TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING

Khwopa College of Engineering
Libali, Bhaktapur
Department of Computer Engineering

A FINAL REPORT ON

AUTOMATIC COLORIZATION OF GREYSCALE JAPANESE COMICS USING GANS

Submitted in partial fulfillment of the requirements for the degree

BACHELOR OF COMPUTER ENGINEERING

Submitted by

Abhinas Regmi	KCE077BCT005
Abi Shrestha	KCE077BCT006
Richard Khewa Limbu	KCE077BCT025
Safal Adhikari	KCE077BCT030

Under the Supervision of
Er. Niranjan Bekoju

Khwopa College of Engineering
Libali, Bhaktapur

CERTIFICATE OF APPROVAL

This is to certify that this minor project work entitled **"Automatic Colorization of Greyscale Japanese Comics using GANs"** submitted by Abhinas Regmi(KCE077BCT005), Abi Shrestha (KCE077BCT006), Richard Khewa Limbu (KCE077BCT025), and Safal Adhikari (KCE077BCT030) has been examined and accepted in the partial fulfillment of the requirements for the degree of Bachelor in Computer Engineering. The project was carried out under special supervision and within the time frame prescribed by the syllabus.

.....
Er. Kishor Kumar Adhikari, PhD
External Examiner
Associate Professor
Department of Computer and
Electronics Engineering
National College of Engineering

.....
Er. Niranjan Bekoju
Project Supervisor
Lecturer
Department of Computer Engineering
Khwopa College of Engineering

.....
Er. Dinesh Gothe
Head of Department,
Department of Computer Engineering
Khwopa College of Engineering

Copyright

Acknowledgement

We take this opportunity to express our deepest and sincere gratitude to our supervisor Er. Niranjan Bekoju, for his insightful advice, motivating suggestions for this project and also for his constant encouragement and advice throughout our Bachelor's program.

Also, we would like to thank our HoD Er. Dinesh Gothe for providing valuable suggestions and for supporting the project.

Abhinas Regmi	KCE077BCT005
Abi Shrestha	KCE077BCT006
Richard Khewa Limbu	KCE077BCT025
Safal Adhikari	KCE077BCT030

Abstract

In the ever-evolving realm of entertainment, Manga and Comics, traditionally presented in the classic black and white format, are increasingly sought after in their colorized renditions. The demand for colourized Manga, underscores the importance of expediting the labour-intensive manual colorization process, prompting the exploration of innovative, automated solutions. In response to this imperative, we propose the development of an advanced model designed to streamline and accelerate the colorization of Manga and Comics.

Our proposed methodology utilizes unpaired datasets, to train a cycle Generative Adversarial Networks (cycleGANs) based model. Our model can learn the mapping between black and white and colourized counterparts without requiring strict image correspondences in the training set. This not only ensures the effective colorization of new Manga inputs but also tackles quintessential problems in existing methodologies. By researching the intersection of Japanese Comics, GANs, and deep learning, our proposal seeks to contribute significantly to the automated colorization landscape, fostering efficiency and creativity in the production of visually captivating Manga and Comics.

Keywords: *Colorization, Manga, cycleGAN, Image Processing, Japanese Comics, Neural Network, Deep Learning*

Contents

List of Figures

List of Tables

List of Abbreviation

Abbreviations	**Meaning**
AI	Artificial Intilligence
API	Application Programming Interface
CNN	Convolutional Neural Network
CSS	Cascading Style Sheets
CUDA	Compute Unified Device Architecture
cGAN	Conditional Generative Adversarial Network
GAN	Generative Adversarial Network
HTML	HyperText Markup Language
JS	JavaScript
ML	Machine Learning
NN	Neural Network
RDP	Remote Desktop Protocol
ResNet	Residual Network
REST	Representational State Transfer
SSH	Secure Socket Shell
BCE	Binary Cross Entropy
MSE	Mean Squared Error
PSNR	Peak Signal-to-Noise Ratio
SSIM	Structural Similarity Index Measure

Chapter 1

Introduction

1.1 Background Introduction

Manga are comics or graphic novels originating from Japan. Most manga conform to a style developed in Japan in the late 19th century, and the form has a long history in earlier Japanese art. The term manga is used in Japan to refer to both comics and cartooning. Outside of Japan, the word is typically used to refer to comics originally published in this country.

Manga is a unique art form, traditionally drawn in black and white and characterised by its distinctive style. It is quite different from regular illustration [7]due to following key points :

- **Exaggerated Expressions**: Large eyes, expressive features, and dynamic body language capture emotions vividly.
- **Unique Linework**: Flowing lines, hatching, and crosshatching create depth and texture.
- **Screen Tones**: Dot patterns add shading, texture, and atmosphere, sometimes replacing traditional shading.
- **Drawing Effects**: motion, fire and light effects, action effects Open edges or boundaries: introduce colour bleeding problem

The line styles, shading, and screen tones used in Manga are different from other types of non-realistic illustrations, and pose an entirely different array of challenges in colourization.

1.2 Motivation

Coloring Japanese comics from black-and-white to color is a challenging and time-consuming. This process often involves artists having to redraw certain parts or even start from scratch to add colors. This is because the original black-and-white versions were not designed with color in mind.

There are existing tools and techniques that can automatically add color to images. However, these tools don't work as well with Japanese comics due to its unique style and features that are different from other types of images, such as photo-realistic and non-photo-realistic images. Many require varying degrees of human intervention, and aren't fully automatic. [1, 6, 8]

Given these challenges, our goal is to develop a new tool specifically for coloring Japanese comics. This tool will be designed to understand the unique features of Japanese comics and add color accordingly. By automating this process, we aim to make the coloring process faster and easier.

Moreover, this tool will not only benefit the artists but also the readers. With color, readers will be able to enjoy these comics in a more engaging and immersive way.

1.3 Problem Definition

Image colorization in image processing is a complex task that involves the addition of colour to grayscale images. In the realm of realistic images, colorization is relatively straightforward due to the consistency of colours associated with objects. Convolutional Neural Networks (CNNs) trained on large datasets of paired grayscale and colour images are commonly employed for this purpose [9–11]. These models learn the associations between grayscale patterns and corresponding colours, making predictions based on established colour conventions. For instance, grass is typically associated with a green hue, and the sky is often coloured blue. In realistic images, these conventions provide a reliable basis for the colorization process. Moreover, when it comes to using handcrafted algorithms for manga colorization, the results often appear flat and unappealing [5]

But the problems with non-realistic graphics are different. Non-realistic images involve abstraction, stylization, and subjective artistic interpretation, in contrast to their realistic counterparts. In these kinds of photos, colours can be quite random and unconventional. The absence of consistency in object representations could result in ambiguity during the colouring process. The process gets more challenging by the variety of textures, brushstrokes, and compositional features. The impact of artistic trends, narrative considerations, and genres on non-realistic images requires the use of specialised techniques [7]. While realistic image solutions can serve as a source of inspiration, it is not flexible enough to accommodate a wide range of artistic genres with deliberate alterations and artistic decisions. [1]

Being said that, manga poses another set of problems on top of being non-realistic illustrations.

- Characters within a page at different panels need to have consistent colours
- Different hatching, shading and screen tones needs to be coloured according to the scenario
- Open and not so well defined edges can be prone to colour bleeding
- Multiple texts and text blobs in the image should be treated differently.

1.4 Goals and Objectives

The main objective of this project are:

- To automate and speed up the process of colouring manga without human intervention.

1.5 Scope and Applications

Colorized manga is appealing to a wider audience, including people who are not familiar with Japanese culture. The following points highlight the scope and application of automatic manga colorization.

- It can be used to color both new and existing manga series in art styles of choice based on the dataset it has been trained on.
- Readers can use the system to colorize different manga according to their preference.

Chapter 2

Literature Review

2.1 Image-to-Image Translation with Conditional Adversarial Networks [1]

Conditional Adversarial Networks are a promising approach for many image-to-image translation tasks, especially those involving highly structured graphical outputs. The paper introduces a conditional variant of GANs, which can be used for image-to-image translation tasks. The model learns a mapping from input images to output images. However due to lack of definite structure in manga and training data pairs this architecture fails to adapt for our purpose.

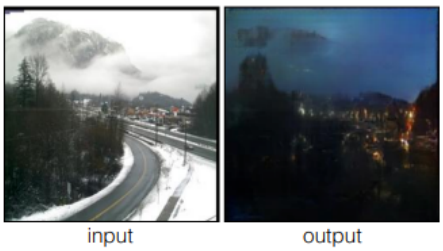


Figure 2.1: Realistic image colorization

source: https://arxiv.org/pdf/1611.07004v1.pdf

2.2 Manga Colorization Using Reference Images [2]

Colourizing manga by tracing pattern continuity and intensity continuity. Intensity continuity to avoid colour bleeding. This is manual and proper edge detection is not possible with the pattern continuity method.

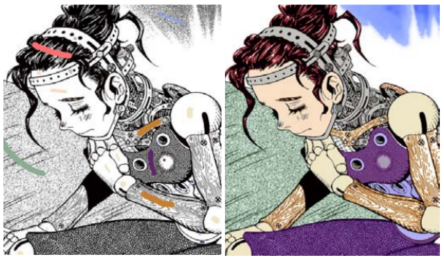


Figure 2.2: Colorization using pattern continuity

source: https://www.youtube.com/watch?v=HpZUOq3O64s

2.3 cGAN-based Manga Colorization Using a Single Training Image [3]

They used cGAN based approach for colorization along with segmentation of manga to produce high resolution and clean images.This method fails to generalise colorization for diverse characters and suffers heavily from colour monotonicity.

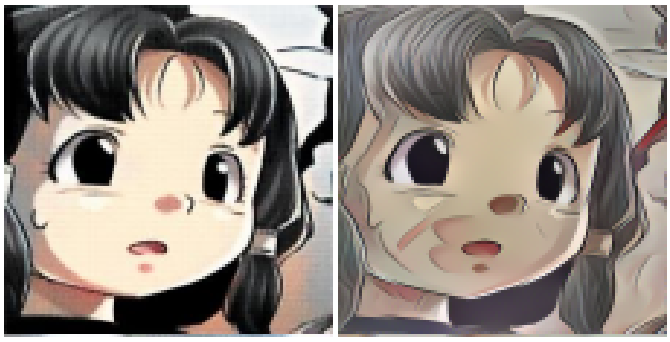


Figure 2.3: Colorization using single image reference

source: https://arxiv.org/pdf/1706.06918.pdf

2.4 Style Transfer for Anime Sketches with Enhanced Residual U-net and Auxiliary Classifier GAN [4]

Style maps and colour hints must be provided by the user for this network to perform colorization. Also the pretrained VGG is used which performs best for classification, but not for paintings.

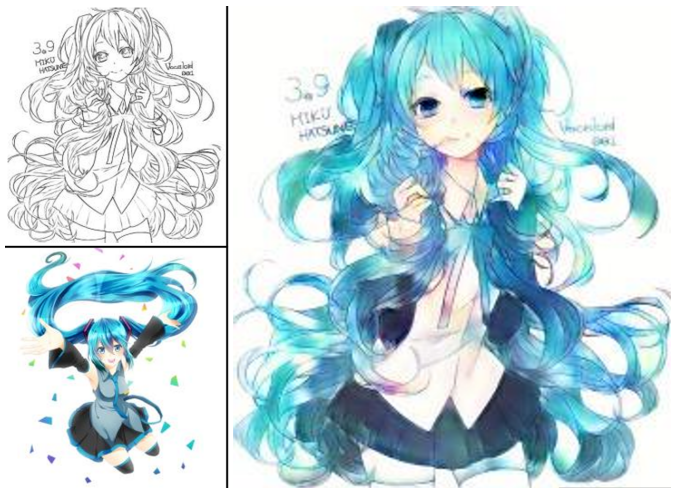


Figure 2.4: Colorization using style transfer method.

source: https://arxiv.org/pdf/1706.03319.pdf

2.5 inkn'hue: Enhancing Manga Colorization from Multiple Priors with Alignment Multi-Encoder VAE and StyleGAN [5]

Failed to fully automate the colorization process as a colour hints and partially coloured image was required as input for custom post processing.

