Final Project
Image Colorization Using GANs

#	Names:	ID	Section
1	Abdulaziz Ebrahim	1946282	B53
2	Ahmed Al-Khayal	1945541	B53
3	Turki Safar Al-Zahrani	1935944	B53

Instructor:
Dr. Mohammed J. Abdulaal

Table of Contents

Table of Figures

Introduction

Image colorization is the process of adding color to a black and white image. In the past, this was typically done by hand, using a set of specialized pens and inks. This was a difficult and time-consuming task that required human intervention and continuous monitoring by experts and historians to guide the artist thought the process. However, thanks to advances in machine learning, we now have algorithms that can color images automatically. These coloring algorithms are based on a type of AI called a generative adversarial network, or GAN.

GANs are made up of two parts: a generator and a discriminator. The generator creates fake images, while the discriminator tries to distinguish between real and fake images. The two parts compete with each other, and as they do so, they both get better at their respective tasks. This process is similar to the way humans learn; we also constantly generate hypotheses and test them against reality.

In this project we utilized the power of GANs to generate colored images from their grayscale. We developed an efficient process to train a fairly small model in short period of time on a tiny dataset. And the results are incredible!

Problem at Hand

As Described, we are trying to color images using a fast, accurate, and automated method. However, till this day, old methods are being used to color historical media. Such methods have many issues: one of them is how costly and time consuming it is to hire an artist to color even a single old family photo. For example, buying a colored historical image by an artist named Jordan J. Lloyd could cost you up to 150$ [1] as for coloring personal photos the rate is even higher. Another huge issue is that artist could manipulate the historical significance of images based on their views and beliefs [2]. Additionally, artists could alter images in a way to represent people in different manner [3].

Current Solutions

Currently historical images and videos are being colored using old slow ways; such as coloring each image or video frame by **hand** which requires a lot of resources and most importantly knowledge about the process itself as well as the photo setting [4]. Another method is **mapping** the brightness of each pixel in the image to a specific color done using a preset palette of colors [5]. There is also a method that us **image matching**; where a colored image is selected as a reference for the color palette and the pixels in the gray image are matched with the source pixel this technique was proposed by E. Reinhard [6]. There is also a method called **colorization by seed**, where the user marks the gray image by colored dots or seeds and an application is responsible for spreading the color of those seeds in the bounded region; Anat Levin system is an example of this type [7].

Our Approach

Our method involves using fully automated systems that are fast, scalable, and require no human intervention that could affect the historical significance of colored images. Our approach include training a scalable GAN network on realistic images and use the generator part of the network as an inference model where the user can provide grayscale image and our generator can instantly color his image without altering any shapes or objects in the photo. This is insured by using L*a*b format where the grayscale image L is kept untouched and the coloring layers A and B are added on to change only the color intensities of the image. This process is described in detail in the data preprocessing section. Our system uses techniques based on the state-of-the-art general solution of image-to-image tasks in deep learning proposed by Berkeley AI Research [8].

Methodology

Overall System

Our system uses techniques based on the state-of-the-art general solution of image-to-image tasks in deep learning proposed by Berkeley AI Research [8]. The first step to our system is acquiring the right data set for the coloring task; the number of images required can be 8000 images at minimum. Then comes the step of preprocessing the data and format conversion. After that we train our model on the training part of the dataset. Finally, we use the generator part of the model for inference.

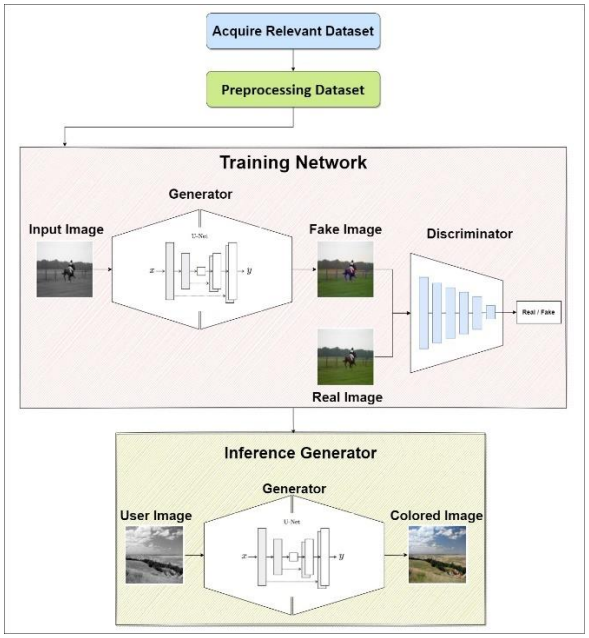

Figure 1: System Flowchart

Model Architecture

GANs are made up of two parts: a generator and a discriminator. Both parts compete against each other. In this section we will describe our implementation of each part. Starting with the discriminator, in our implementation it is composed of 5 convolutional layers that produce a decision whether the image is real or fake.

