# Colorize B&W Manga

Bing-Shu Wu, Pin-Shao Chen
National Yang Ming Chiao Tung University
No. 1001, Daxue Rd. East Dist., Hsinchu City 300093, Taiwan
shu0924.cs09@nycu.edu.tw, cps@cs.nctu.edu.tw

## Abstract

*The purpose of this project is to colorize B&W manga. We chose two ways, CNN and U-net, to be the models to train the agent. Refer to these two websites and this paper [1–3] to implement the CNN version and the U-net version. Finally, we improve the U-net version by putting the classifier in, which is referred by the website [1].*

## 1. Introduction

Coloring gray-scale images can have a big impact in many domains, for instance, understand historical images from a new perspective. This project is trying to color B&W manga. We love watching anime and manga and want to see B&W manga which have not be animated yet to be colorized. Hence by this opportunity, we try to make a tool that can colorize manga so that the manga matches the color of the anime.

Currently we know of the following ways to be models for colorize gray-scale images: CNN, U-net, GAN, Mask R-CNN. Finally chose CNN as the baseline and U-net as the main approach to colorize manga. Since these two methods are rarely mentioned in the course, it takes some time to learn and understand. We refer to this website [1] to implement the CNN version. Then refer to this paper [3] and this website [2] to understand and implement the U-net version. Finally, we have five different models to color manga. Then we compare the resulting photos with the photos that have been colored by existing programs in the Internet to see which one is the closest to the anime.

## 2. Related Work

We used the information of this website [1], from the alpha version to the beta version, and gradually made the author's Full version. His beta version is a Sequential Model composed of Convolution 2D, while the full version is based on the beta version and add the Inception-ResNet-v2 model as a Classifier in the middle layer. In the end, we only retain the concept of Classifier, and choose the VGG16 model that is more suitable for our dataset. For the part other than Classifier, we implement a simplified version of the standard U-net structure.

This website [3] explains the U-net in detail and provides the relevant code, we reduced the number of filters to avoid hardware requirements of standard U-net exceeding the limit of our GPU. After combining the above two references, we also changed many parameters, such as learning rate, optimizer, activation function, etc., to optimize the coloring results.

## 3. Methodology

In order for the machine to learn how to color manga, we have to give it some examples of coloring. Then we use anime as the training dataset to train the models.

First, collect the data of anime. We write a simple program to screenshot once every two seconds. Play the anime and run the program until we get $600$ photos. Then randomly select $240$ images from it as the training dataset and $16$ images as the testing dataset. Then there is the data processing part. We directly use $skimage$ to resize the image to $256 \times 256$ to let the training process becomes faster. This is also to make the adjusted image size closer to the input image size $224 \times 224$ required by the VGG16 Classifier as our tool. And we also convert the color space of the picture from $RGB$ to $Lab$, $L$ is the corresponding graysacle value, so all the model has to do now is to predict the value of channel $a$ and channel $b$ for a given set of $L$ channels of size $256 \times 256$.

For the Evaluation Matrix, since there is no right or wrong in color, It is difficult to design a function to absolutely define the accuracy of an algorithm. Although this part can be measured by GAN, it is beyond the scope of our implementation. Therefore, we use the built-in CategoricalAccuracy() of $keras$ as our Evaluation Matric. After testing, this function does almost correctly reflect the status of a model. The principle of CategoricalAccuracy() is as Figure 1.

```
def categorical_accuracy(y_true, y_pred):
    return K.cast(K.equal(K.argmax(y_true, axis=-1),
                          K.argmax(y_pred, axis=-1)),
                  K.floatx())
```

Figure 1. Categorical Accuracy

We can see that CategoricalAccuracy() will check for the innermost axis to check whether the maximum value of the prediction result (y_pred) occurs at the same index as the the real image (y_true). We know that the matrix shape of the prediction result is $256 \times 256 \times 2$, where $256 \times 256$ is the image size, so the innermost axis is the value of $a$ and $b$ of $Lab$. Hence CategoricalAccuracy() can just measure whether the "relative color" is similar to the original image, rather than requiring the color to be absolutely consistent with the original image.

We use $keras$ and $tensorflow$ as model building tools to implement our baseline and main approach. These two methods have same difference from reference: our learning rate is very small. We found that the preset $0.001$ cannot be used as the learning rate, which will cause the results to not converge. The model will end the training process early and let every picture looks like yellowing photo. Therefore, we set the learning rate to $0.0001$, which makes the training results significantly improvement.

Baseline is a simple CNN, which is the Sequential Model we made with reference to [1]. The first half of the model adopts a design similar to the classifier which is to place Convolution 2D Layer and Max Pooling 2D Layer in. And the second half is to place the Convolution 2D Layer and Up Sampling 2D Layer in. The so-called Up Sampling 2D Layer is actually just copying the value of an element several times, as Figure 2. In this way, our baseline model is completed. The training process allows the machine to learn how to use the features learned in the first half and convert it into a color image output in the second half.

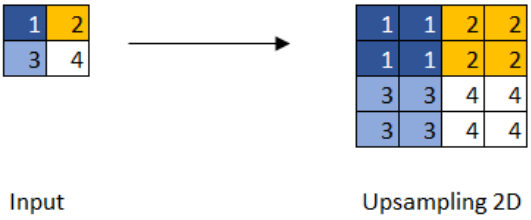



Figure 2. Up Sampling 2D Layer

The Main approach is to make some changes with the baseline. We upgrade the simple CNN to U-Net, and connect the Layers of the first half and the second half. In addition, we directly added the ready-made VGG16 Classifier to the Layer in the middle. The concept here is like ordinary people learning to color. We must first know what an object is and what properties it has before we can draw a reasonable color. The model is no longer Sequential Model, it has two different inputs, one is our gray-scale image, and the other is the output of VGG16 Classifier, which is a $1 \times 1000$ one-dimensional array.