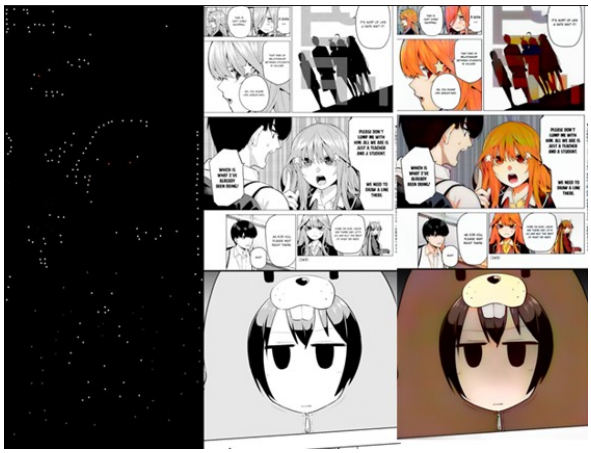


Figure 2.5: Semi-automatic colorization using multi-encoder VAE and StyleGAN

source: https://arxiv.org/pdf/2311.01804.pdf

2.6 Semi-automatic Manga Colorization Using Conditional Adversarial Networks [6]

If some object has several suitable colours, the neural network will produce a value close to the average of these colours that leads to degradation of the results, and monotone.

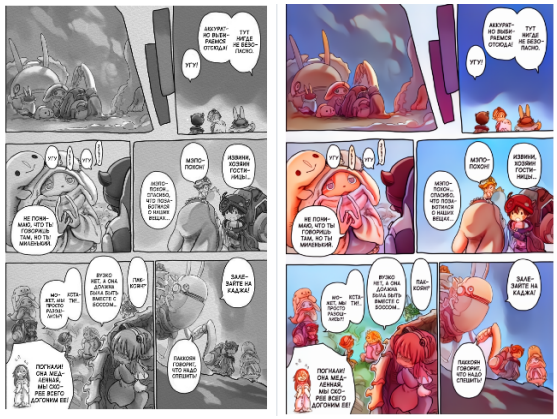


Figure 2.6: Semi-automatic colorization using cGAN

source: https://github.com/qweasdd/manga-colorization-v2

2.7 Review Matrix

S.N.	Title	Author(s)	Findings
1	Semi-automatic Manga Colorization Using Conditional Adversarial Networks [6]	Maksim Golyadkin and Ilya Makarov	A cGAN-powntinuities, allowing users to personalize results with color hints and fine-tune for stylistic accuracy.
2	Image-to-Image Translation with Conditional Adversarial Networks [1]	Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros	This paper introduces cGAN for image-to-image translation, enabling the generation of realistic images by conditioning on input images from different domains.
3	cGAN-based Manga Colorization Using a Single Training Image [3]	Kiyoharu Aizawa and Paulina Hensman	This method uses GANs to color entire manga pages, learning patterns and letting us personalize with hints and style tweaks.
4	Deep Extraction of x'Manga Structural Lines [7]	Chengze Li, Xueting Liu, And Tien-Tsin Wong	It is a data-driven approach that uses CNN to identify structural lines in pattern-rich manga.
5	Style Transfer for Anime Sketches with Enhanced Residual U-net and Auxiliary Classifier GAN [4]	Lvmin Zhang, Yi Ji and Xin Lin	a method to apply the style of a painting to an anime sketch.

6	Comicolorization: Semi-Automatic Manga Colorization [9]	Chie Furusawa, Kazuyuki Hiroshiba, Keisuke Ogaki and Yuri Odagiri	This paper semi-automatically colorizes monochrome manga images using reference images, maintaining consistent colors for characters across panels, and allows for user revisions.
7	Deep Residual Learning for Image Recognition [12]	Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun	This approach to convolution solves gradient vanishing problem allowing us to train very deep networks.
8	U-Net: Convolutional Networks for Biomedical Image Segmentation [13]	Olaf Ronneberger, Philipp Fischer, and Thomas Brox	Encoder and decoder architecture of U-net can be used for obtaining high resolution output and also enables better image segmentation.
9	A Style-aware Discriminator for Controllable Image Translation [14]	Kunhee Kim, Sanghun Park, Eunyeong Jeon, Taehun Kim, and Daijin Kim	The paper proposes a style-aware discriminator that acts as a critic and a style encoder to provide conditions which learns a controllable style space using prototype-based self-supervised learning and guides the generator. [14]
10	Wasserstein GAN [15]	Martin Arjovsky, Soumith Chintala, and Léon Bottou	This paper aims to solve mode collapse and hyperparameter sensitivity present in GANs using Wasserstein distance.
11	Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks [10]	Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros	Style transfer between two sets of unpaired image dataset is possible.

Table 2.1: Review Matrix with Research Papers, Authors, and Findings.

Chapter 3

Analysis

3.1 Requirement Analysis

3.1.1 Hardware Requirements

- High capacity **RAM** to handle memory-intensive tasks
- **Nvidia RTX series GPU** for CUDA based computations
- **SSD storage** for faster read/write speeds during image processing
- Additional high-capacity **external storage** for storing large datasets and image collections

3.1.2 Software Requirements

- **Python3.10** - a high-level, general-purpose programming language.
- **Torch** - an open source ML library used for creating deep neural networks and is written in the Lua scripting language.
- **Torchvision** - a library consisting of popular datasets, model architectures, and image transformations for computer vision.
- **FastAPI** - a modern web framework for building RESTful APIs in Python.
- **HTML/CSS/Javascript** - set of languages used for creating web pages
- **SSH/RDP** - network communication protocols to access the remote server for development purposes.
- **VSCode** - code editor for development.
- **Git/GitHub** - version control System and repository.
- **Ngrok** - a tool that creates a secure tunnel to the localhost
- **Tensorflow** - a machine learning framework for building and training neural networks.
- **Keras** - a high-level neural networks API running on top of TensorFlow that facilitates fast experimentation and prototyping.
- **Visdom** - a Python library for creating interactive visualizations and monitoring the training of machine learning model.

3.2 Feasibility Study

3.2.1 Economic Feasibility

The total expenditure of the project is just computational power. For this, the college has assisted us with a remotely accessible virtual machine that is sufficiently capable for our project. As for the other aspects of this project, none of them pose any economic cost. The process of acquisition and preparation of dataset required to train the model is explained in the methodology section.

3.2.2 Techical Feasibility

The dataset, acquired in accordance with the Fair Usage Policy, will be preprocessed. This involves transforming the colored images into grayscale. Given the complexity of our model and the specifications of our machine, we anticipate a training duration of approximately 60 hours.

3.2.3 Operational Feasibility

After the model is trained, a web interface will be created for user accessibility and interaction. Furthermore, a Python package will also be created for easy integration into other codebases.

3.2.4 Legal Feasibility

Manga are protected by copyright as works of art, making it difficult for large dataset for training. That's why we have gathered legally available dataset to train our model under Fair Usage Policy.

Chapter 4

Methodology

An iterative and incremental software development approach, such as Agile methodology, can be adopted. Agile methodologies emphasize flexibility, collaboration, and iterative development, which align well with the nature of the project and its potential changes and refinements.

4.1 Agile Methodology

Agile development is a flexible and iterative approach to software development that prioritizes adaptability and collaboration. It emphasizes delivering small, incremental releases of a product with the goal of responding quickly to changing requirements and feedback. Agile methodologies are based on the Agile Manifesto, which values individuals and interactions, working solutions, customer collaboration, and responding to change over rigid processes and tools.

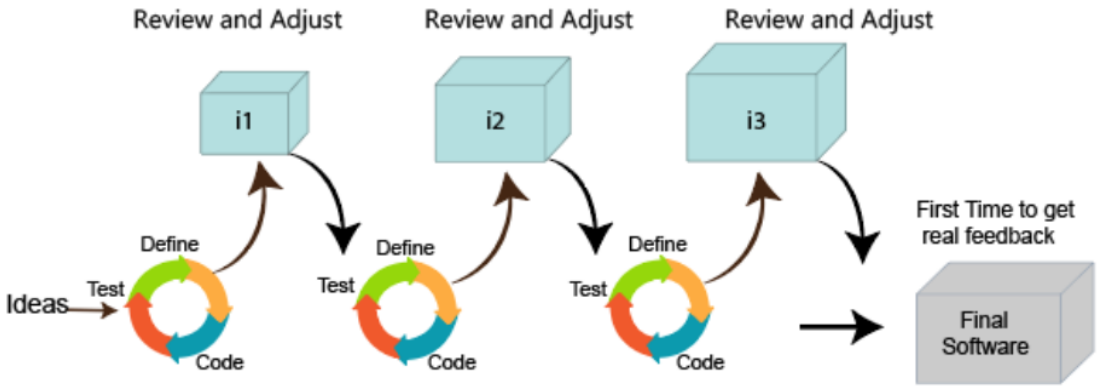


Figure 4.1: Agile methodology

source: https://www.javatpoint.com/advantage-and-disadvantage-of-agile-methodology

The Agile development process is typically organized into iterations or sprints, each of which includes the following key steps:

- **Requirements**: In Agile, requirements are expressed as user stories, which are short descriptions of a feature told from the perspective of the end user. Typically, the product owner collaborates with stakeholders to define and prioritize user stories, which in our case, are the objectives and highlights of the problem statement. These stories are added to the product backlog.

- **Design**: Design involves planning how the requirements will be implemented, considering the overall architecture, user interface, and other relevant aspects. During sprint planning, the development team discusses and plans how to address the problem statements and achieve the objectives. Design decisions are made collaboratively, and the team may create mockups or prototypes to visualize the solution.

- **Development**: This phase involves coding or implementing the solutions outlined in the design phase. Development work is carried out based on the tasks assigned to team members during the sprint planning. Continuous collaboration and communication are crucial to address any issues or changes that may arise.

- **Testing**: Testing is an integral part of Agile development and involves validating that the implemented features meet the specified requirements. Testing will be done parallely with development, creating test cases based on user stories and conducting various testing types (unit testing, integration testing, etc.) to ensure the quality of the code and output.

- **Deploy**: Deployment involves making the developed features available for use by end-users. The product increment is deployed to a staging or production environment. Continuous integration and deployment practices ensure that the latest changes are automatically integrated and deployed when ready.

- **Review**: At the end of each iteration, a review or demo is conducted to showcase the completed features to stakeholders. The team will participate in a review meeting to provide feedback, discuss any adjustments needed, and reprioritize the product backlog.

- **Launch**: The final step involves releasing the product increment to users. In our project, a web interface will be rolled at launch which will facilitate user interaction. Once all necessary adjustments are made based on feedback and the priorities based on objectives the product is officially launched. The cycle then repeats with a new iteration.

It's important to note that the design, development, testing, deployment, review, and launch steps occur iteratively and multiple times throughout the project. Agile development promotes continuous improvement and adaptation to changing requirements, allowing teams to deliver value to users more frequently and efficiently.