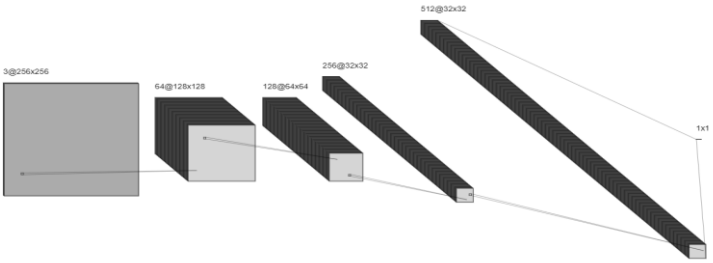

Figure 2: Discriminator architecture

As for the GAN model, we build our network according to the U-NET architecture. Where the network consists of two parts: encoder that down samples the image and extract features. And a decoder that regenerate the image by up sampling passed encoder features. However, what is different about this architecture compared to encoder-decoder implementation is the skip connections, where up sampling layers has direct connections with the encode feature maps which boosts the model accuracy.

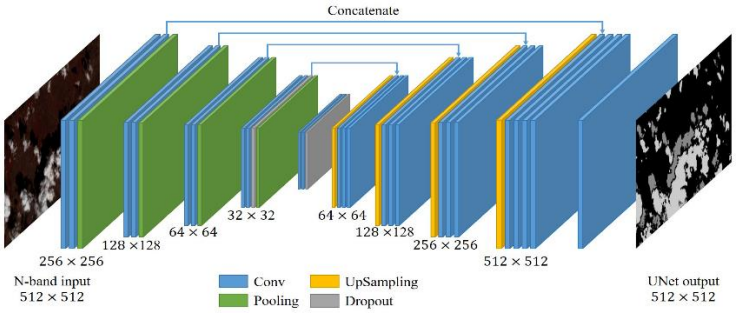

Figure 3: U-NET Generator architecture

Putting it all together. Connecting our generator output to the discriminator input as well as feeding real images to the discriminator:

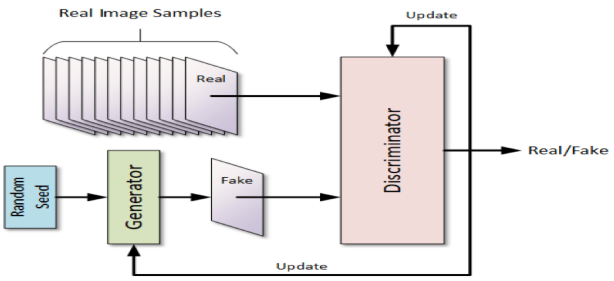

Figure 4: Model architecture

Training Datasets

We trained our model on Three different datasets to understand its limits. Each with a different purpose.

- **Places365**: determine how the model performs on real world scenery
- **Annotated Anime Faces:** understand how the model performs on drawn character faces
- **Black Colver Manga**: determine how the model deals with both drawn characters and scenery

The first dataset is called **Places365**. Which is a designed following the principles of human visual cognition. It consists of over 10 million images comprising of 400+ unique scenes such as a park, airfield, living room, etc.… Each class features from 5 to 30 thousand training images [9]. We used only 8300 images to train our model (7000 training/ 1300 testing).

Figure 5: Places365 dataset sample

The Second dataset is called **Annotated Anime Faces Dataset** which contains around 6600 of labeled colored drawn anime characters collected from Pixiv (Japanese online community for artists) [10]. We used the entire dataset using 5300 images for training and 1300 for testing.

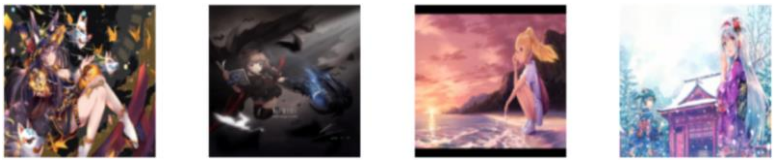

Figure 6: Annotated Anime Faces dataset sample

The third dataset is called **Black Clover Manga Dataset** which is fairly small containing around 1000 colored manga pages of the popular anime black clover. This dataset was collected by a Kaggle user (Chandler Timm Doloriel) [11]. We used the entire dataset using 900 images for training and 100 for testing.

