Deep Koalarization: Image Colorization using CNNs and Inception-Resnet-v2

E4040.2021.SPRING.OJBK.report
Shengjie Lin SL4830, Yaoxuan Liu YL4578
Columbia University

Abstract

The project aims to develop a new deep learning model which combines a deep Convolutional Neural Network trained from scratch with high-level features extracted from the Inception-ResNet-v2 pre-trained model to complete image colorization tasks. Students do a comprehensive review of the paper, and then reproduce the results of it by recreating the neural network and architecture with Tensorflow and Python code. The results show that our approach is able to successfully colorize high-level image components such as the sky, the sea, the tree, the ground, and the skin. And the performance highly depends on the specific contents in the images. The comparisons between our results and the results in the original paper are fully discussed.

1. Introduction

Coloring gray-scale images can be very useful in many domains such as processing historical images and enhancement of surveillance feeds. A variety of apps and software that can reproduce old photos with color are also popular. With the development of deep learning, models such as Inception [1], ResNet [2], or VGG [3] can be generally used to train colored image datasets. These models can help to improve the results as a prior colorization step when applied on gray-scale images.

However, designing and implementing an effective and reliable architecture that automates this process is still a challenging task. The preprocessing of the training data also needs to be elaborately conducted. In recent years, CNNs have been proven to obtain accurate results on object recognition. In this regard, some previous research has been done. Zhang et al. [4] proposed a multi-modal scheme where each pixel was given a probability value for each color. Larsson et al. [5] adopted fully convolutional VGG-16 [6] with a classification layer excluded to evaluate a color probability distribution of each pixel. Another approach is developed by Iizuka, Serra et al. [7] based on using global-level and mid-level features to encode the images. Inspired by these methods, we transfer a pre-trained model, namely Inception-ResNet-v2, into a deep CNN encoder-decoder architecture to colorize the gray-scale image.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

First of all, the original paper consider images with size H*W in the CIE L*a*b color space [8] where each space has been demonstrated in Fig. 1. This kind of color space can separate the color characteristics from the luminance that contains the main image features in a gray-scale image. The author assumes that there is a mapping between L and a, b which means that a & b channels would be predicted after the L channel is input. Combining the luminance with the predicted color components, a final colorized image can be generated with a high level of detail.

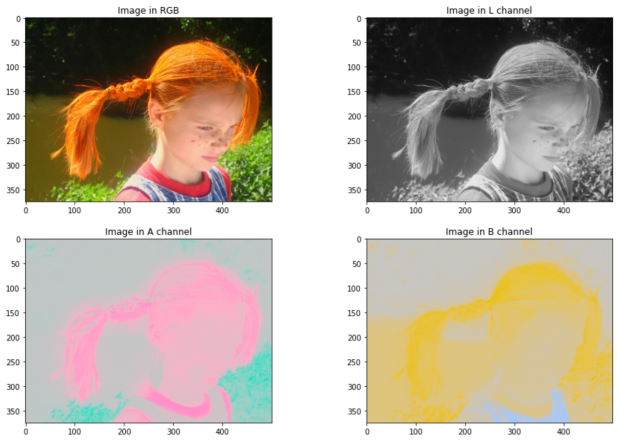

Fig. 1: Demonstration of L*a*b color space

Next, the model architecture is divided into four main components which is shown in Fig. 2. The encoder component can obtain the mid-level features, while the feature extractor can identify the high-level features in images. These two parts are then merged as a fusion layer. Finally, the decoder uses these features to predict the output.

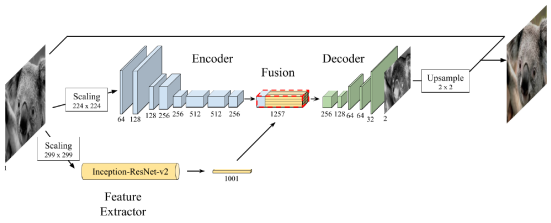

Fig. 2: Overview of the deep learning model architecture

The detailed information about network layers is listed in table 1 below. The left part is the encoder network, the mid part is the fusion network, and the right part is the decoder network. The activation function of each layer is ReLu, except for the last layer which adopts the hyperbolic tangent function. The feature extraction part is a transferred architecture from Inception-Resnet-v2, discarding the last softmax layer.