Chapter 5

System (or Project) Design and Architecture

5.1 Cycle-GAN

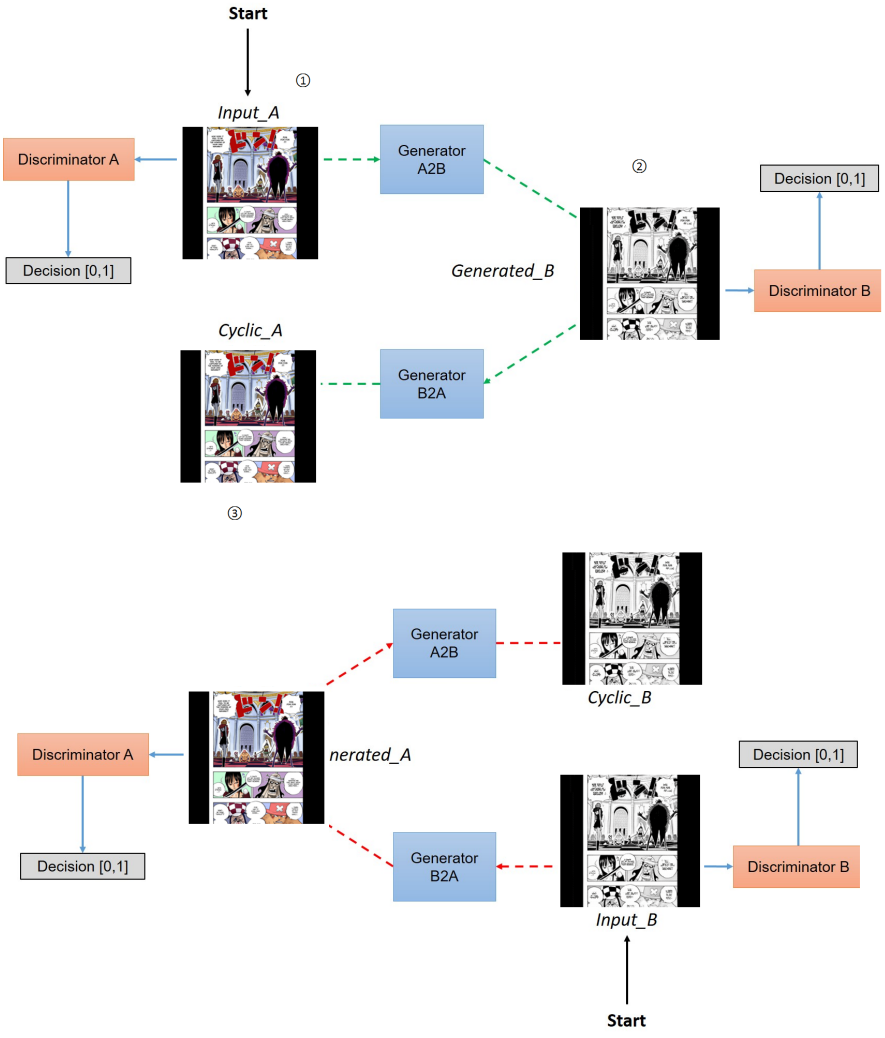


Figure 5.1: Cycle-GAN implementation for manga colorization

CycleGAN [16] is a model proposed for performing image-to-image translations without paired data. The key idea behind CycleGAN is to learn mapping functions between two domains A and B such that the distribution of images from domain A mapped to domain B is similar to the distribution of images in domain B, and vice versa . CycleGAN consists of two GANs - G: A $\rightarrow$ B and F: Y $\rightarrow$ X. G tries

to transform images from domain A to look like they came from domain B, and F does the opposite.

Along with the generators G and F, there are also two discriminators, DA and DB. DA tries to distinguish between images from domain A and generated images from F. Similarly, DB tries to distinguish between images from domain B and generated images from G. [10].

5.1.1 Generator

The ResNet9 generator is a variant of the ResNet architecture adapted for use in generative adversarial networks (GANs) for image synthesis tasks. It employs deep residual blocks with skip connections to capture complex patterns in input noise, while normalization layers stabilize training. Activation functions introduce non-linearity, enabling the model to learn intricate mappings between noise and generated images. The generator takes a black and white manga panel as input and learns to predict the corresponding colored version of the manga panel. The generator passes images through series of convolution layers with downsampling and upsampling to produce colored output. The generator architecture (figure 5.2) is constructed as in [17].

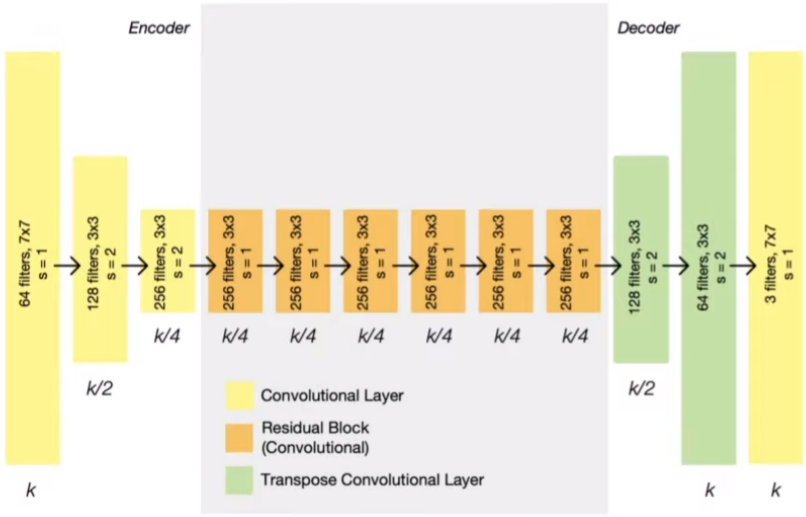


Figure 5.2: ResNet9 Generator Architecture

Input Processing

The input image is initially processed through a series of layers to extract features. It starts with a reflection padding layer followed by a convolutional layer with a kernel size of 7 and 64 output channels. Instance normalization is applied after the first convolutional layer, which normalizes the activations across the spatial dimensions of each channel independently for each sample in a batch.

Downsampling

After the initial processing, the generator performs downsampling to reduce the spatial dimensions of the features while increasing the number of channels. This downsampling is achieved through two sets of convolutional layers with stride 2, effectively halving the spatial dimensions of the features while doubling the number of channels. Instance normalization and ReLU activation are applied after each downsampling operation.

Residual Blocks

The core of the generator architecture consists of a series of residual blocks. These blocks aim to capture and refine image details across different scales. Each residual block contains two convolutional layers with 3x3 kernels, followed by instance normalization and ReLU activation. The input to each residual block is added to the output of the convolutional layers, allowing the network to learn residual mappings.

Upsampling

After the residual blocks, the generator performs upsampling to reconstruct the high-resolution output image. Similar to the downsampling stage, upsampling is achieved through two sets of transposed convolutional layers. These layers increase the spatial dimensions while reducing the number of channels. Instance normalization and ReLU activation are applied after each upsampling operation.

Output Layer

Finally, the upscaled features are passed through a reflection padding layer followed by a convolutional layer with a kernel size of 7 to generate the output image. The output image is normalized using the hyperbolic tangent (Tanh) activation function, which maps the pixel values to the range [-1, 1].

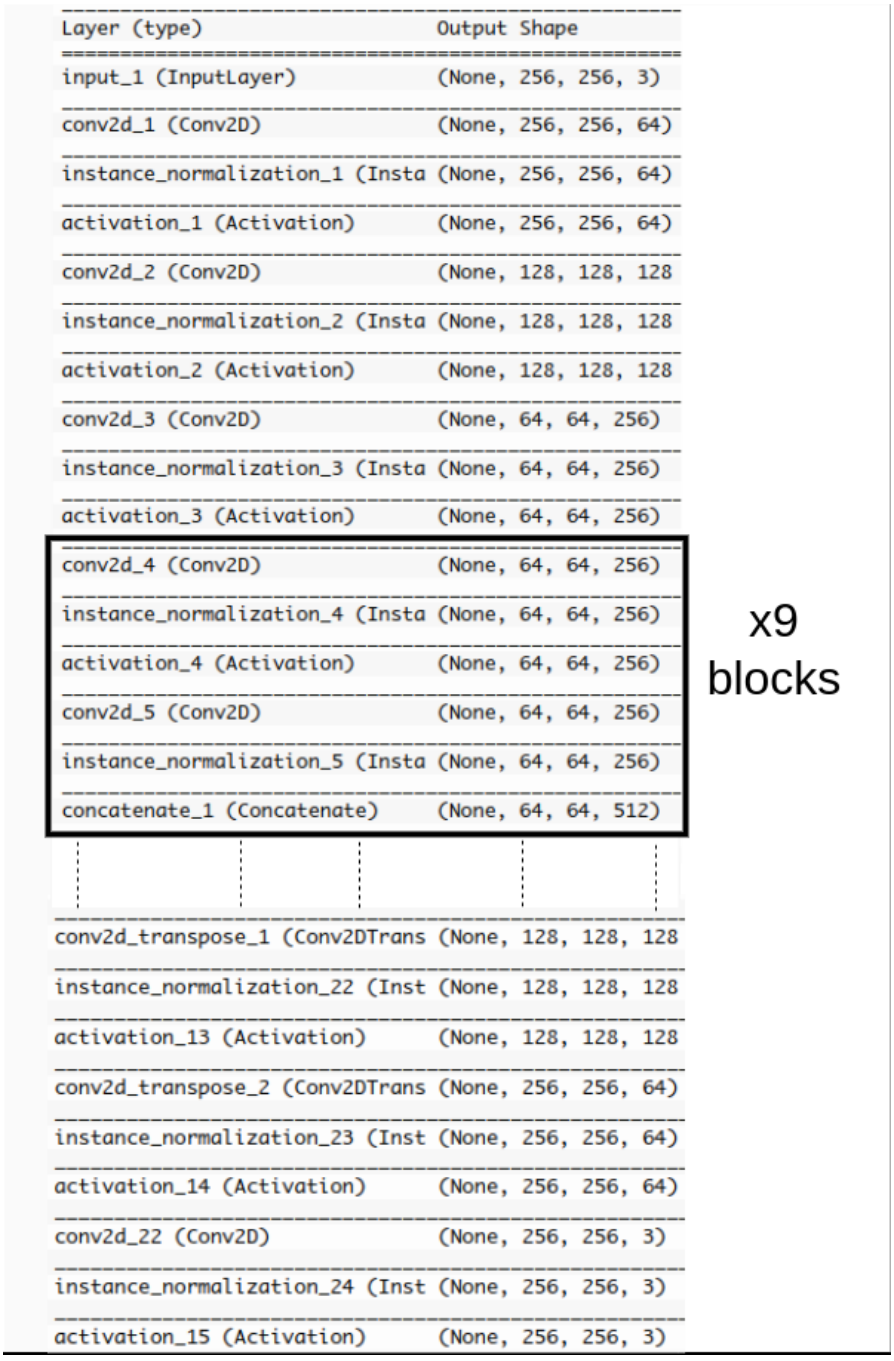


Figure 5.3: Generator Architecture Summary

5.1.2 Discriminator

The PatchGAN discriminator [10] in CycleGAN is designed to discriminate between real and fake images at the patch level rather than the image level. It consists of convolutional layers with decreasing spatial dimensions and increasing channel depths.

Architecture

The architecture follows a pattern of convolutional layers with increasing channel depths: C64-C128-C256-C512. Each layer is followed by a normalization step, often using instance normalization, and a non-linear activation function, typically Leaky ReLU.

Patch-Level Discrimination

Instead of outputting a single probability for the entire image, the discriminator outputs a matrix of probabilities corresponding to different patches of the input image. This patch-based approach allows the discriminator to capture fine-grained details and textures in both real and fake images.

Final Layers

Although not mentioned in the paper [10], the discriminator includes a final hidden layer C512 with a stride of 1×1, followed by an output layer C1 also with a 1×1 stride, utilizing a linear activation function. This configuration enables the discriminator to produce an output feature map of size 16×16 when applied to an input image of size 256×256.

Training Objective

The discriminator aims to minimize the classification error between real and fake patches, effectively learning to distinguish between authentic and generated image patches.