The encoder (as figure 3) is similar to the first half of the baseline. In the decoder, compared with the second half of the baseline, we additionally use the concatenate layer to connect the decoder with the same-level encoder, and replace the Upsampling2D Layer with a learnable Transpose Convolution 2D Layer. The middle of the model is to combine the the output of the encoder with the output of the classifier. Since Maxpooling2D $((2, 2))$ appears $4$ times in the encoder, the output shape of the encoder will be $\frac{256}{2^4} \times \frac{256}{2^4} \times$ filter_num, which is $16 \times 16 \times$ filter_num. Hence Repeat Vector repeats the VGG16 output in $16 \times 16$ times, then reshapes it into a shape of $16 \times 16 \times 1000$. Finally, we uses the concatenate layer to combine the output of the encoder with the output of the repeated and reshaped VGG16 Classifier, and then add two layers of Convolution 2D to complete the middle part.

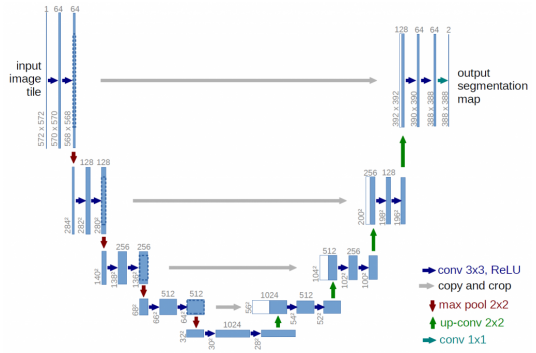


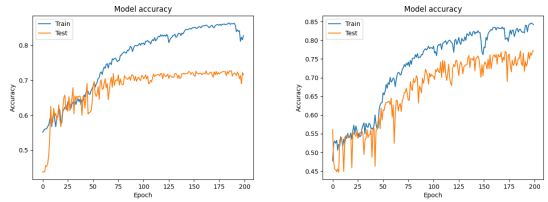

Figure 3. U-net (the left half is called the encoder, the right half is called the decoder, and the connection method is the gray arrow.)

Finally, we replace all ReLU by LeakReLU (alpha=0.2) in the models to let activation function not to be too monotonous. This change makes the results better and less likely to fall into local optimal, but the hardware requirements will be higher.

## 4. Experiments

After training, agent can infer the testing dataset. We compare beta_v2(baseline, epoch=200, batch size=16) and u-net LeakReLU (main approach, epoch=200, batch size=8) for the following examples. By the way, it has been tested that when the batch size is larger, it will have better results, but if batch size is too large, it will be limited by the hardware and cannot be trained.

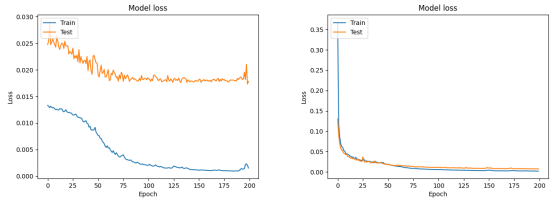
For Accuracy, as shown in Figure 4, main approach (b) have higher value of test data than baseline (a).




(a) CNN beta_v2 (b) U-net LeakReLU

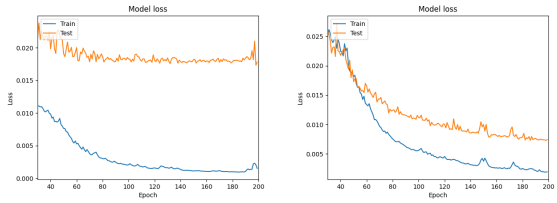
Figure 4. Accuracy

For their MSE Loss, as shown in Figure 5, obviously (b) converges better than (a). Then enlarge Figure 5, as shown in Figure 6, which help us to see the detail of (b). This can help us to compare the convergence of the baseline and the main approach.




(a) CNN beta_v2 (b) U-net LeakReLU

Figure 5. Loss (MSE)




(a) CNN beta_v2 (b) U-net LeakReLU

Figure 6. Loss after 30 epoch

There is still a lot of direction for research on Image Colorization, such as adding GAN to determine whether a prediction result is a reasonable coloring to replace the rough MSE loss, or using Mask R-CNN to see if the result will be better than U-net or not.

In the end, we try to use above agent to color manga, However, the result is not very ideal. Instead, the effect of the app on the Internet is better, probably because the agent on the Internet has a higher degree of learning, and able to identify various objects and give relatively reasonable colors. But the problem remains that the colors may deviate from the anime's color scheme. Based on the fact that we don't have such good hardware for the agent to learn, we may reconsider whether there are other ways in the future.

## References

[1] Emil Wallner (2017). How to colorize black & white photos with just 100 lines of neural network code. Retrieved 2022, May 24 from https://emilwallner.medium.com/colorize-b-w-photos-with-a-100-line-neural-network-53d9b4449f8d. 1, 2

[2] Jeremy Zhang (2019). Unet — line by line explanation. Retrieved 2022, Jun 4 from https://towardsdatascience.com/unet-line-by-line-explanation-9b191c76baf5. 1

[3] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp.234-241, Springer, 2015. 1