Figure 7: Black Clover Manga dataset sample

Data Preprocessing

Our preprocessing method involved various techniques, from resizing and rescaling the training data to format conversion. The following are the preprocessing methods we used:

- **RGB to L*a*b format conversion**

The most important preprocessing method in our training process was converting training images from RGB format to L*a*b. To understand why is that we first look at how images are represented in RGB format. When loading images, we get a three-dimensional array of (height, width, color) with the last axis representing the color of the image. This color property indicate how much Red, Green, and Blue is in the image.

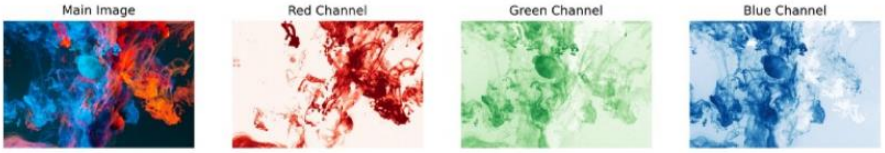

Figure 8: Image in RGB color space

Now in L*a*b color space, we also have three numbers representing each pixel but these numbers have a totally different meaning. The first channel **L** encodes the **Lightness** of the image which when visualized it represent the image in black and white. The second and third channel ***a** and ***b** encode how much green-red and yellow-blue in the image respectively.

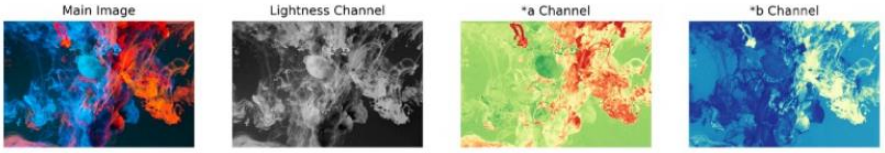

*Figure 9: Image in L*a*b color space*

If it's not clear by now why are we using L*a*b format, well it's mainly for two reasons. The first being reducing the amount of data preprocessing we need; our model takes in grayscale images and uses colored images to determine the loss. We need to convert each image to both grayscale and RGB. Where compared to L*a*b where we can extract the grayscale image by using the first channel. Secondly, and this is a big reason, since our model is a generative one; working with RGB images requires our model to predict three channels. However, using L*a*b format our model need only to predict two color channels which reduces the training time and increases the accuracy of the model.

The following represent the mathematical way of converting an image to L*a*b format from RGB:

First, we need to convert an image from RGB to XYZ this is done by the following matrix multiplication:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = [M] \cdot \begin{bmatrix} r \\ g \\ b \end{bmatrix} \quad (1)$$

Where the compounded RGB channels denoted with upper case (R,G,B), or generically V are made linear with respect to energy (denoted with lower case (r,g,b), or generically v:

$$v = \begin{cases} V/12.92 & \text{if } V \leq 0.04045 \\ ((V + 0.055)/1.055)^{2.4} & \text{otherwise} \end{cases} \quad (2)$$

And the transformation matrix [M] is calculated from the RGB reference primaries:

$$[M] = \begin{bmatrix} SrXr & SgXg & SbXb \\ SrYr & SgYg & SbYb \\ SrZr & SgZg & SbSb \end{bmatrix} \quad (3)$$

Where, given the chromaticity coordinates of an RGB system (xr, yr), (xg, yg) and (xb, yb) and its reference white (Xw, Yw, Zw): (4)

○ Xr = xr/yr ○ Yr = 1 ○ Zr = (1-xr-yr)/yr	○ Xg = xg/yg ○ Yg = 1 ○ Zg = (1-xg-yg)/yg	○ Xb = xb/yb ○ Yb = 1 ○ Zb = (1-xb-yb)/yb

$$\begin{bmatrix} S_r \\ S_g \\ S_b \end{bmatrix} = \begin{bmatrix} X_r & X_g & X_b \\ Y_r & Y_g & Y_b \\ Z_r & Z_g & Z_b \end{bmatrix}^{-1} \begin{bmatrix} X_W \\ Y_W \\ Z_W \end{bmatrix} \quad (5)$$

Now that we have the XYZ matrix, we can obtain the L*a*b channels via:

- $L = 116\, f_y - 16$ (6)
- $a = 500(f_x - f_y)$ (7)
- $b = 200(f_y - f_z)$ (8)