By utilizing a PatchGAN discriminator, CycleGAN is able to enforce high-frequency structure consistency between input and output images, facilitating the translation of image style and domain adaptation tasks

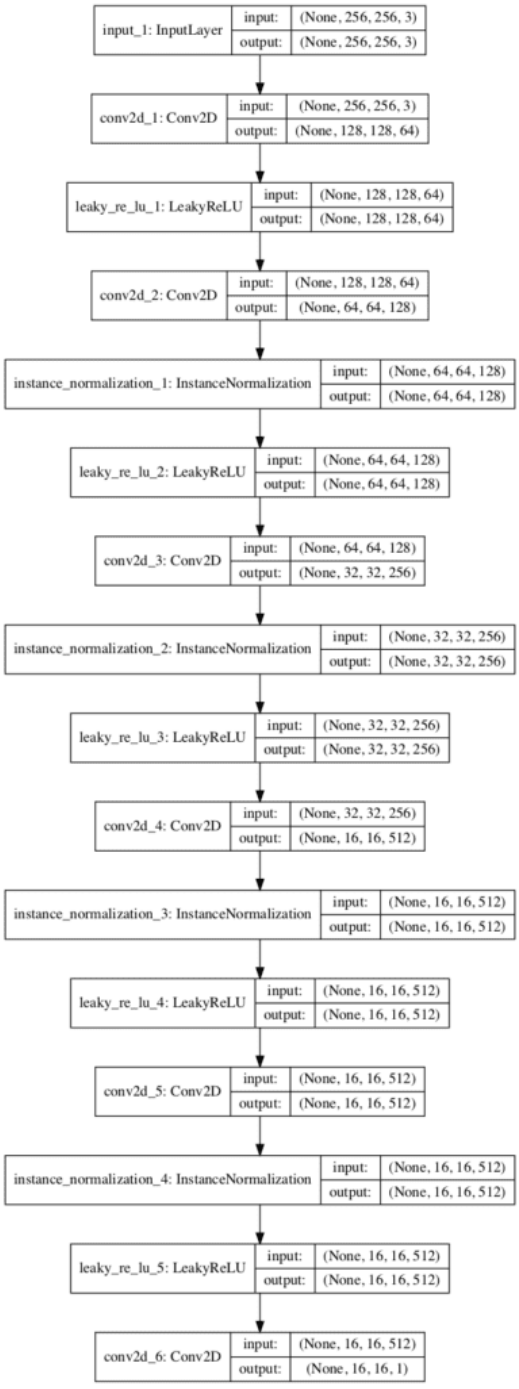


Figure 5.4: Discriminator Architecture

```
_________________________________________________________________
Layer (type)                 Output Shape
=================================================================
input_1 (InputLayer)         (None, 256, 256, 3)
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 64)
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 128, 128, 64)
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 128)
_________________________________________________________________
instance_normalization_1 (In (None, 64, 64, 128)
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 64, 64, 128)
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 256)
_________________________________________________________________
instance_normalization_2 (In (None, 32, 32, 256)
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 32, 32, 256)
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 512)
_________________________________________________________________
instance_normalization_3 (In (None, 16, 16, 512)
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 16, 16, 512)
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 512)
_________________________________________________________________
instance_normalization_4 (In (None, 16, 16, 512)
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 16, 16, 512)
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 16, 16, 1)
_________________________________________________________________
```

Figure 5.5: Discriminator Architecture Summary

5.2 Loss Function

To teach the model to generate similar colors for similar patterns and introduce a variety of colors, we use a mix of different loss functions.

- **L1 Loss**
  In order to enforce the generator to produce similar coloring to the ground truth, we use a pixel level L1 metric that calculates the distance between output image $\mathcal{G}\left(x, d, h, \mathcal{E}\left(x\right)\right)$ and ground truth image y.

$$\mathcal{L}_{L_1}^{\mathcal{G}} = \|\mathcal{G}\left(x, d, h, \mathcal{E}\left(x\right)\right) - y\| \tag{5.1}$$

  Where,

  - $\mathcal{L}_{L_1}^{\mathcal{G}}$ = Generator L1 Loss
  - $\mathcal{G}$ = Generator function
  - x = Input to Generator
  - d = Conditioning information
  - h = Hidden state
  - $\mathcal{E}(x)$ = Encoder function applied to input x
  - y = Ground truth image

- **Adversarial losses**
  For the mapping function G:X→Y and its discriminator D(y) , we express the objective as:

$$\mathcal{L}_{\mathcal{G}adv} = \mathbb{E}_{x,h}[log\left(1 - \mathcal{D}\left(\mathcal{G}_{main}\left(x, d\left(x\right), h, \mathcal{E}\left(x\right)\right)\right)\right)] \tag{5.2}$$

  where G tries to generate images G(x) that look similar to images from domain Y , while D(y) aims to distinguish between translated samples G(x) and real samples y. G aims to minimize this objective against an adversary D that tries to maximize it.

  We introduce a similar adversarial loss for the mapping function F : Y → X and its discriminator D(x) as well:

$$\mathcal{L}_{Dadv} = -\mathbb{E}_y[log\mathcal{D}\left(y\right)] - \mathbb{E}_{x,h}[log\left(1 - \mathcal{D}\left(\mathcal{G}_{main}\left(x, d\left(x\right), h, \mathcal{E}\left(x\right)\right)\right)\right)] \tag{5.3}$$

- **Cycle Consistency Loss**
  Adversarial losses alone cannot guarantee that the learned function can map an individual input x_i to a desired output y_i. To further reduce the space of possible mapping functions, the learned mapping functions should be cycle-consistent, for each image x from domain X, the image translation cycle should be able to bring x back to the original image, i.e., x $\rightarrow$ G(x) $\rightarrow$ F(G(x)) $\approx$ x. We encourage this behavior using a cycle consistency loss as found in [10]:

  $$\mathcal{L}_{cyc}(G,F) = \mathbb{E}_{x\sim p_{data}(x)}[\|F(G(x)) - x\|] + \mathbb{E}_{y\sim p_{data}(y)}[\|G(F(y)) - y\|] \quad (5.4)$$

  Where,

  - $\mathcal{L}_{cyc}(G,F)$ = Cycle consistency loss between generators G and F
  - $\mathbb{E}_{x\sim p_{data}(x)}$ = Expectation operator with respect to data distribution $p_{data}(x)$
  - $\|.\|$ = Normalization operation
  - $\mathbb{E}_{y\sim p_{data}(y)}$ = Expectation operator with respect to data distribution $p_{data}(y)$

5.3 Performance Evaluation Metrics

Due to the architecture and domain similarity, we can use the same metrics as in [18] to evaluate the performance of our model.

- **PSNR**:
  PSNR (Peak Signal-to-Noise Ratio) is a metric commonly used to measure the quality of reconstructed images or videos by comparing them to the original ones. It measures the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation.

  PSNR values are typically expressed in decibels (dB) and range from 0 to infinity. Higher PSNR values indicate higher image quality, with values closer to infinity representing perfect reconstruction.

  In practical terms, PSNR values above 30 dB are often considered excellent, while values below 20 dB may indicate significant loss of quality.:

  $$PSNR = 10 * log_{10}\left(\frac{MAX_I^2}{MSE}\right) l \quad (5.5)$$

  where, MAX is the maximum possible pixel value of the image, (like 255 for 8-bit image). Mean Square Error (MSE) is an objective criterion between predicted images and real images, which is represents as:

  $$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\|T(i,j) - F(i,j)\|^2 l \quad (5.6)$$

  m , n represent the size of the real image and the colorized image respectively.

- **SSIM**:
  SSIM (Structural Similarity Index Measure) is a metric used to quantify the similarity between two images. It takes into account three components: luminance, contrast, and structure, to evaluate how closely the images resemble each other perceptually.

  The SSIM values range from -1 to 1, where 1 indicates perfect similarity between the images, 0 indicates no similarity, and -1 indicates perfect dissimilarity. Typically, values above 0.9 are considered excellent, indicating high similarity between the images.

$$SSIM = \frac{(2\mu_x\mu_y + c_1)(\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{5.7}$$

$$c_1 = (k_1 L)^2 \quad c_2 = (k_2 L)^2 \tag{5.8}$$

  where

  - μ_x is the mean of image x,
  - μ_y is the mean of image y,
  - σ_x^2 is the variance of image x,
  - σ_y^2 is the variance of image y,
  - σ_{xy} is the covariance of image x and image y,
  - L is dynamic range of pixel values, usually L is equal to 255,
  - $k_1 = 0.01$ and $k_2 = 0.03$ which are constants for stability.

Chapter 6

Experiments

6.1 Data Collection

To train our CycleGAN model effectively, we collected a diverse dataset of manga images. We amassed a total of 53GB of data, consisting of 20GB of black and white images and 30GB of colored images in .cbz (Comic Book Zip) format. Each category comprised approximately 100 volumes, ensuring a balanced representation of both monochrome and colored styles. To ensure consistent colorization results across our dataset and avoid issues arising from disparate artistic styles, we made the deliberate decision to utilize manga images exclusively from the series "One Piece" for training our Cycle-GAN model.
This choice was driven by several key factors:

- **Uniform Art Style:**One Piece maintains a remarkable level of artistic consistency throughout its extensive volume count. This consistency provides the model with a well-defined framework for learning color palettes, shading techniques, and character depictions, ultimately leading to more faithful and visually unified colorization outputs.
- **Data Abundance:** One Piece boasts a vast wealth of training data, encompassing hundreds of chapters across diverse settings, characters, and emotional portrayals. This abundance allows the model to acquire a comprehensive understanding of the series' overall color scheme and apply it effectively to unseen images.

However, it is crucial to acknowledge the potential limitations associated with this data selection:

- **Generalizability:** Restricting the data to a single series may limit the model's ability to generalize effectively to other manga styles. Future exploration of transfer learning techniques could potentially address this limitation.
- **Data Bias:** Utilizing solely One Piece introduces potential biases based on the specific color choices and techniques employed within that series. Diversifying the dataset in the future could help mitigate this issue.

6.2 Data Preprocessing

To ensure the effectiveness of our CycleGAN model in colorizing manga images, we implemented a comprehensive data preprocessing pipeline specifically tailored to "One Piece" dataset. This process aimed to refine the raw dataset and provide the model with high-quality, consistent training material.

6.2.1 CBZ Extraction and Conversion

The initial step involved converting all collected CBZ files into individual PNG images. This conversion ensured compatibility with our Networks which accept images as inputs.

6.2.2 Page Filtering

For filtering at high level, we addressed the presence of RGB images in the dataset such as RGB cover pages, end pages, and special trivia and informative pages in the manga.

By effectively removing these colored pages, we ensured the model exclusively learned from black and white manga panels, preventing potential colorization inconsistencies arising from pre-colored content.

6.2.3 Panel Detection and Classification

Subsequently, we addressed the presence of non-manga pages 8.1 within the dataset which typically are:

- Empty pages
- Text-only or heavily text-biased pages
- Trivia and other non-manga-panel pages

To efficiently filter out these non-relevant pages, we employed a Sequential Convolutional Neural Network (CNN) classifier featuring three 2D convolutional layers followed by max-pooling layers, then flattened and connected to two dense layers for classification.
This classifier which has architecture as in figure 6.1 was trained on 69 bad manga panels and 900 good manga panels, with Precision: 0.9775, Recall: 1.0, Accuracy: 0.9797 on Validation data on 50 epochs of training.
With this CNN classifier, we could systematically identify and remove such non-manga pages from our dataset, ensuring that only relevant content was retained for further processing and analysis.