Layer	Kernels	Stride	Layer	Kernels	Stride	Layer	Kernels	Stride
conv	$64 \times (3 \times 3)$	2×2	fusion	-	-	conv	$128 \times (3 \times 3)$	1×1
conv	$128 \times (3 \times 3)$	1×1	conv	$256 \times (1 \times 1)$	1×1	upsamp	-	-
conv	$128 \times (3 \times 3)$	2×2				conv	$64 \times (3 \times 3)$	1×1
conv	$256 \times (3 \times 3)$	1×1				conv	$64 \times (3 \times 3)$	1×1
conv	$256 \times (3 \times 3)$	2×2				upsamp	-	-
conv	$512 \times (3 \times 3)$	1×1				conv	$32 \times (3 \times 3)$	1×1
conv	$512 \times (3 \times 3)$	1×1				conv	$2 \times (3 \times 3)$	1×1
conv	$256 \times (3 \times 3)$	1×1				upsamp	-	-

Table 1: Specific network layers

Then, authors of the original paper use 60,000 images ImageNet dataset to train, and 10% of them are used as validation data. The optimal parameters are found by minimizing the loss function as below.

$$C(\mathbf{X}, \boldsymbol{\theta}) = \frac{1}{2HW} \sum_{k \in \{a,b\}} \sum_{i=1}^{H} \sum_{j=1}^{W} (X_{k_{i,j}} - \tilde{X}_{k_{i,j}})^2,$$

where we calculate the Mean Square Error (MSE) between the predicted value in a*b* space and their true value. The optimizer they use is Adam to update the model parameters $\boldsymbol{\theta}$ by backpropagation. The initial learning rate is set as $\eta = 0.001$.

Finally, other than presenting the training result, they also conduct a user study to evaluate how compelling the colors look to a human observer by questioning them whether the image is fake or real.

2.2 Key Results of the Original Paper

The results of the original paper turn out to be quite good for some of the images. However, some small objects or some portions of the images are still in the gray-scale or the wrong color. For example, Fig. 3 demonstrates the comparison between their results and ground truth images. It can be seen that the sky, the water, the leaves have been well colorized. But the butterfly has not been colorized. And the people's clothes have been changed from green-yellow to red.

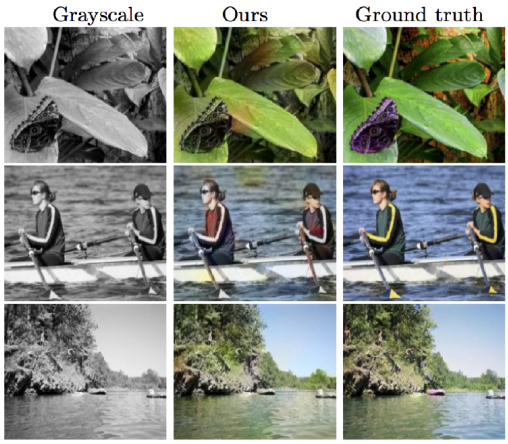

Fig. 3: Comparison of their predicted images and ground truth images

3. Methodology (of the Students' Project)

The students' project successfully reproduces most of the methodology in the original paper. But there are still some differences in detail.

Firstly, in the fusion component, we need to mirror the feature vector (*1001*1*1*) and concatenate it multiple times to make the semantic information conveyed by the feature vector uniformly and fully distributed in the whole region of images. Compared with the original paper, we write it in a different way that we directly use the RepeatVector and Reshape function imported from Keras. The pseudo-code is described as below:

*IR_v2_output = RepeatVector(28*28)(IR_v2_output)*
IR_v2_output = Reshape((28,28,1256))(IR_v2_output)

Secondly, in the original paper, the authors obtain a *1001*1*1* tensor from the feature extractor path and then apply it as an input in the encoder-decoder model. However, in our architecture, the encoder, feature extractor, and decoder are connected and combined as one model.

In addition, the dataset we choose is also different from the original paper. The authors of the original paper use the ImageNet dataset with 60,000 images having different sizes. However, we choose to use Places training dataset, which is the same as Iizuka et al. In order to save time on training, we use the small validation images (*256*256*) in

Places as our training set. The file contains 36,000 images with the same H*W shape (*256*256*) and the file sizes of them are quite small (545M in total).

Finally, in the training process, we borrow the idea from the decay rate in Tensorflow to train the model two times with a smaller learning rate. The initial learning rate in the first time training $\eta 1$ = 0.001 and the number of epochs is 10. For the second time training, the learning rate is set as $\eta 2$ = 0.0001 and the number of epochs is 15. The optimizer used for training is Adam. The batch size in our model is 32.