Where

$$f_x = \begin{cases} \sqrt[3]{x_r} & \text{if } x_r > \epsilon \\ \frac{\kappa x_r + 16}{116} & \text{otherwise} \end{cases} \quad (9)$$

$$f_y = \begin{cases} \sqrt[3]{y_r} & \text{if } y_r > \epsilon \\ \frac{\kappa y_r + 16}{116} & \text{otherwise} \end{cases} \quad (10)$$

$$f_z = \begin{cases} \sqrt[3]{z_r} & \text{if } z_r > \epsilon \\ \frac{\kappa z_r + 16}{116} & \text{otherwise} \end{cases} \quad (11)$$

$$x_r = \frac{X}{X_r} \qquad y_r = \frac{Y}{Y_r} \qquad z_r = \frac{Z}{Z_r} \quad (12)$$

$$\epsilon = \begin{cases} 0.008856 \\ 216/24389 \end{cases} \qquad \kappa = \begin{cases} 903.3 \\ 24389/27 \end{cases} \quad (13)$$

- **Image Resizing**

Here we used Pytorch transformer function to resize our images to be (256x256) In width and height. Where we used Bicubic interpolation where it interpolates pixel color values, introducing a continuous transition into the output result even where the original data has discrete transitions.

- **Image Rescaling**

We rescaled our L*a*b image to be in range of [-1, 1] by:

- $L = (L / 50) - 1$ (14)
- $ab = ab/110$ (15)

- **Image Augmentation**

We augmented our image by using random horizontal flips around y-axis. This was done to pseudo increase the amount of data we have.

- **Image Batching**

We divided and grouped the images into batches, where each batch contained 32 images. This very recommended especially when you have a lot of data in order to increase the speed and efficiency of the training process.

Feature Extraction

Luckly for us neural network models extract features by their own without our help. Both our generator and discriminator have 2D convolutional layers that learn to map and extract image features. Convolutional layers extract features by convolving a learnable weight matrix (kernel) by the input image and the result is a feature map.

$$G[m,n] = (f * h)[m,n] = \sum_j \sum_k h[j,k] f[m-j, n-k] \quad (16)$$

Where f represents the input image and h represent the kernel. And the indexes of rows and columns are of the result feature matrix are marked with m and n respectively.

Model Optimization and Classification

Since our model composed of different networks, we have multiple loss functions that we are going to combine. The generator takes a grayscale image (L channel) and produces 2-channel image (ab). On the other hand, the discriminator takes the two generated channels concatenates them with the original grayscale and classify the image wither its real or fake. Of course, the discriminator also sees real images.

Starting off with the **discriminator** we used **Binary Cross Entropy Loss** as our loss function since we only have two out comes (Real/Fake):

$$\text{Loss} = -\frac{1}{\text{output size}} \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i) \quad (17)$$

Moving on to the **generator** Loss. Consider ***x*** as the input grayscale image, ***z*** as the input noise for the generator, and ***y*** is the output two color channel from the generator. Additionally, **G** is the generator, and **D** is the discriminator. Then the loss for GAN is:

$$L_{cGAN}(G, D) = E_{x,y}[log\ D(x, y)] + E_{x,y}[log\ (1 - D(x, G(x, z))] \quad (18)$$

The above loss function help produce good colorful photos, but to help further the model we introduced some supervision by combining the above loss function with **L1 Loss** of the predicted colors with the actual colors.

$$L_{L1}(G) = E_{x,y,z}[||y - G(x, z)||] \quad (19)$$

Finally, our combined loss function will be:

$$G^* = arg \min_G \max_D L_{cGAN}(G, D) + \lambda L_{L1}(G) \quad (20)$$

Where λ is used to control the contribution of the two losses to the final loss.

Results

After training the model on the three different datasets mentioned before for 100 epochs each, we ended up with the following results:

- Places365

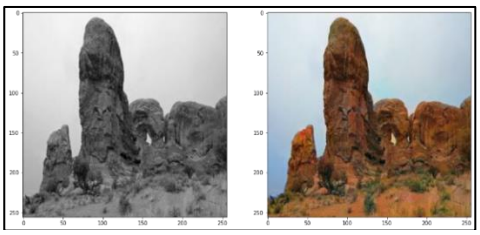

Figure 10: Model Result 1

- Annotated Anime Faces

Figure 11: Model Result 2

- Black Colver Manga

Figure 12: Model Result 3

In generative networks accuracy isn't a preferred measurement and most often isn't used at all because it is misleading. To measure the accuracy in generative models we take the absolute difference between generated images and original images. However, most often the accuracy value is very low no matter how good the model performs. This is because the pixel value of the generated image is always different from the original. But when the image is looked at by humans, they hardly can differentiate. So, the correct measurement for accuracy would be to ask a person to distinguish between real and generated images. Which we did, we presented two of our friends with 10 images (1 real, 9 fake). The **first** had an accuracy of **50%** which is equivalent to guessing randomly. The **second** had an accuracy of **40%** which is worse than guessing randomly. And to validate our model we split the data mostly into 80% training and 20% validation.