This refined selection process ensured the model was trained on a focused dataset specifically containing relevant manga artwork. Through these collection and preprocessing steps, we prepared a high-quality, filtered and consistent dataset for our Cycle-GAN model.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 255, 255, 16)	160
max_pooling2d (MaxPooling2D)	(None, 127, 127, 16)	0
conv2d_1 (Conv2D)	(None, 127, 127, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_2 (Conv2D)	(None, 63, 63, 16)	4624
max_pooling2d_2 (MaxPooling2D)	(None, 31, 31, 16)	0
flatten (Flatten)	(None, 15376)	0
dense (Dense)	(None, 256)	3936512
dense_1 (Dense)	(None, 1)	257

Figure 6.1: Architecture of Manga panel classifier

Figure 6.2: Bad pages as filtered out by the classifier

6.3 Model Training

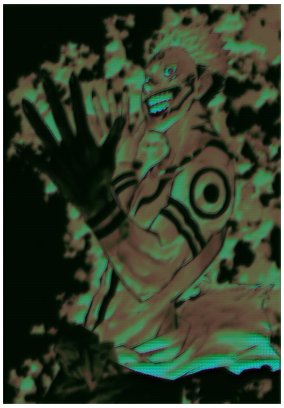
6.3.1 Training a cycle-GAN with UNET Generator

Initially, we trained a deep UNET with 3 skip connections. The model consisted of multiple Squeeze Extract ResNet. The UNET Model had a total of 32212514 trainable parameters.We used PatchGAN as our discriminator. We trained the model on OnePiece Manga Dataset with Cycle-GAN architecture.
We trained the model for 200 epoch with following parameters:

- Learning rate for Color and BW Generators = 1e-4
- Learning rate for Color and BW Disciminator = 4e-4
- Batch Size = 2
- White Color Penalty Threshold = 0.8
- Input Image Width = 112
- Input Image Height = 112
- Model Initialization = Random Initialization
- Optimizer = Adam
- Adam Optimizer Betas = 0.5, 0.999
- Lambda Cycle = 10

We evaluated L1 loss, cycle consistency loss and adverserial loss during the training for both generators and discriminators. Due to its architecture, U-Net Genrator was able to presever the features properly but failed to colorize the image as per our liking. Also, due to the complexity and depth of the generator, we could not accomodate images of size larger than 256x256 in the VRAM.
The losses and output during training of the generator is shown below.

(a) Image Input to UNET



(b) Image Output from UNET

Figure 6.3: Output of UNET Generator

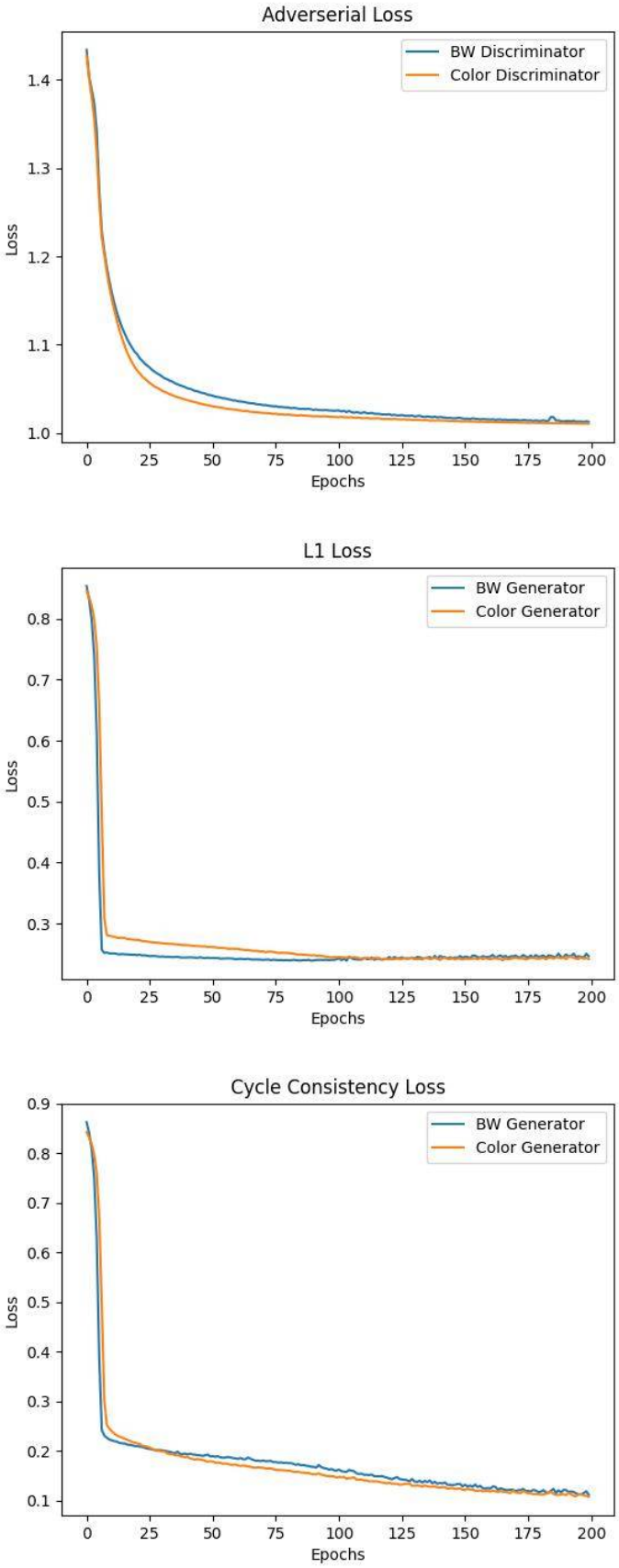


Figure 6.4: Losses for generators and discriminators.

6.3.2 Training a Cycle-GAN with Resnet9 Generator

We realized the simpler ResNet9 generator architecture was able to preserve feature properly (not as much as U-net) and also should be able to colorize as per our liking. We trained cycleGAN with a ResNet9 generator for both Color, and Black and White Generator. We used a 16x16 PatchGAN as the discriminator. Due to resource and training time constraints, we trained the model with 362 images in both domains from OnePiece Manga Dataset for 200 epochs.
Initially, the images were resized without preserving the aspect ratio of original panel before passing to the model during training. The image outputs at an intermediate training stage have been given in **??**, which shows the loss of structural information due to resizing. Consequently, when using the so-trained model for inference, it was able to properly color resized images in the order of 256x256 and failed to appropriately fill larger images in original size or aspect ratio.

So, the model was trained on unresized and cropped images with following parameters.

- Number of Epoch = 200
- Batch Size = 1
- Learning Rate for Generator = 0.0002
- Learning Rate for Discriminator = 0.0002
- Input Image Width = 256
- Input Image Height = 256
- Model Initialization = Normal Initialization
- Initialization Gain = 0.02
- Optimizer = Adam
- Adam Optimizer Betas = 0.5, 0.999
- Lambda Cycle = 10
- Normalization = Instance Normalization

(a) Real BW

(b) Real Color

(a) Fake Color

(b) Fake BW

(a) Identity BW

(b) Identity Color

(a) Recreated BW

(b) Recreated Color

Figure 6.8: Images output during training

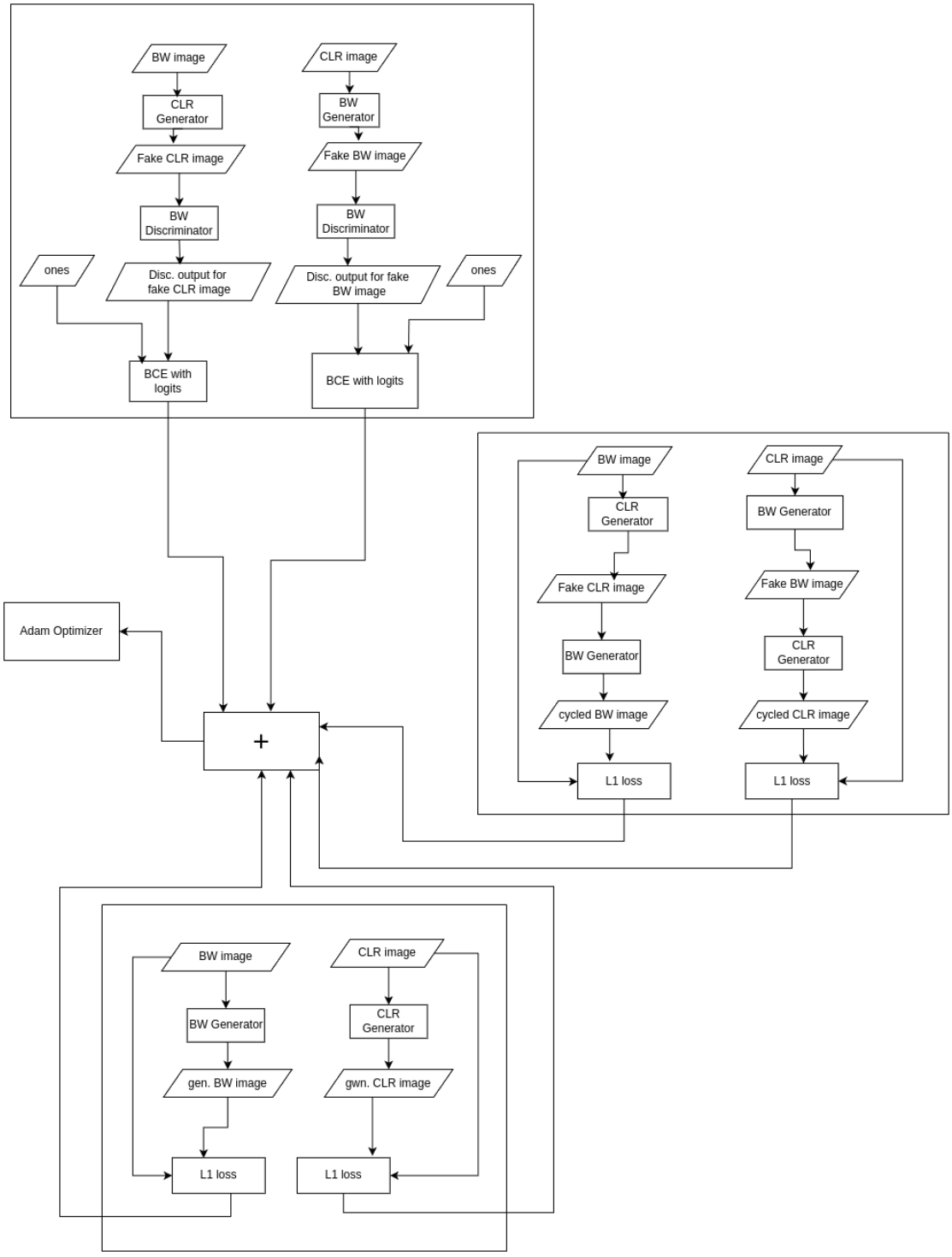


Figure 6.9: Generator Training Procedure

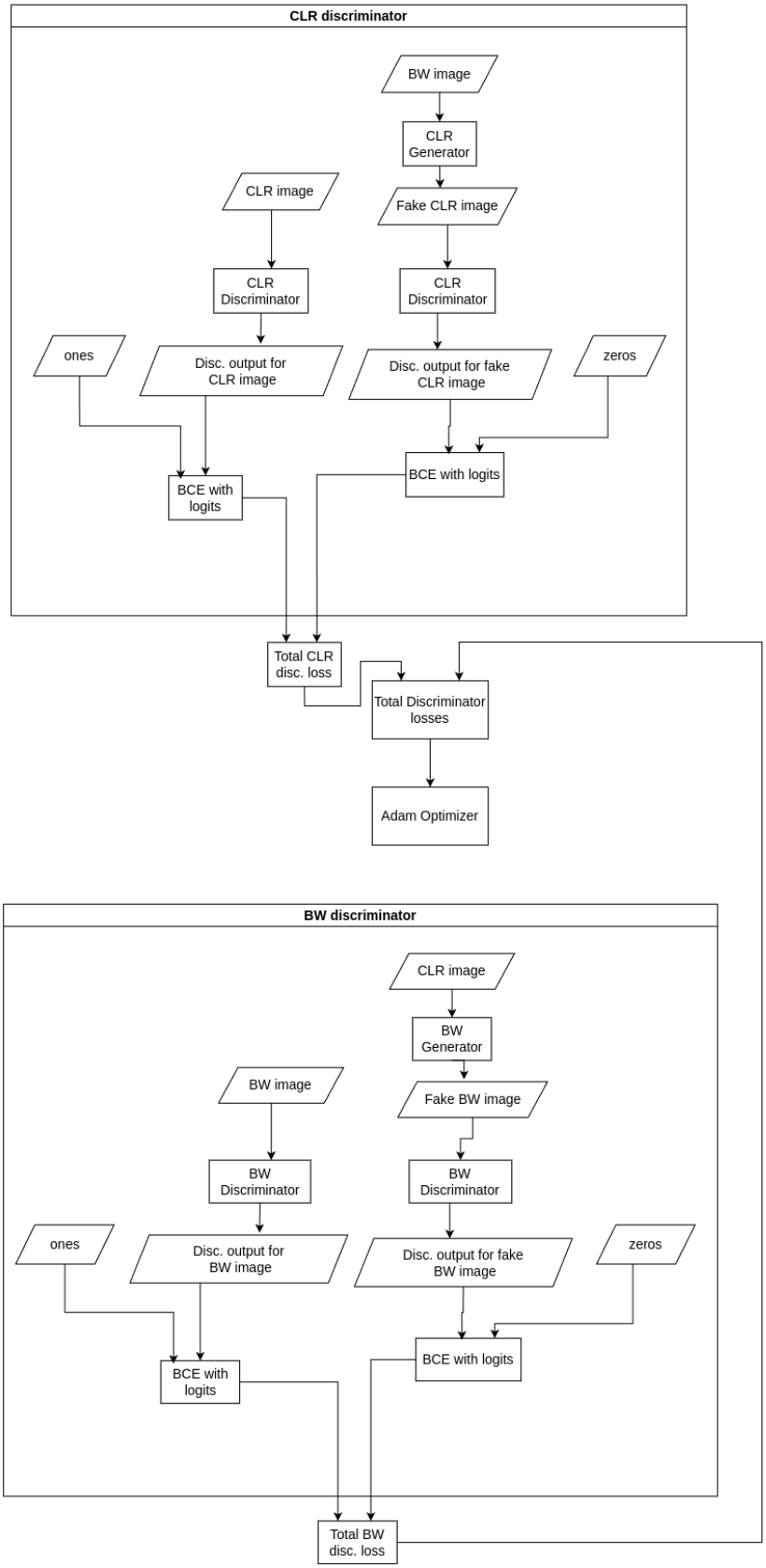


Figure 6.10: Discriminator Training Procedure

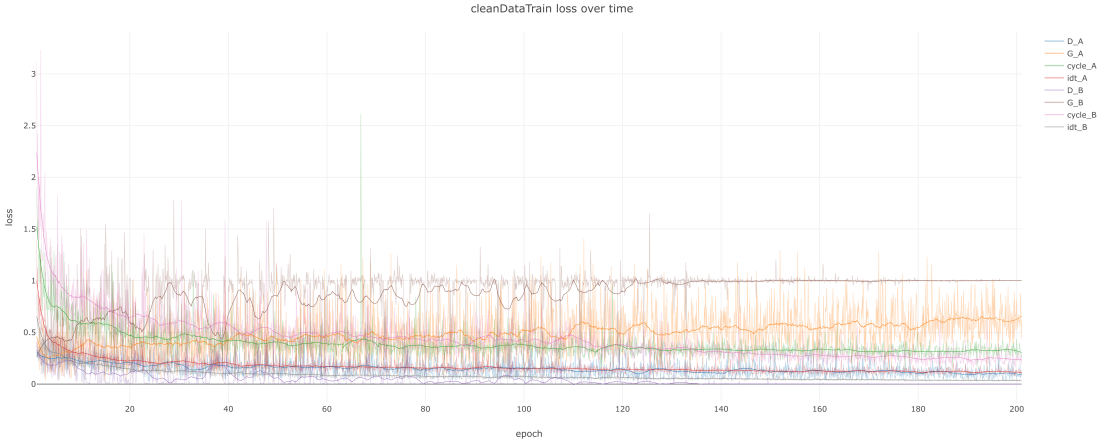


Figure 6.11: Losses for training with ResNet9Generator and PatchGAN

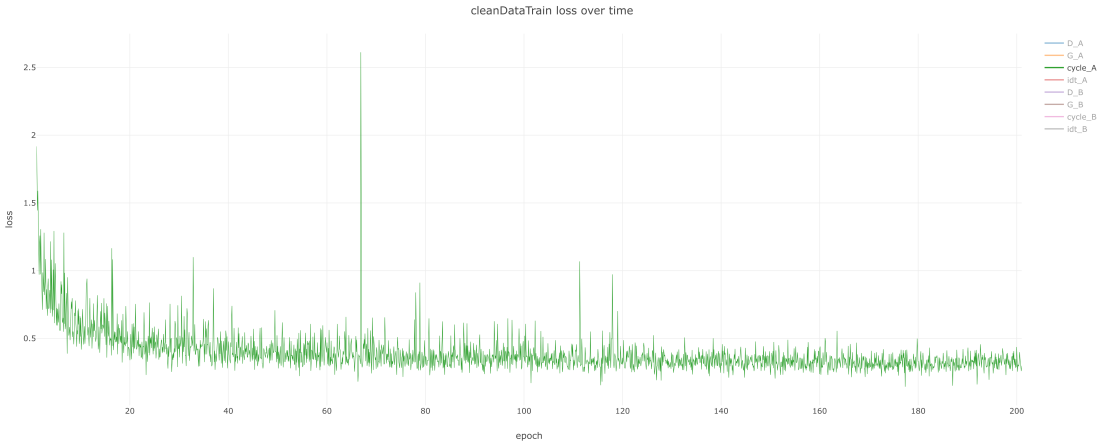


Figure 6.12: Cycle Consistency Loss (L1) for Color Generator

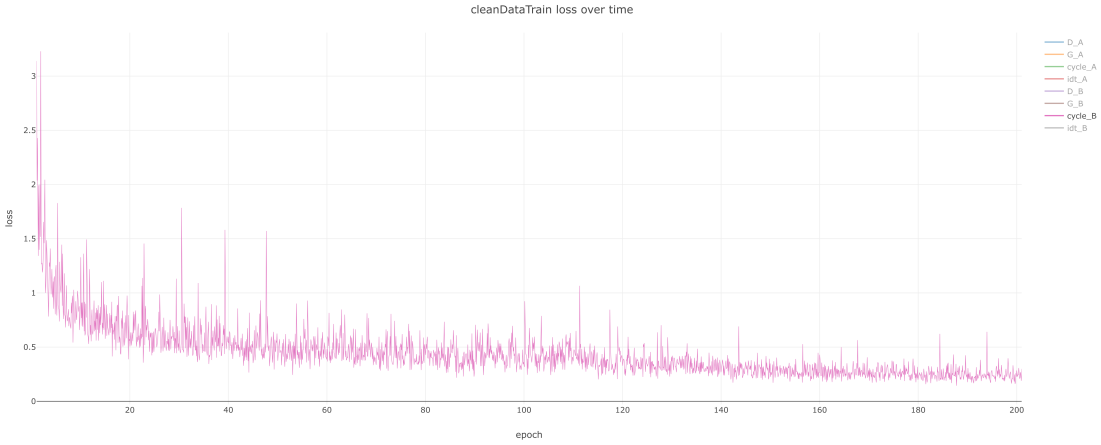


Figure 6.13: Cycle Consistency Loss (L1) for BW Generator

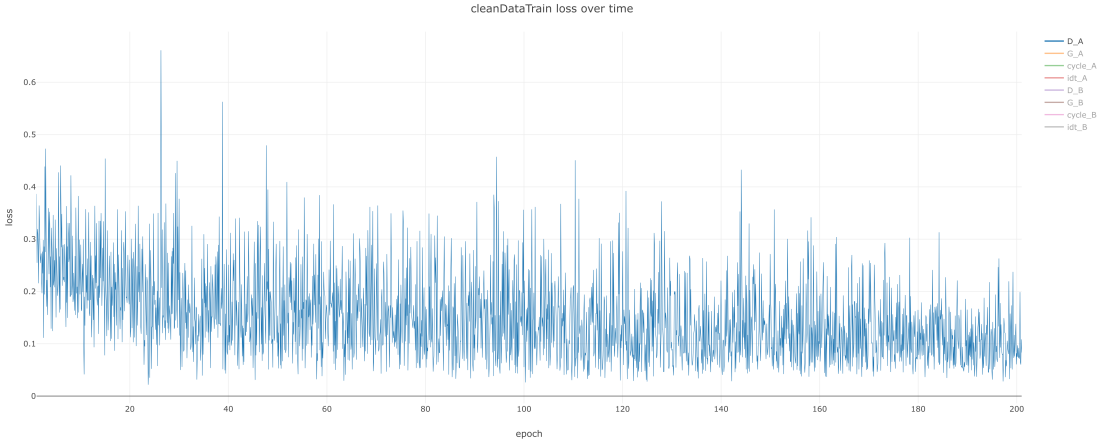


Figure 6.14: BCE Loss for Color Discriminator

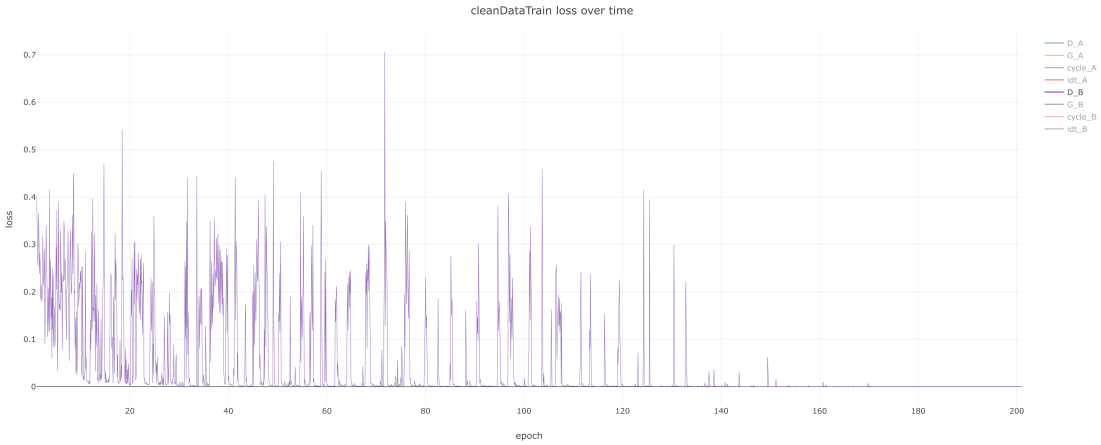


Figure 6.15: BCE Loss for BW Discriminator

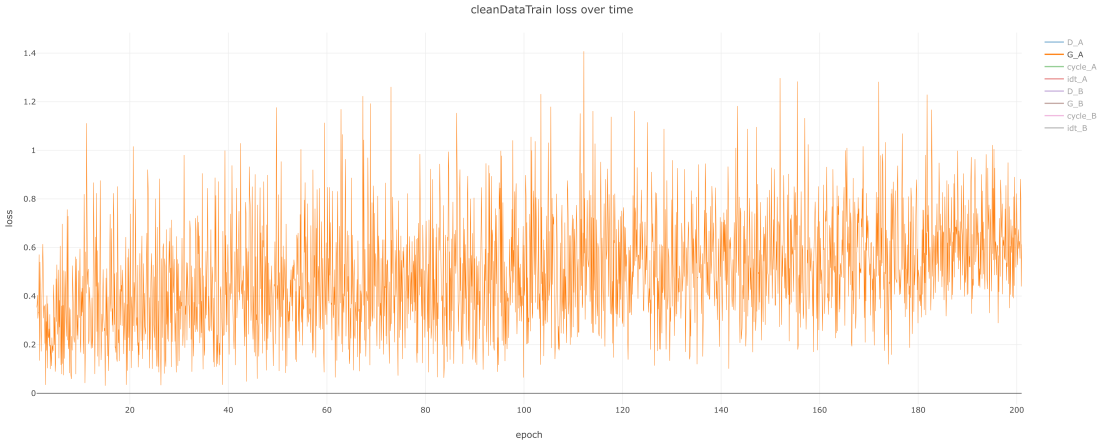


Figure 6.16: Adversarial (BCE) Loss for Color Generator

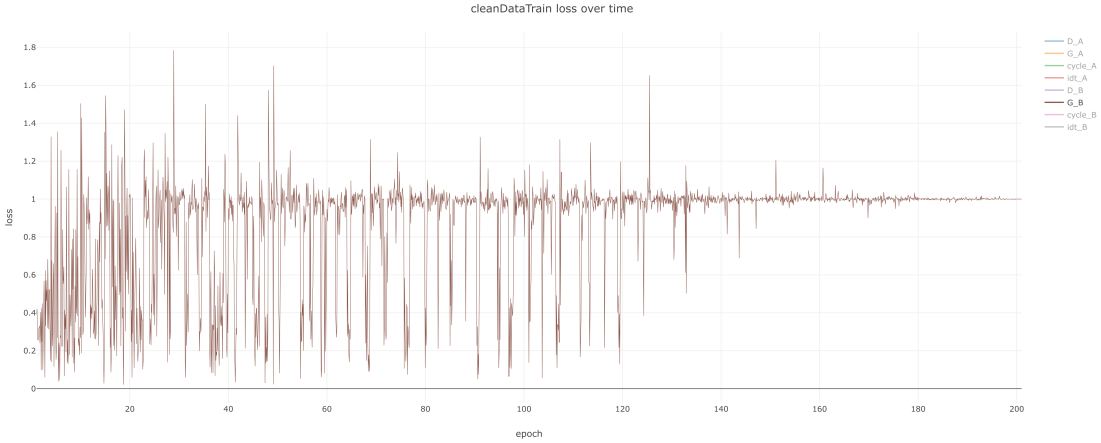


Figure 6.17: Adversarial (BCE) Loss for BW Generator

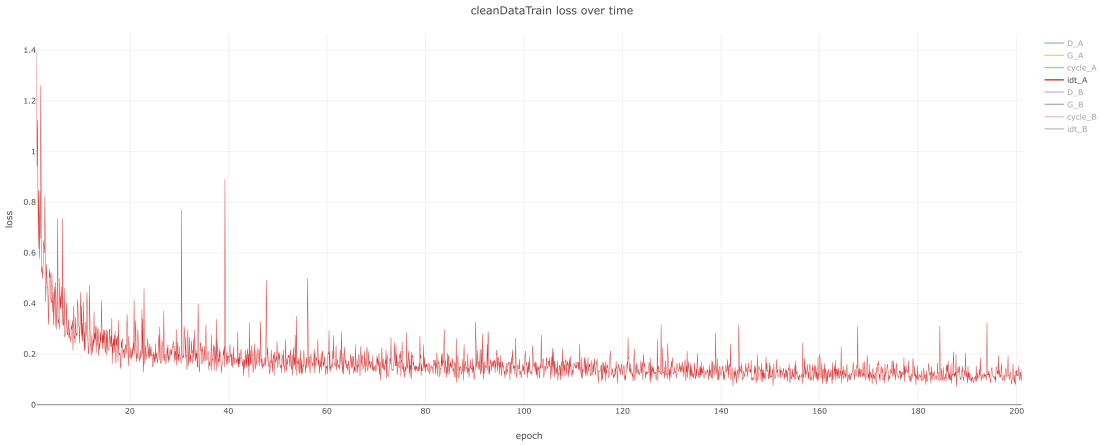


Figure 6.18: Identity (L1) Loss for Color Generator

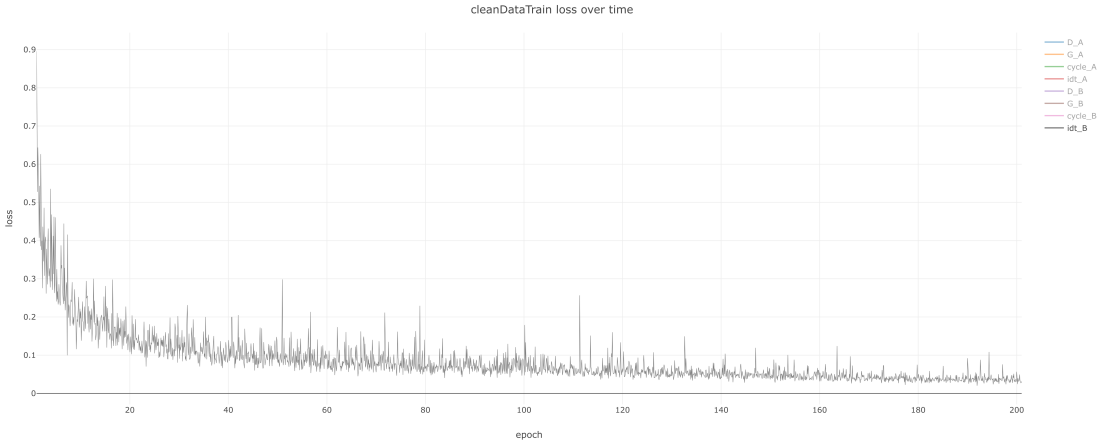


Figure 6.19: Identity (L2) Loss for Color Generator

where,

- D_A= BCE Loss for Color Discriminator,
- D_B= BCE Loss for BW Discriminator,
- cycle_A= Cycle Consistency (L1) Loss for Color Generator,
- cycle_B= Cycle Consistency (L1) Loss for BW Generator,
- G_A= Adversarial (BCE) Loss for Color Generator,
- G_B= Adversarial (BCE) Loss for BW Generator,
- idt_A= Identity (L1) Loss for Color Generator,
- idt_B= Identity (L1) Loss for BW Generator

6.3.3 Interfacing

6.3.3.1 Training Interface

We've implemented real-time visualization of the training process and results of the model using Visdom. This integration has enhanced our ability to monitor and analyze the evolution of the model's performance at every epoch. With it, we can dynamically visualize various metrics such as loss functions, image translations, and other relevant statistics, providing us with valuable insights into the model's behavior and enabling us to make informed decisions for optimization.

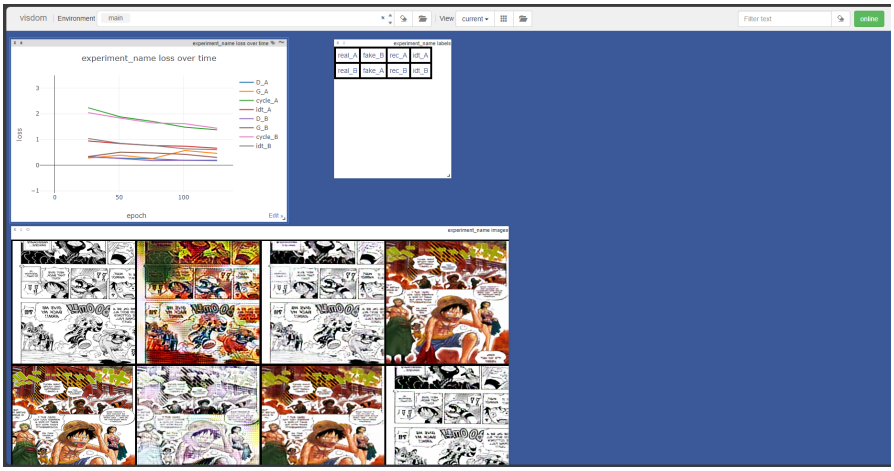


Figure 6.20: Web-based, real-time visualization of model training and outputs

6.3.3.2 User Interface

A simple UI (figure 6.21)has been developed using React and FastAPI for easier interaction with the model.

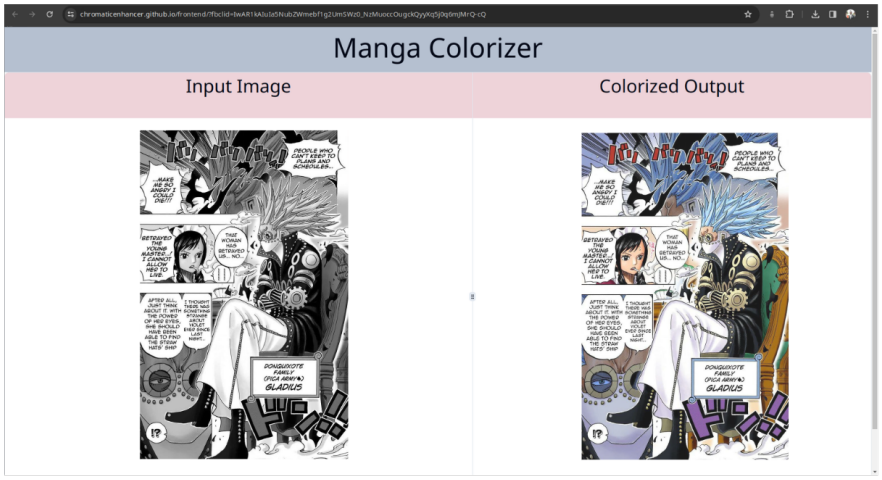


Figure 6.21: Demonstration of colorization via web interface

Chapter 7

Expected Outcome

The main aim of this proposed system is to streamline the process of manga colourization making it fully automatic, overcoming the shortcomings of previous colorization models and architectures such as conditional GANs [6], pix2pix [1] and style2paints [4] by reducing monotonicity and eradicating the need to provide manual color hints.

Ideally, our model should produce colored outputs that closely resemble those hand-painted by a human artist. However, acknowledging the complexities involved, we anticipate our model achieving results that are visually comparable to, or surpass the quality of existing semi-automatic and automatic models. It is important to note that the subjective nature of art allows for a spectrum of color interpretations; thus, while unconventional color choices may seem unrealistic, they are not inherently incorrect. For instance, a tree painted red can still evoke artistic merit within a specific style or context.

Given this consideration, it's crucial to recognize that the absence of contextual understanding within the text could lead to unintended outcomes. For instance, in a scenario where a manga text describes a "bright green bird," the model might erroneously color the bird white if it lacks contextual awareness and hasn't been exposed to a diverse range of bird illustrations in its training dataset.

(a) Black and white manga cover

(b) human-colored manga cover

Figure 7.1: Expected outcome for ideal automatic colorizer

Chapter 8

Actual Outcomes

8.1 GAN losses (200 epochs)

Discriminator_A	0.083
Generator_A	0.613
Cycle_A	0.298
Identity_A	0.141
Discriminator_B	0.000
Generator_B	1.000
Cycle_B	0.299
Identity_B	0.032

Table 8.1: Losses on training for 200 epochs
where,
A = Black and White Dataset
B= Colored Dataset

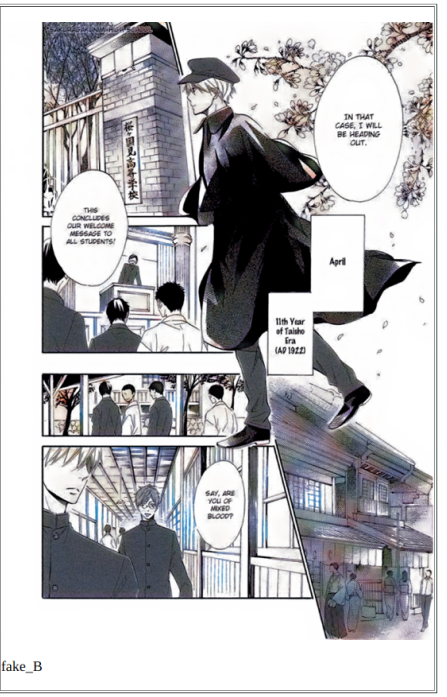
The losses for Generators and Discriminators reached the values as in 8.1 at the end of 200th epoch. The discriminator for colored image had its loss converged to 0 while the loss of the generator peaked to 1. The effect of adversarial nature of the networks was seen in the graph of losses. Color generator and color discriminator losses reached plateau at around 160 epochs.