3.1. Objectives and Technical Challenges

The objective of the student's project is to try to reproduce the whole approach and architecture in the original paper, compare and discuss the differences in the model and results. Also, the optimal parameters in our model need to be found in the training process.

The technical challenges mainly lie in the component of image processing. The original RGB channels of training images range from [0, 255], but the value ranges of CIE L*a*b color spaces are different where the range of L channel is [0, 100] and range of a & b channels are [-128, 128]. Then, the pixel values of all three channels are centered and scaled within the interval of [-1, 1] respectively. In addition, the original shape of images (*256*256*) needs to be scaled into different shapes for the encoder (*224*224*) and the feature extractor (*299*299*).

All of these changes in the image processor need to be carefully designed and operated in the model. Finally, these biggest technical challenges are solved by drawing a clear structure of the whole process which is shown in Fig. 4.

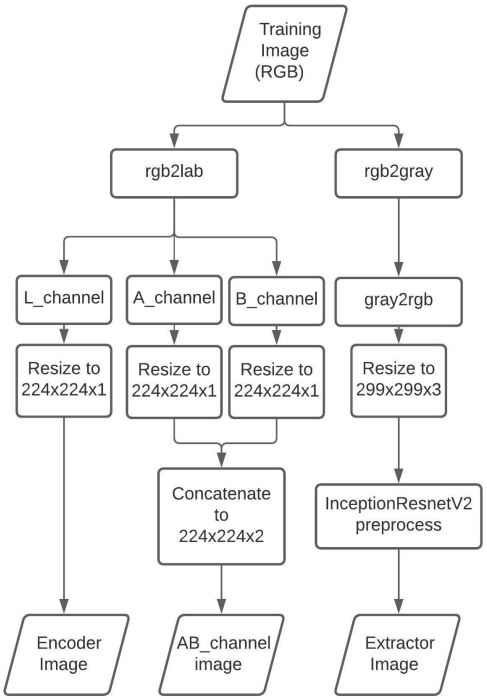

Fig. 4: Structure of "Image processor"

3.2. Problem Formulation and Design Description

Fig. 5 describes the flow chart of our project. After we obtain the raw training data, the file names of them are required to be appended into one list. Then, we need to load and generate data for the training set and validation set respectively. Detailed information about the data generator is written as a class using Python code which is uploaded as the utils file in GitHub. The image processor process has been fully discussed in 3.1 as shown in Fig. 4. When all of these steps have been finished, the training set is well-prepared for training in the model. Once trained, we fed the network with some images, and the predicted result will be generated.

Fig. 6 shows the block diagram of the whole model architecture in our project. The L channel images generated after image processing are scaled into the encoder and feature extractor. And they will be concatenated together as a single volume with features of shape *H/8*W/8*1257*. Then, applying 256 convolutional kernels of size *1*1*, we can generate a feature volume of *H/8*W/8*256* which is applied series of convolutional and up-sampling layers to obtain a final layer with dimension *H*W*2*. The loss function is derived by

calculating the MSE between the final layer and the a & b channel images generated by the image processor.

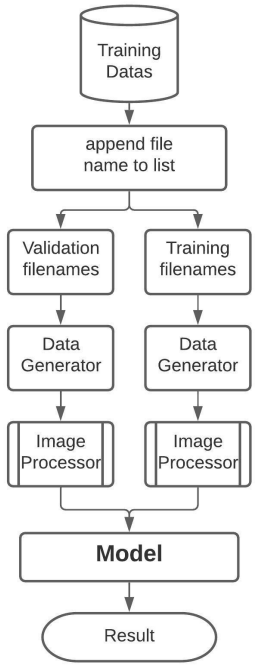

Fig. 5: Flow chart of the project

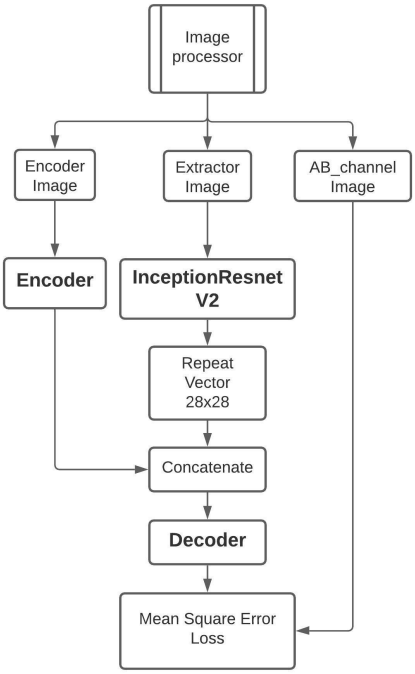

Fig. 6: Block diagram of the whole model architecture

4. Implementation and results

The network was trained and tested on the Google Cloud Platform (GCP) with the instance created by ourselves. And complete details about the code, the architecture, the implementation, and training results can be found on the GitHub page of our project.