Conclusion

The system we developed consists of four stages: the first is data acquisition in the preferred domain whether it is real life images or drawings. The second is applying discussed preprocessing methods to presented data. The third stage is training the GAN model. Forth and finally, is detaching the generator model for inference tasks. Concluding from the results we obtained, our system looked very reliable even on the small scale we trained on. And detaching the generator model provided us with a small and fast network that could potentially be used in real-time application like real time coloring of camera feed from space or in mines and caves. However, the model has a lot of areas where it can be improved: first, training the model on more data provides it with more information and feature maps that could boost its performance tremendously. Additionally, and this a great step to take, is to use transfer learning in both the generator and discriminator networks. Using a pretrained model like Resnet as the Encoder part of the generator, and as the discriminator itself, can decrease the training time a lot and boost the “accuracy” of the model tremendously.

References

[1] colorgraph, "Prints by Jordan J. Lloyd," [Online]. Available: https://colorgraph.co/collections/jordanjlloyd. [Accessed 18 11 2022].

[2] ladyscience, "The Problem With Colorizing Historic Photographs," [Online]. Available: https://www.ladyscience.com/commentary/the-problem-with-colorizing-historic-photographs-2021. [Accessed 2022].

[3] Khmer Times , "Outrage and horror as website publishes altered “colourised and smiling” images of Khmer Rouge victims from S-21 prison," Khmer Times , [Online]. Available: https://www.khmertimeskh.com/50837136/outrage-and-horror-as-website-publishes-altered-colourised-and-smiling-images-of-khmer-rouge-victims-from-s-21-prison/. [Accessed 18 11 2022].

[4] InstaRestoration, "History of photo colorization," [Online]. Available: https://www.instarestoration.com/blog/history-of-photo-colorization. [Accessed 18 11 2022].

[5] D. Frias, "Equalization and Brightness Mapping Modes of Color-to-Gray," 2022.

[6] E. A. K. A. O. A. G. J. Erik Reinhard, "Color Transfer between Images," in *Color Imaging*, New York, 2022, pp. 467-474.

[7] D. L. Y. W. Anat Levin, "Colorization using Optimization," 2004.

[8] J.-Y. Z. T. Z. A. A. E. Phillip Isola, "Image-to-Image Translation with Conditional Adversarial Networks," 2022.

[9] A. L. A. K. A. O. a. A. T. Bolei Zhou, "Places: A 10 million Image Database for," 2017.

[10] K. u. ANDY8744, "Annotated Anime Faces Dataset," [Online]. Available: https://www.kaggle.com/datasets/andy8744/annotated-anime-faces-dataset?select=train. [Accessed 18 11 2022].

[11] C. T. DOLORIEL, "Black Clover Manga Dataset," [Online]. Available: https://www.kaggle.com/datasets/chandlertimm/black-clover-manga-dataset?select=colored. [Accessed 18 11 2022].

[12] D. Wilimitis, "The Kernel Trick in Support Vector Classification," towardsdatascience, [Online]. Available: https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f.

[13] V. Explained, "Support Vector Machine (SVM) in 2 minutes," youtube, [Online]. Available: https://www.youtube.com/watch?v=_YPScrckx28.

[14] S. Narkhede, "Understanding AUC - ROC Curve," towardsdatascience, [Online]. Available: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

[15] S. w. J. Starmer, "ROC and AUC, Clearly Explained!," youtube, [Online]. Available: https://www.youtube.com/watch?v=4jRBRDbJemM.

[16] "Neural Networks IBM Cloud Education," ibm, 17 August 2020. [Online]. Available: https://www.ibm.com/cloud/learn/neural-networks. [Accessed 26 October 2022].

[17] simplelearn, "Neural Network In 5 Minutes | What Is A Neural Network? | How Neural Networks Work | Simplilearn," Youtube, [Online]. Available: https://www.youtube.com/watch?v=bfmFfD2RIcg.