8.2 Test Results (different sized inputs)

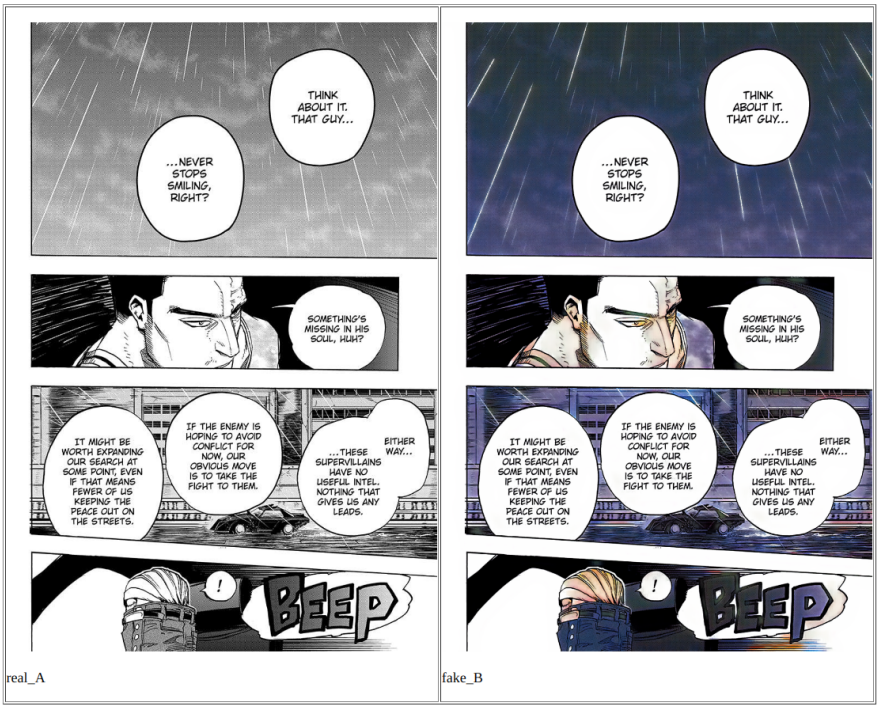
real_A

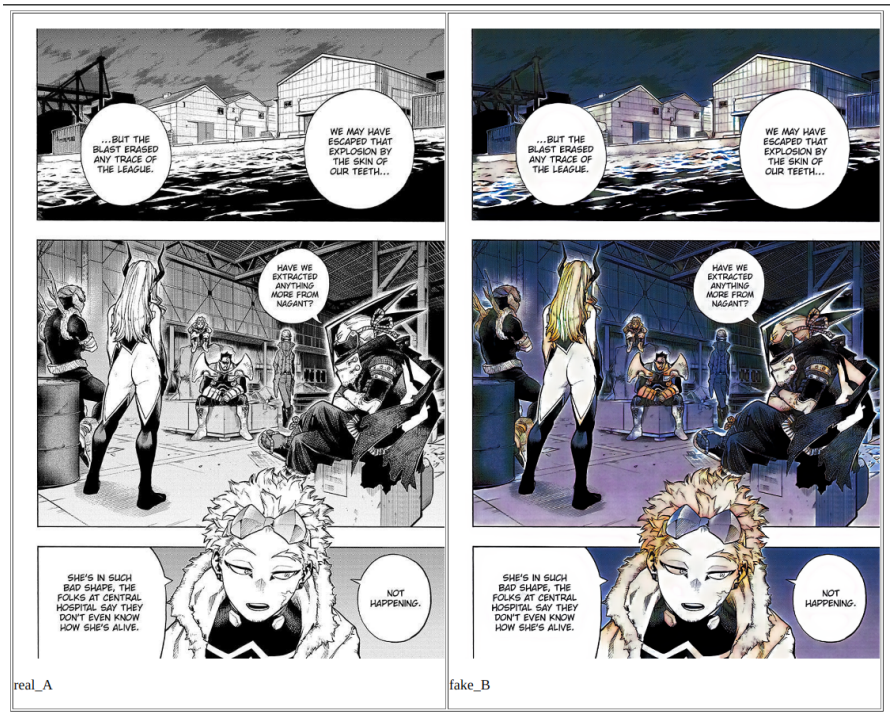
fake_B

real_A



fake_B





For testing and deployment, we only need to load the color generator. So far, it's been impressively consistent with colors across panels on a page, especially in darker and shaded spots where it fills in suitable colors that match the environment's mood. It made sure to steer clear of coloring over text blobs, and inherently white backgrounds.The details in reconstructed image remain intact and the test is well-readable. Despite being trained on cropped images from a small dataset, it's adapting well to a range of new test images. Our model can handle high-definition images of different sizes and colorize them effectively.

8.2.1 Limitations

While our model excels at coloring manga panels at certain scenarios such as dark room or rainy/cloudy weather, it fails to produce such good results at other scenarios. It also seems to have heavy inclination towards blue color.

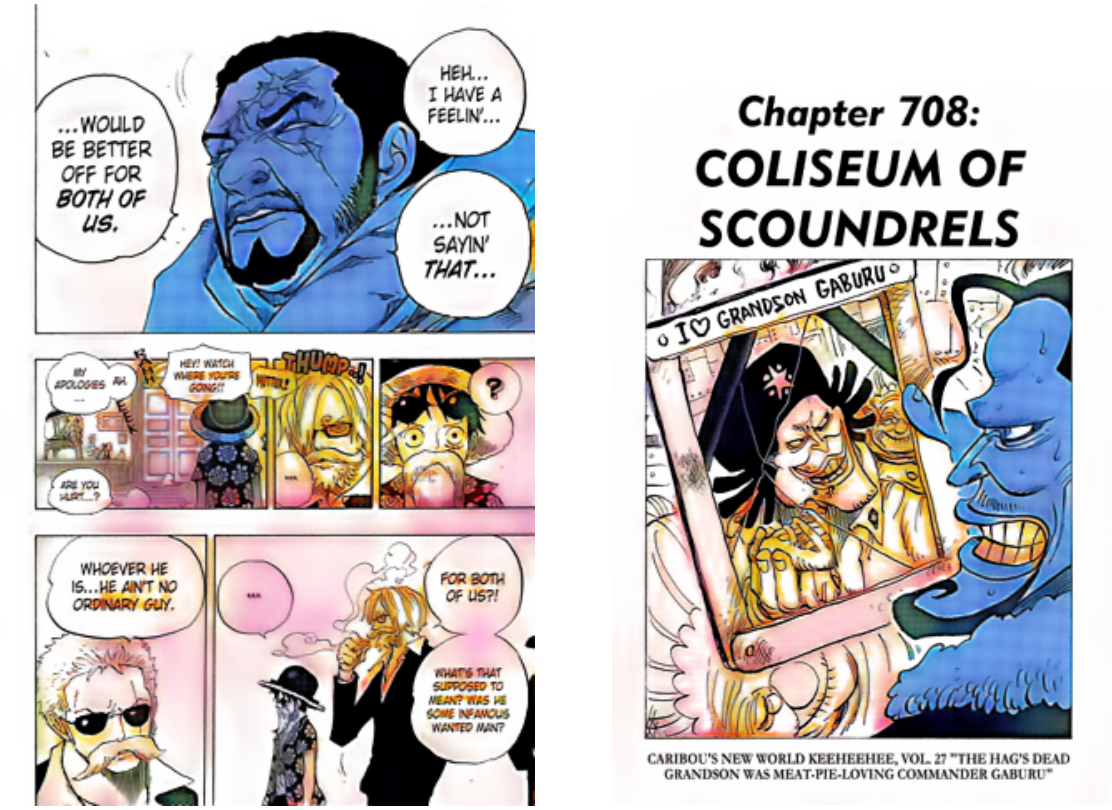


Figure 8.1: Failed colorized outputs

8.3 Performance Evaluation

In evaluating the fidelity of the reconstructed images compared to their original counterparts, we utilized Peak Signal-to-Noise Ratio (PSNR) as a metric. This measure allows for a quantitative evaluation of reconstruction quality, with higher PSNR values indicating higher fidelity. Our assessment involved four different images, each representative of distinct manga panel styles. Across all image pairs, the PSNR values were consistently high, averaging at 24.3742 dB. This average PSNR suggests satisfactory performance of the color generator in reconstructing the images.

To evaluate structural similarity between the original and reconstructed images, we employed the Structural Similarity Index (SSIM). SSIM considers various aspects such as luminance, contrast, and structure, providing a comprehensive assessment. SSIM values range from -1 to 1, with 1 indicating perfect similarity. While slight visual differences were observed between the image pairs, the average SSIM value of 0.9039 confirms that the reconstructed images maintain structural coherence.

In summary, our CycleGAN model has demonstrated reasonable performance based on both PSNR and SSIM scores, indicating successful image reconstruction. Nonetheless, further optimization through fine-tuning or experimentation, along with training on a larger dataset, may yield even more desirable results.

Figure 8.2: Real(top row) and Reconstructed colored (bottom row) images

Image Pair Labels	PSNR	SSIM
a	24.5019	0.9404
b	24.2575	0.8277
c	24.8711	0.9297
d	23.8663	0.9180
Average	24.3742	0.9039

Table 8.2: PSNR and SSIM table for real and reconstructed real images

Chapter 9

Conclusion and Future Enhancements

Conclusively, for the project Automatic Colorization of Japanese Comics using GANs is completed as it was successfully deployed as a python package and a website.
As per future enhancements following can be considered:

- Use of contrastive-unpaired-translation to reduce the time and resources required for training [19]
- Better hardware can be used to train image in higher resolution to get more quality results
- Better and larger datasets can be used to accomodate colorization of variety of scenarios

Bibliography

[1] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," 2018.

[2] Y. Qu, T.-T. Wong, and P.-A. Heng, "Manga colorization," *ACM Transactions on Graphics (ToG)*, vol. 25, no. 3, pp. 1214–1220, 2006.

[3] P. Hensman and K. Aizawa, "cgan-based manga colorization using a single training image," 2017.

[4] Y. J. LvMin Zhang and C. Liu, "Style transfer for anime sketches with enhanced residual u-net and auxiliary classifier gan," 2017.

[5] T. Jiramahapokee, "inkn'hue: Enhancing manga colorization from multiple priors with alignment multi-encoder vae," *arXiv preprint arXiv:2311.01804*, 2023.

[6] M. Golyadkin and I. Makarov, "Semi-automatic manga colorization using conditional adversarial networks," in *Analysis of Images, Social Networks and Texts*, W. M. P. van der Aalst, V. Batagelj, D. I. Ignatov, M. Khachay, O. Koltsova, A. Kutuzov, S. O. Kuznetsov, I. A. Lomazova, N. Loukachevitch, A. Napoli, A. Panchenko, P. M. Pardalos, M. Pelillo, A. V. Savchenko, and E. Tutubalina, Eds. Cham: Springer International Publishing, 2021, pp. 230–242.

[7] C. Li, X. Liu, and T.-T. Wong, "Deep extraction of manga structural lines," *ACM Transactions on Graphics (SIGGRAPH 2017 issue)*, vol. 36, no. 4, pp. 117:1–117:12, July 2017.

[8] Y. Shimizu, R. Furuta, D. Ouyang, Y. Taniguchi, R. Hinami, and S. Ishiwatari, "Painting style-aware manga colorization based on generative adversarial networks," in *2021 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2021, pp. 1739–1743.

[9] C. Furusawa, K. Hiroshiba, K. Ogaki, and Y. Odagiri, "Comicolorization: Semi-automatic manga colorization," 2017.

[10] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," 2020.

[11] D. I. Varga and T. Szirányi, "Convolutional neural networks for automatic image colorization," 2017.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[13] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[14] K. Kim, S. Park, E. Jeon, T. Kim, and D. Kim, "A style-aware discriminator for controllable image translation," 2022.

[15] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," 2017.

[16] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[18] S. Huang, X. Jin, Q. Jiang, J. Li, S.-J. Lee, P. Wang, and S. Yao, "A fully-automatic image colorization scheme using improved cyclegan with skip connections," *Multimedia Tools and Applications*, vol. 80, no. 17, pp. 26 465–26 492, 2021.

[19] T. Park, A. A. Efros, R. Zhang, and J.-Y. Zhu, "Contrastive learning for unpaired image-to-image translation," 2020.