4.1. Project Results

Fig.7 illustrates results for some examples that our network produces after predicting the colorization. Generally speaking, the results turn out to be pretty good which shows basic agreements with the ground truth photo. Most regions in the images have generated near-photorealistic colors, such as the tree, the sky, the ground, the water, and the grass. However, the color saturation of our predicted images is lower than that of the true images. In addition, some small portions or specific objects in the images are not well colored. More discussions on the results will be presented in 4.3.

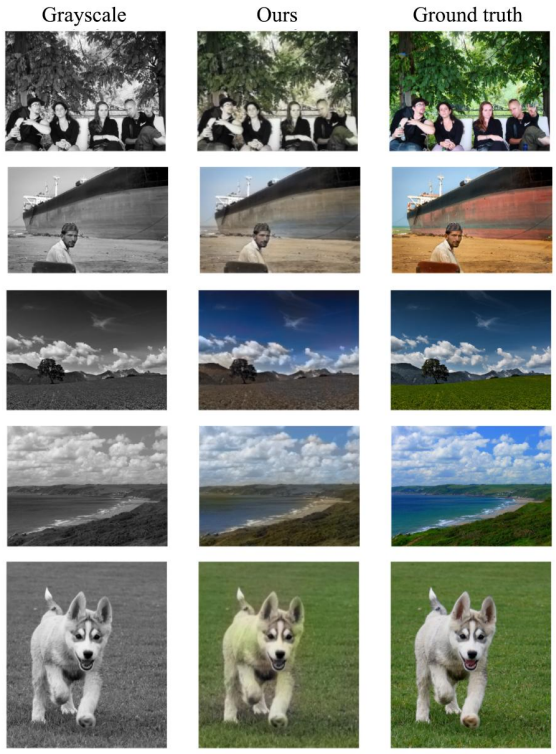

Fig. 7: Some result examples from students' network

The total training time is approximately 22 hours. Fig. 8 shows the relationship between the epoch and MSE loss. As more epochs have been trained, both the training and

validation loss are gradually reduced. In particular, when the second time training started with a smaller learning rate, the decrease of loss is more obvious. The ultimate training and validation loss is relatively small which means the network has been well trained.

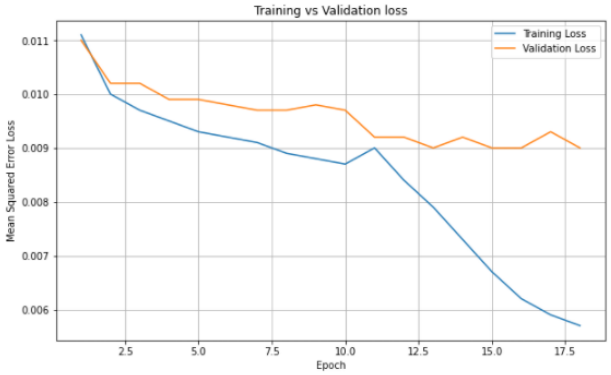

Fig. 8: Loss of training set and validation set

4.2. Comparison of the Results Between the Original Paper and Students' Project

The generated image results indicate that students and authors of the original paper can achieve a similar colorization pattern on gray-scale images. The overall performance of our project is pretty good, but still not as good as their project.

In the first and second row, our approach does not work well to recognize the green vegetation and the lake which are successfully colorized in the authors' results. The butterfly and the rower's clothes are colorized incorrectly, while they are only slightly colorized and largely keep the grayscale in our results. The last row demonstrates a landscape example where both two results can provide a vivid and accurate image.

