main problems.

1. Lack of quality training data
2. Mode Collapse – a characteristic problem of cGANs.

While trying to identify why the model did not work as expected, we saw that for a variety of parameters, the loss values of the generator and discriminator were stabilizing and not improving.

Moreover, the generator was creating a similar output even when the training(colorized) image was given. Based on these evidences, we believe that our model suffered from mode collapse.

Image Processing Techniques

Denoising: We apply a Gaussian Kernel in a correlation operation to mitigate noise, preserving the essence of the image while reducing graininess.

Difference of Gaussian: This step is designed to retain high-level details. By combining it with edge detail extraction, we generate binary image data that maintains the integrity of the original image structure.

Edge Detection: Utilizing various edge detection algorithms—such as Sobel, Prewitt, and Canny we extract delicate edge contours. These contours serve as a sketch-like detail within the training dataset.

Segmentation: We apply segmentation to the foundational binary image using a depth-first search algorithm. Each traversal identifies distinct components, which are instrumental in localizing the colorization process relative to the generated image.

Color Mapping: The segments identified through segmentation are then mapped onto the generated image with color. We color each segment by averaging the hues in the corresponding area of the colored image.

Saturation Enhancement: Post color mapping, images often exhibit reduced saturation. To counteract this, we enrich the image by applying histogram equalization on the saturation channel in HSV space.

Color Quantization: To achieve seamless color transitions across segments, we employ a quantization process. Each pixel is classified and then processed through a K-Means clustering algorithm to identify k dominant colors. Subsequently, each pixel is reassigned to the nearest color center, ensuring a cohesive and polished look.

Ethical Considerations in Manga Colorization

As we developed our machine learning model for colorizing monochrome manga images, we encountered several ethical challenges, each warranting careful consideration.

Limited Open-Source Availability: The scarcity of openly available monochrome manga images presents a significant challenge. Due to this limitation, our model's training dataset might not be sufficiently diverse, leading to a potential bias in the colorization results. This could add a lack of accuracy or diversity in the color palettes applied to certain types of images or scenes.

Character Bias in Colorization: When provided with only a few samples of any character, the model may exhibit inconsistencies in colorization, assigning different color schemes in separate instances. This inconsistency can be particularly problematic when the same characters tend to recur, or certain information is to be maintained across images like school, military or team uniforms.

Intellectual Property Theft: Once perfected, our model could potentially be used unethically for repurposing and republishing colored versions of copyrighted manga. This constitutes a significant ethical dilemma, as it encroaches on the original creators' intellectual property rights. It's crucial to consider safeguards against such misuse, including implementing strict usage policies and perhaps technical measures to prevent the model from being used on copyrighted works without authorization. significant ethical dilemma, as it encroaches on the original creators' intellectual property rights. It's crucial to consider safeguards against such misuse, including implementing strict usage policies and perhaps technical measures to prevent the model from being used on copyrighted works without authorization.

References

1. Hensman, P., & Aizawa, K. (2017). CGAN-based manga colorization using a single training image. *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR).*
2. Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*
3. Morevna Project, 2016
4. A tutorial on cGANs