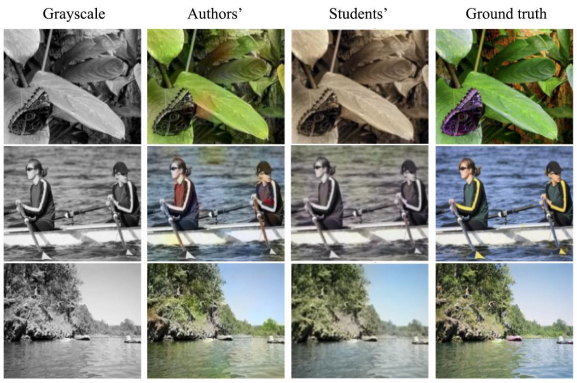

Fig. 9: Comparison of the results between the original paper and students' project

4.3. Discussion of Insights Gained

From the whole project and the results, a lot of useful insights can be gained regarding image colorization.

The most inspiring one is that the result of colorization largely depends on the quality of datasets. If the dataset contains too many disturbing and non-effective images, the training process as well as results will be greatly influenced. Fig. 10 demonstrates two examples to tell us what is bad training data and what is a good one. The left image provides no useful information because the main body of it is in black. Also, the two motorcycles are in two different colors which makes the color information more complex. The right image can be regarded as a good training image which has a large area of green plants, the ground, and human figures. The features of them are easily recognized and the colors are typical.

Fig. 10: Bad training data and good training data

Then, the size of the training set is very important as well. If the size of the training set is small, the network will perform better only when certain image features appear. The model will have a low capacity. If the size of the training set is too large, some generated pictures will tend to be low saturated, and specific objects will present a grayish color. The reason is that if the network attempts to minimize the loss between images where the colors are different for the same object, a conservative prediction will be conducted by assigning a neutral gray color. It is also a possible reason for the low saturation of our results.

In addition, when training a CNN model, applying a decay rate to the learning rate will greatly improve the training performance.

Finally, we want to make a discussion about the white padding. In order to preserve the aspect ratio of the image, the authors add the white padding in the input. However, it may take lots of training data and epochs to make the CNN understand the function of these white paddings instead of regarding them as part of the images. Therefore, in our project, we discard the use of white padding to save computational power. Each image may

get stretched or shrunk, but the influence is eliminated because the size of images we use are the same in width and height.

5. Conclusion

The project validates an end-to-end deep learning model, combining a deep Convolutional Neural Network with an Inception-ResNet-v2 transferring model, could be suitable for image colorization tasks. Students do a comprehensive review of the paper, and then reproduce the results by recreating the neural network and the architecture with Tensorflow and Python code written by ourselves. The results show that our approach is able to successfully obtain similar and satisfying colorization results which achieves the objectives and goals of the project. However, the performance is highly dependent on their specific contents in images. From this process, we learn that the quality and size of the dataset will be extremely important for colorization tasks. We believe that image colorization still has a huge potential in the future with better and larger dataset. It could also be interesting to apply the techniques to grayscale videos.

6. Acknowledgement

We would like to appreciate Prof. Zoran Kostic, TAs Desmond Yao, Hongzhe Ye, Zhuoxu Duan, and Richard Samoilenko for your efforts and help throughout the whole semester.

7. References

[1] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. CoRR abs/1512.00567 (2015)
[2] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR abs/1512.03385 (2015)
[3] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR abs/1409.1556 (2014)
[4] Zhang, R., Isola, P., Efros, A.A.: Colorful image colorization. In: European Conference on Computer Vision, Springer (2016) 649-666
[5] Larsson, G., Maire, M., Shakhnarovich, G.: Learning representations for automatic colorization. In: European Conference on Computer Vision, Springer (2016) 577-593
[6] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. ICLR (2015)
[7] Iizuka, S., Simo-Serra, E., Ishikawa, H.: Let there be color!: joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. ACM Transactions on Graphics (TOG) 35(4) (2016) 110
[8] Robertson, A.R.: The cie 1976 color-difference formulae. Color Research & Application 2(1) (1977) 7-11

8. Appendix

8.1 Individual Student Contributions in Fractions

	sl4830	yl4578
Last Name	Lin	Liu
Fraction of (useful) total contribution	50%	50%
Code	70%	30%
Results and Discussion	50%	50%
Report writing	30%	70%

8.2 Other relevant sources

Places dataset: http://places2.csail.mit.edu/download.html

Color space definitions in python (RGB and LAB): https://fairyonice.github.io/Color-space-defenitions-in-python-RGB-and-LAB.html

A detailed example of using data generators with Keras: https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly