Final Project Report

Authors:	*StudentID:*
Fateme Noorzad	810198271
Amirhossein Rokni Lamoki	810198303
Ali Elahi	810696336

Deep Learning and Applications

Spring 2021

Contents

Chapter 1

Introduction

In project, we are to simulate the given paper under "Stylization-Based Architecture for Fast Deep Exemplar Colorization" name. In the upcoming chapters. First of all, a summery of this paper is given. Then, in the next chapters, the simulation procedure as well as its results are given.

Chapter 2

Summery of the Given Paper

2.1 Introduction

Colorization is a classic task in computer vision which aims to add color to a gray image. The reason why this task is popular among people is that this addition can make images more visually plausible and perceptually meaningful. In addition, it has colossal application in practical usages.

It is worth mentioning that although colorization is a very worldwide task, it is full of ambiguity and diversity. The reason lying behind the mentioned features is that there are no unique answer when dealing with such problems. To make this point clear, think we want to color a car on a street. As it is clear, cars can be found in various color, hence the solution to this problem. This causes colorization problem to be a challenging one.

One approach in colorization, which results in high accuracy is the intervention of humans. This approach of solving colorization matter was a well liked one in previous works. Some papers [1] [2] [3] propose a method in which a user manually adds colors to a grayscale image meticulously, then the given patter of colors are propagated in the whole image. This method, although seems very encouraging, can be a challenging one for an untrained user with the lack of skills and/or art sensitivity.

Other related works [4] [5] try to weaken the intervention of unskilled human beings by replacing them with a related reference image. However, this resulted in a inferior image since the reference image had dissimilarity with the target one in viewpoints, content, and even lightening.

In the recent years, with the advances in deep learning, colorization techniques based on this approach became well received due to their remarkable results. [6] [7] [8] In this method, a colorization network is implemented and then trained on a large

[1] Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. 23(3):689–694, 2004.

[2] Qing Luan, Fang Wen, Daniel Cohen-Or, Lin Liang, Ying-Qing Xu, and Heung-Yeung Shum. Natural image colorization.In Proceedings of the 18th Eurographics conference on Rendering Techniques, pages 309–320. Eurographics Association,2007.

[3] Liron Yatziv and Guillermo Sapiro. Fast image and video colorization using chrominance blending. IEEE transactions on image processing, 15(5):1120–1129, 2006.

[4] Xiaopei Liu, Liang Wan, Yingge Qu, Tien-Tsin Wong, Stephen Lin, Chi-Sing Leung, and Pheng-Ann Heng. Intrinsic colorization. 27(5):152, 2008.

[5] Alex Yong-Sang Chia, Shaojie Zhuo, Raj Kumar Gupta, Yu-Wing Tai, Siu-Yeung Cho, Ping Tan, and Stephen Lin. Semantic colorization with internet images. 30(6):156, 2011.

[6] Zezhou Cheng, Qingxiong Yang, and Bin Sheng. Deep colorization. In IEEE ICCV, pages 415–423, 2015.

[7] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Learning representations for automatic colorization. In IEEE ECCV, pages 577–593. Springer, 2016.

[8] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In ECCV, pages 649–666. Springer, 2016.

number of image dataset to learn the relationship between the grayscale image and its corresponding color version. The only arduous task is to determine the network parameters. When this condition is satisfied, the coloring quest is a piece of cake. However, it is worth noting that although this method has extraordinary success, without any human intervention its results are uncontrollable.

In the most recent works, the controllability from interaction of human and robustness from learning to achieve more accurate and promising colorization has been combined. [9] [10] [11] In this paper, same as these works, a novel deep learning method for fast exemplar-based colorization inspired by the works done in the stylization networks' field [12] [13] is proposed. The proposed architecture consists of two parts: The transfer sub-net and the colorization sub-net. In the upcoming sessions these two parts are elaborated completely.

Experiments which are explained in the next sections show that the proposed work outperforms the state-of-art exemplar-based colorization methods in less time. In addition, the transfer sub-net presents remarkable effects in photorealistic image stylization tasks.

Thus, the main contribution of this approach can be itemized as below:

- A novel two sub-nets architecture is proposed that jointly learns faithful colorization with a related reference and plausible color prediction with an unrelated one.
- Better colorization is achieved in less time compared with other exemplar-based methods by using the AdaIN operation for feature matching and blending.
- The transfer sub-set can be extended to photorealistic image stylization without any additional modification.

2.2 Related Work

Current colorization methods can be divided into four categories. In the below sub-sections, these categories are represented. In the next sub-section the photorealistic image stylization is discussed.

2.2.1 Colorization

Scribble-based Colorization

This method is one of the interactive methods that propagates the initial strokes to the whole grayscale image. In order to perform the propagation, low-level similarity metrics [14], such as spatial offset and intensity difference are carried out. In fact, Levin

[9] Richard Zhang, Jun-Yan Zhu, Phillip Isola, Xinyang Geng, Angela S Lin, Tianhe Yu, and Alexei A Efros. Real-time user-guided image colorization with learned deep priors. ACM Transactions on Graphics, 36(4):1–11. 109372

[10] Patsorn Sangkloy, Jingwan Lu, Chen Fang, Fisher Yu, and James Hays. Scribbler: Controlling deep image synthesis with sketch and color. In IEEE CVPR, pages 5400–5409, 2017.

[11] Mingming He, Dongdong Chen, Jing Liao, Pedro V Sander, and Lu Yuan. Deep exemplar-based colorization. ACM Transactions on Graphics (TOG), 37(4):47, 2018.

[12] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In IEEE ICCV, pages 1501–1510, 2017.

[13] Yijun Li, Ming-Yu Liu, Xueting Li, Ming-Hsuan Yang, and Jan Kautz. A closed-form solution to photorealistic image stylization. In IEEE ECCV, pages 453–468, 2018.

[14] Similarity metrics determine the most similar obj with the highest values as it implies they live in closer neighborhoods.

et al. [15] solve a Markov Random Field with the assumption that adjacent pixels with similar intensity should have akin colors to propagate color points. Later methods aim to present advanced similarity metrics. One of which is a method in which by employing the energy-optimization formulation, the propagation is done.

Although these methods are capable of providing plausible colorization results when given good prior colors, the challenge of an unprofessional colorization by an untrained user is still present.

Exemplar-based Colorization

This methods perform the coloring task by the guidance of a reference image. In fact, they can be divided into main categories:

- **Global Transfer Method**: In this method, by matching global color statics like mean, variance, and histogram, the transformation of color from the reference image to the target one is performed. [16] Nevertheless, due to the fact that in this approach th spatial pixel information is ignored, the results are unrealistic.

- **Local Transfer Method**: This method considers different level of correspondences, such as pixel level [17], superpixel level [18] [19], and segmented region level [20]

As was discussed earlier, these methods are susceptible to generate terrible results when the two images have various appearances but perceptually similar semantic structures.

Learning-based Colorization

In these methods, end-to-end networks are trained to perform the automatic colorization. In details, these networks reconstruct an image by predicting every pixel of the target image with loss functions such as L_2 loss [21], classification loss [22], and $L_1 + GAN$ [23]. All of mentioned studies learn the parameters of the network parameters from huge image datasets automatically without any user intervention. Nevertheless, most of these methods produce a single accaptable image for each grayscale image

[15] Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. 23(3):689–694, 2004.

[16] Chufeng Xiao, Chu Han, Zhuming Zhang, Jing Qin, and Shengfeng He. Example based colorization via dense encoding pyramids. Computer Graphics Forum, (12), 2019.

[17] Xiaopei Liu, Liang Wan, Yingge Qu, Tien-Tsin Wong, Stephen Lin, Chi-Sing Leung, and Pheng-Ann Heng. Intrinsic colorization. 27(5):152, 2008.

[18] Alex Yong-Sang Chia, Shaojie Zhuo, Raj Kumar Gupta, Yu- Wing Tai, Siu-Yeung Cho, Ping Tan, and Stephen Lin. Semantic colorization with internet images. 30(6):156, 2011.

[19] Raj Kumar Gupta, Alex Yong-Sang Chia, Deepu Rajan, Ee Sin Ng, and Huang Zhiyong. Image colorization using similar images. In Proceedings of the 20th ACM inter-national conference on Multimedia, pages 369–378. ACM, 2012.

[20] Revital Ironi, Daniel Cohen-Or, and Dani Lischinski. Colorization by example. In Rendering Techniques, pages 201–210. Citeseer, 2005.

[21] Satoshi Iizuka, Edgar Simo Serra, and Hiroshi Ishikawa. 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):110, 2016.

[22] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Learning representations for automatic colorization. In IEEE ECCV, pages 577–593. Springer 2016.

[23] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In IEEE CVPR, pages 1125–1134, 2017.

with no doubt about the colorization. In addition, the results are uncontrollable without any user interactions.

Hybrid Colorization

The most recent methods, as was mentioned before, combine the controllability and robustness together. Some of each are:

- Combining scribble-based methods with learning based methods by providing strokes.[24] [25]
- Utilizing histogram of reference image as a guidance for colorization by pyramid structure network.[26]
- Implementing a similarity network to build bidirectional mapping. Although this method ensures proper colorization even when the reference does not have the desired conditions, it is computationally inefficient, specially in the case of unseen images and objects.

2.2.2 Photorealistic Image Stylization

This method is evolved from the traditional style transfer. [27] [28] The most discrepancy is that the output of the former should still maintain the original edge structure clearly. To suppress distortions appearing in stylization results, DSPT [29] adds a regularization term to the loss function of the neural style algorithm. In another work [30] a modified Whitening and Coloring Transforms (PhotoWCT) model that utilizes unpooling layers and additional smooth operations are implemented. Due to the fact that these methods have poor robustness and cause artifacts and blurriness in the result images, they cannot be directly employed in the colorization task. Hence, inspired from these methods, the proposed method in this paper utilizes a transfer sub-net which can not only be used in the image stylization task but also can generate an appropriate chrominance map for gray input.

2.3 Stylazation-Based Colorization

As was explained before, the major problem of the classic exemplar-based colorization lies in the selection of reference image. In fact, finding an image as the reference which matches all the charactristics and objects in the target image is an arduous task. In addition, this procedure is a time consuming one. Hence, the proposed

[24] Richard Zhang, Jun-Yan Zhu, Phillip Isola, Xinyang Geng, Angela S Lin, Tianhe Yu, and Alexei A Efros. Real-time user-guided image colorization with learned deep priors. ACM Transactions on Graphics, 36(4):1–11.

[25] Patsorn Sangkloy, Jingwan Lu, Chen Fang, Fisher Yu, and James Hays. Scribbler: Controlling deep image synthesis with sketch and color. In IEEE CVPR, pages 5400–5409, 2017.

[26] Chufeng Xiao, Chu Han, Zhuming Zhang, Jing Qin, and Shengfeng He. Example based colourization via dense encoding pyramids. Computer Graphics Forum, (12), 2019.

[27] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In IEEE CVPR, pages 2414–2423, 2016.

[28] Leon A Gatys, Alexander S Ecker, Matthias Bethge, Aaron Hertzmann, and Eli Shechtman. Controlling perceptual factors in neural style transfer. In IEEE CVPR, pages 3985–3993, 2017.

[29] Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. Deep photo style transfer. In IEEE CVPR, pages 4990–4998, 2017.

[30] Yijun Li, Ming-Yu Liu, Xueting Li, Ming-Hsuan Yang, and Jan Kautz. A closed-form solution to photorealistic image stylization. In IEEE ECCV, pages 453–468, 2018.

method aims to generate plausible colorization result in real time, no matter how related the two input images are.

Here, we denote the target image as T and the reference image as R while we operate mainly in the CIELab space. Plus, as was discussed earlier, to reach the set goal, a network with two sub networks is implemented. Below, the architecture of the mentioned network can be found. In addition, in the upcoming paragraphs, each sub-net of the proposed method is elaborated.

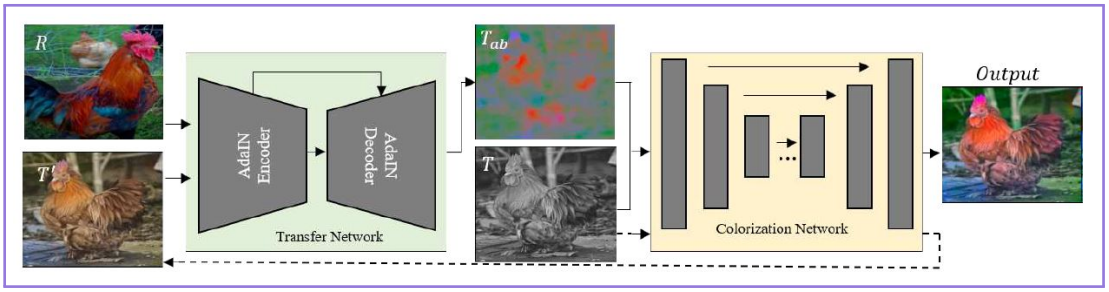

FIGURE 2.1: System pipeline

2.3.1 Transfer Sub-Net

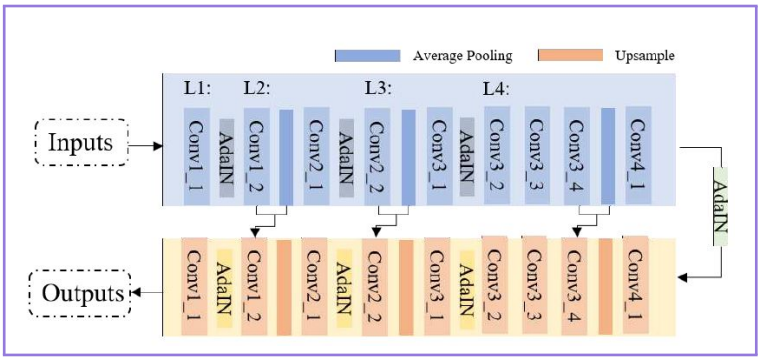

FIGURE 2.2: Detailed architecture of transfer sub-net.

The transfer sub-net is inspired by the recent observations that style transfer in real time is possible and feature extracting, feature blending, and content-consistency are among the tasks which photorealistic image stylization performs well on them.

As Fig. 2.1 suggests, the transfer sub-net generates an initial ab map, for the target gray image by matching its basic feature statistics with reference image in different layers directly. In fact, the coarse chrominance imformation for the target image is obtained by matching and blending the features of the reference with those of the target.

In order to implement this network with the architecture which is elaborated in Fig. 2.2, the pre-trained VGG19 module from $conv1\_1$ layer to $conv4\_1$ layer is utilized as the encoder. Based on this design for layers, in a symmetrical manner, the decoder is implemented as well.

One point worth mentioning here is that multi-decoders that were employed in stylization methods, destroy the property of VGG19 network in some cases and amplify the artifacts and blurriness in images. Hence, for the proposed network, a single pass encoder-decoder network is used.

In details, two input images are fed into the pre-trained encoder and the intermediate output of the $conv\{i\}\_1$ layers as the feature representation. In fact, to

match the features, WCT [31] calculates SVD to project the content features to the eigen-space of style features. However, in the case of large images, the cumbersome computation is intolerant for a regular computer. To accelerate this procedure, the fast Adaptive Instance Normalization (AdaIN) [32] is utilized. In details, AdaIN is a variant of IN [33] that is proposed to replace the batch normalization (BN) layers in the stylization network. AdaIN extends IN by adaptively computing the mean and variance independently for each channel and each sample. Mathematially speaking, for a content input x and a style input y, AdaIN is defined as:

$$AdaIn(x, y) = \sigma(y)\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \mu(y)$$

where $\sigma(.)$ is the standard deviation and $\mu(.)$ is mean.

In a comparision between BN, IN, and AdaIN, AdaIn has no learnable affine parameters, hence its speed and success in real-time applications. As Fig.2.2 suggests, the AdaIN operation is carried out after each $conv\{i\}\_1$ layer in the transfer sub-net. It is resposible to deal with arbitrary reference images and generate reliable *ab* map for the subsequent colorization sub-net.

On one hand, the simple architecture of the proposed model has the mentioned benefits, on the other hand, the simple symmetrical architecture is usually the cause of artifacts and distortions during reconstruction process. Thus, to solve this matter, useful information form specific layers of the encoder by skip connections are fed into the decoder.

In addition to the mentioned alternations, every max-pooling layers in the architecture of VGG19 is replaced with average pooling layers, since it has been proved that this type of layers result in more accurate results. [34]

Putting all the above alternations and notes result in the network depicted in Fig.2.2. However, there is another obstacle that needs to be tackled. The target image is a grayscale one, while the reference image is in colors, hence they cannot be fed into the same encoder network. One proposed method to solve this matter is to train a gray VGG19 which only uses the luminance channel of the the image. [35] This approach, although is plausible for its acceptable top-5 accuracy, yeilds to poor feature extracting due to neglecting the chrominance information. To disentangle this issue, in this paper, the original color VGG19 is utilized with the help of a pre-trained colorization network which supplies the gray target image's pre-color information.

In spite of the fact that the pre-color information tackles the problem of same encoder network, it still is not as accurate as desired. However, as Fig.2.3 suggests, it is a fair method.

[31] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal style transfer via feature transforms. In Advances in neural information processing systems, pages 386–396, 2017.

[32] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In IEEE ICCV, pages 1501–1510, 2017.

[33] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient f

[34] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In IEEE CVPR, pages 2414–2423, 2016.

[35] Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. 23(3):689–694, 2004.

FIGURE 2.3: Intermediate outputs of transfer sub-net with a gray VGG19 extractor and our pre-colorization strategy, respectively.

2.3.2 Colorization Sub-Net

As was mentioned in section 2.4.1, the T_{ab} obtained by the transfer sub-net is inaccurate and may cause some artifact and blurrieness especially when dealing with unrelated reference. Based on the solution stated in that section, a colorization sub-net which takes known luminance T along with initial chrominance T_{ab} as inputs is implemented.

As appealing this approach seem, there are some obstacles in the way:

- It is not trouble-free to train such network to have desirable results.
- As was mentioned multiple times, there is no "correct" solution for the colorization task.
- It is expected that the network propagate the right color to right regions.
- The network has to color objects in a meaningful manner when a reliable reference is not available.

Inspired by the design strategies in similar methods [36], the colorization sub-net introduced in this paper adopts an analogous U-Net architecture [37] which is formed by ten feature blocks and one output block. In fact, color images are randomly sampled in ab channel and fed into the network so that it can learn the complete ab information.

One point which is worth noting here is that in order to avoid the averaging problem, the loss to train this network is fomulated as below:

$$L_c = L_h((1 + \lambda M) \odot F_c(x), (1 + \lambda M) \odot y)$$

where L_h is Huber loss [38], F_c is the colorization sub-net, x and y represent the input and output respectively, M is the binary mask indicating the location of sampled ab

[36] Richard Zhang, Jun-Yan Zhu, Phillip Isola, Xinyang Geng, Angela S Lin, Tianhe Yu, and Alexei A Efros. Real-time user-guided image colorization with learned deep priors. ACM Transactions on Graphics, 36(4):1–11.

[37] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net:Convolutional networks for biomedical image segmentation. In International Conference on Medical Image Computing and Computer-Assisted Intervention, 2015.

[38] Huber loss is used to evaluate how close the output and ground truth are

channels, $\odot$ is the dot product. With the help of λ more attention is payed to the locations with meaningful ab values.

2.4 Experiments

2.4.1 Implementation Details

Transfer Sub-Net

As was discussed in section 2.4.1, the encoder is a pre-trained VGG19 network , hence the decoder needs to be trained. To so so, this module is trained on the Microsoft COCO dataset by minimizing the sum of the L_2 reconstruction loss with weight 0.8 and the perceptual loss with weight 0.2.

ADAM optimizer is used to minimize the mentioned loss with learning rate initially set to 0.0001 and then decreased by a factor of 2 every 5 epochs.

With transfer sub-net being tranied as above, more accurate feature matching and color transferring by leveraging semantic label maps in AdaIN operation when they are available can be achieved.

Colorization Sub-Net

The colorization sub-net is trained on the ImageNet dataset. The weight coefficient λ is set to 10 in all experiments. The first four blocks, use the pre-trained weights which are then fine-tuned to help training and the whole colorization sub-net is trained with ADAM optimizer. The learning rate of which is set to 0.00001 and then decreased by 10 after 10 epoch. In order to reuse the network for pre-colorizing of the target image, it is also trained for 5 epochs providing input images with empty ab map. Note that the coarse ab map obtained by transfer sub-net is randomly sampled to fit the trained colorization sub-net.

2.4.2 Ablation Study

First we feed the target image with an empty ab map to the colorzation sub-net and obtain the pre-colorized result.(Fig2.4(b)) Since there is no initial color to be propagated, the sub-net only produces an undersaturated result. Then, we feed the pre-colorized target image along with the reference to the transfer sub-net and show coarse colorization result.(Fig2.4(c)) The final result is shown in Fig.2.4(d) by using the two sub-nets. It can be seen that the result via the entire networks has saturated color and few artifacts, which demonstrates the effectiveness of two cascaded sub-net architecture.

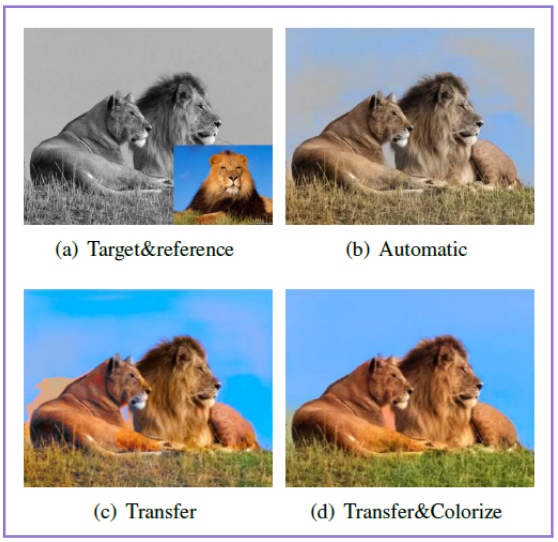

FIGURE 2.4: Ablation study about our model architecture.

AdaIN's Effect

The colorization result is shown in Fig.2.5(b). We also show the result obtained by only using AdaIN in encoder or decoder net in Fig.2.5(c) and (d), respectively. Among the three result images, Fig.2.5(b) are more saturated and with less artifacts, which is sufficient to verify the importance that AdaIN has played in the feature matching and blending process.

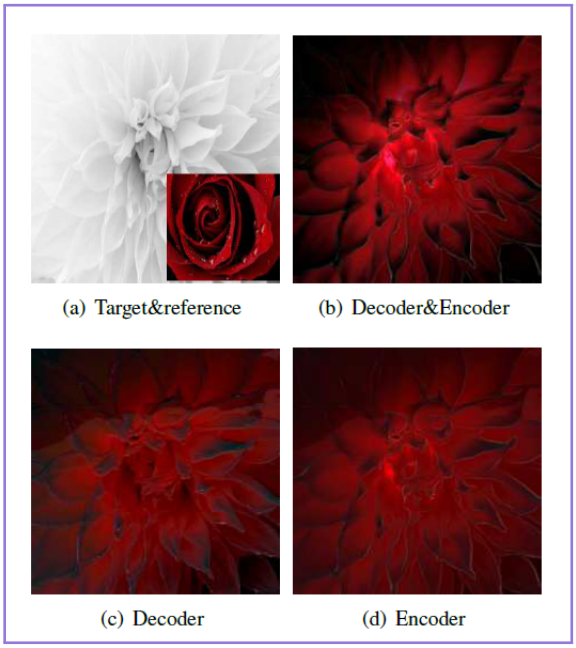

FIGURE 2.5: Ablation study about our model architecture.

2.4.3 Comparision with Other Colorization Methods

The proposed method with five existing exemplar-based colorization methods[39][40][41][42][43] are compared and present visual comparison and user study to evaluate these methods. The runtime is also compared since efficiency is one of the most important factors in the pratical application.

Another comparison case is shown in Fig.2.6, where five reference images are fed to guide the colorization of a car. The first row of Fig.2.6 shows five different reference images. Corresponding colorization results of He *et al.* and the proposed method are shown in the second and third rows, respectively. When the references also contain cars (the first four ones), both the two methods can correctly match them in the two input images in spite of different colors or shapes. The color of our results looks more saturated and vivid. The last reference is a cock that is totally unrelated to the target. The proposed method can still predict reasonable color for the target image since the colorization sub-net is trained through large-scale datasets directly without reference images. While the color of the car in result of the other source looks unnatural because their network must borrow colors from the reference, even though there is no appropriate color.

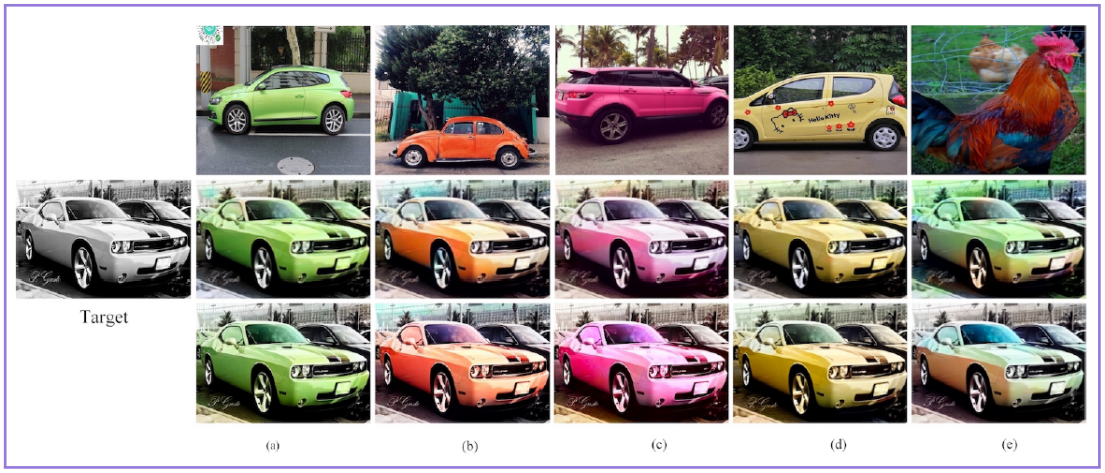

FIGURE 2.6: Results of colorization with different reference images. The first row shows five different references. The second and third row are their corresponding colorization results by He *et al.* and the proposed method, respectively.

[39]Tomihisa Welsh, Michael Ashikhmin, and Klaus Mueller. Transferring color to greyscale images. 21(3):277–280, 2002.

[40]Fabien Pierre, J-F Aujol, Aur´elie Bugeau, Nicolas Papadakis, and V-T Ta. Luminance-chrominance model for image colorization. SIAM Journal on Imaging Sciences, 8(1):536–563, 2015.

[41]Mingming He, Dongdong Chen, Jing Liao, Pedro V Sander, and Lu Yuan. Deep exemplar-based colorization. ACM Transactions on Graphics (TOG), 37(4):47, 2018.

[42]Raj Kumar Gupta, Alex Yong-Sang Chia, Deepu Rajan, Ee Sin Ng, and Huang Zhiyong. Image colorization using similar images. In Proceedings of the 20th ACM international conference on Multimedia, pages 369–378. ACM, 2012.

[43]Chufeng Xiao, Chu Han, Zhuming Zhang, Jing Qin, and Shengfeng He. Example based colourization via dense encoding pyramids. Computer Graphics Forum, (12), 2019.

2.4.4 Comparision with Stylization Methods

2.5 Conclusion and Discussion

In this paper a fast stylization-based colorization architecture which consists of two sub-nets is proposed. The transfer sub-net aims at obtaining a coarse *ab* map for the target gray image by matching features of its pre-colorized version and the reference image. The colorization sub-net trained on large-scale image dataset is used to refine the coarse *ab* map as well as provide pre-color for the target image. Its unique design of inputting sampled ab map avoids the difficulty of building training dataset for exemplar-based colorization and make the network more likely to propagate known chrominance information to semantically related areas.

Extensive experiments show that our method works well even given an unrelated reference in less time. However, there is still plenty of room for improvement. For instance, the AdaIN operation only aligns low-level feature statistics, i.e., mean and variance, which influences the matching accuracy. Besides, some colors in the coarse ab map are hardly propagated by the colorization sub-net. The reason may lies in that images in the training dataset are unevenly distributed. In the future, we will introduce higher level features for feature matching and explore more advanced network to solve these problems.

Chapter 3

Dataset and Preprocess

For the task of implementing the given paper, we had two choices for dataset: COCO dataset or some classes of ImageNet. Due to the fact that the given paper used the former dataset, it was selected. In the below section, this dataset and its features are elaborated.

3.1 COCO Dataset

The COCO dataset stands for Common Objects in Context, and is designed to represent a vast array of objects that we regularly encounter in everyday life. Format of this dataset is automatically understood by advanced neural network libraries. There are even tools built specifically to work with datasets in COCO format, e. g. COCO-annotator and COCOapi. Understanding how this dataset is represented will help with using and modifying the existing datasets and also with creating the custom ones.

- **The Computer Vision Benchmark**: The COCO dataset is labeled, providing data to train supervised computer vision models that are able to identify the common objects in the dataset. Of course, these models are still far from perfect, so the COCO dataset provides a benchmark for evaluating the periodic improvement of these models through computer vision research.
- **A Checkpoint for Transfer Learning**: Another motivation for the COCO dataset is to provide a base dataset to train computer vision models. Once the model is trained on the COCO dataset, it can be fine-tuned to learn other tasks, with a custom dataset.

3.1.1 COCO Dataset Tasks

Object Detection

Model should get bounding boxes for objects, i. e. return list of object classes and coordinates of rectangles around them. In this case objects are discrete, separate objects, often with parts, like humans and cars. The official dataset for this task also contains additional data for object segmentation. Fig.3.1 is an example of such case.

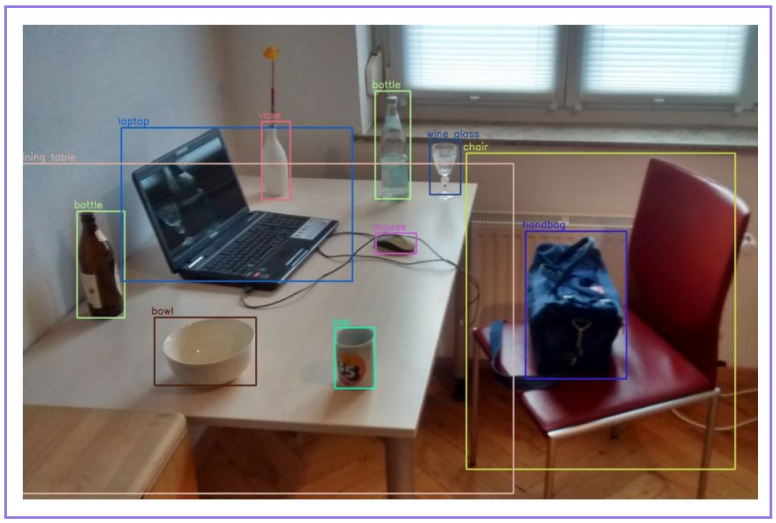

FIGURE 3.1: Objects are annotated with a bounding box and class label.

Object/Instance Segmentation

In this case the model should get not only bounding boxes for objects, but also segmentation masks, i. e. coordinates of polygon closely around the object. Fig.3.2 is an example of such case.

FIGURE 3.2: Objects are segmented with specific masks.

Stuff Segmentation

In this case model should do object segmentation, but not on separate objects, but on background continuous patterns like grass or sky. Fig.3.3 is an example of such case.

FIGURE 3.3: Background continuous patterns are segmented with specific masks.

Semantic Segmentation

In this case the boundary of objects are labeled with a mask and object classes are labeled with a class label. Fig.3.4 is an example of such case.

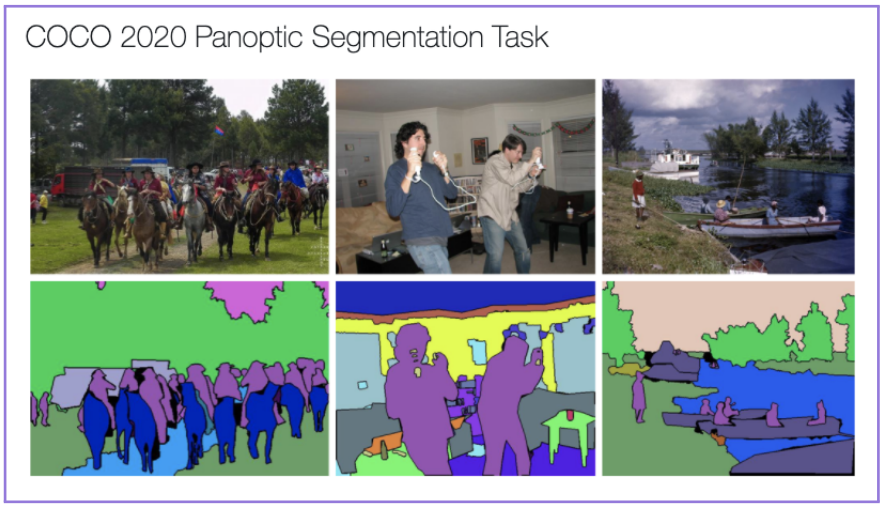

FIGURE 3.4: Panoptic Segmentation task.

Keypoint Detection

Humans are labeled with key points of interest. (elbow, knee, etc.) Fig.3.5 is an example of such case.

FIGURE 3.5: Panoptic Segmentation task.

3.1.2 COCO Dataset Format

File format used by COCO annotations is JSON, which has dictionary as a top value. Fig.3.6 elaborates this in a better manner.

FIGURE 3.6: Basic structure of the COCO JSON file.

[R.4, R.1]

3.1.3 COCO Dataset Class List

FIGURE 3.7: COCO dataset class list.

[R.2, R.3]

Due to the lack of resources, four classes of "Giraffe", "Zebra", "Snowboard", and

"Surfboard" are extracted and the transfer sub-net and total network are trained on them.

3.2 COCO Dataset Preprocessing

Now that all the details regarding this dataset is explained, we can describe the required preprocess to make the data ready for the network.

As the first step of preprocess, after loading dataset, images' pixel size are altered to be 256 × 256. Afterwards, as the paper discusses, RGB format is changed into LAB.

3.2.1 LAB

Another popular way of representing color images is with the Lab color space [1] (also knows as CIELAB).

The Lab color space expresses colors as three values:

- L: This value represents the lightness on a scale from 0 (black) to 100 (white), which in fact is a grayscale image.
- a: This value represents green-red color spectrum, with values ranging from -128 (green) to 127 (red).
- b: This value represents blue-yellow color spectrum, with values ranging from -128 (blue) to 127 (yellow).

Hence, Lab encodes an image into a grayscale layer and reduces three color layers into two. Fig.3.8 elaborates this in a better manner.

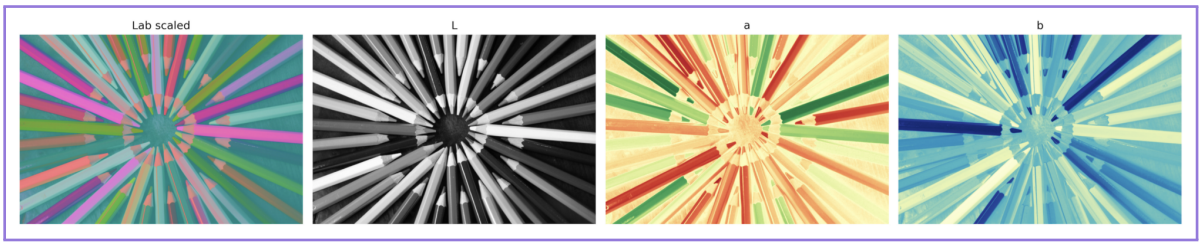

FIGURE 3.8: An image depicted in LAB color space.

[R.5] Hence, to transform the images from RGB to LAB, skimage library is employed. [R.6] However, it is essential to note that only the output of transfer sub-net is transformed in the LAB space. Then, it is fed into the colorization network.

3.2.2 Finding the Reference Image

The most challenging part in the preprocessing step is finding the reference image. This task was arduous due to the fact that for instance, if an image belonged to the apple class, there were multiple images where apples took a small portion of the image, hence have various color palettes. Thus, labeling these images as the reference image would result in nonreliable references.

To tackle this problem, we searched for the images in the dataset which have the

[1] A color model is a mathematical way of describing colors. A color space is the method of mapping real, observable colors to the color model's discrete values.

close palettes. Among all the classes, the ones related to animals and sports were suitable for our conditions. For instance, in a class under the name "Skateboard", in all images, someone is skateboarding.

By choosing the classes which had the described conditions, seven classes and about 13-14 thousand data was gathered.

Chapter 4

Transfer Sub-Net

As was discussed in chapter 2 section 2.4.1, this network consists of two main part: an encoder and a decoder. In the upcoming sections, both of these structures are discussed in details. In addition, the implementation procedures are elaborated in the related sections as well. [1]

4.1 Encoder

4.1.1 VGG19 Architecture

Based on what has been explained in the given paper, for this module, a VGG19 is employed with some alternations. The reason why this CNN is utilized is that comparing to other networks, for feature extraction, they are performing very well: [R.11]

- VGG is so big that it is incidentally capturing a lot of information that the other models discard and accidentally generalizing better despite worse task-specific performance.
- Resnets are unrolled iteration/shallow ensembles: the features do exist but they are too spread out to be pulled out easily and the levels of abstraction are all mixed up - instead of getting a nice balance of features from the bottom and top, they're spread out wildly between layer 3 and 33 and 333 etc. While VGGs, being relatively shallow and modular and having no residual connections or other special tricks to smuggle raw information up the layers, are forced to create more of a clearcut hierarchical pyramid of abstractions.
- Residual connections themselves somehow mess up the optimization procedure by affecting properties like independence of features.
- VGG's better performance is due to not downsampling aggressively, doing so only after two convolutions and then max pooling. In this interpretation, GoogLeNet fails because it downsamples in the first layer.

Style transfer uses the features found in the 19-layer VGG Network, which is comprised 16 convolution layers, 3 fully connected layer, 5 max-pool layers, and 1 SoftMax layer.

Convolutional layers are named by the stack and their order in the stack. As first convolutional layer is named as *conv*1_1 and the deepest convolutional layer is *conv*5_4. In fact Fig4.1 depicts this network and its correspoding layers' names and Table4.1 gives the dimension of each layer in details.

[1] For the train procedure, check chapter.6 under section 6.1

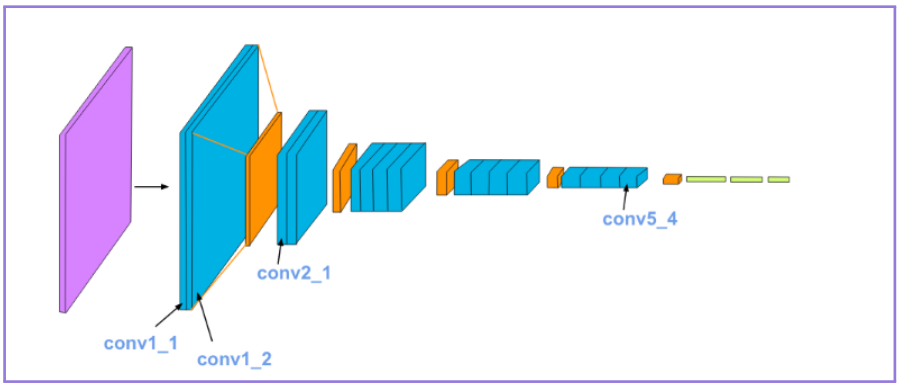

FIGURE 4.1: Detailed architecture of VGG19.

1	Conv3x3 (64)	Conv3x3 (64)	MaxPool	-	-
2	Conv3x3 (128)	Conv3x3 (128)	MaxPool	-	-
3	Conv3x3 (256)	Conv3x3 (256)	Conv3x3 (256)	Conv3x3 (256)	MaxPool
4	Conv3x3 (512)	Conv3x3 (512)	Conv3x3 (512)	Conv3x3 (512)	MaxPool
5	Conv3x3 (512)	Conv3x3 (512)	Conv3x3 (512)	Conv3x3 (512)	MaxPool
6	FC(4096)	FC (4096)	FC (4096)	SoftMax	-

TABLE 4.1: VGG19's architecture [R.12]

Style transfer relies on separating content and style of an image. Our target is to create a new image containing style of style image and content of content image(base image):

- Content(objects and their arrangement) from the given content image.
- Style(colour and texture) from given style image.

Now, to perform the above mentioned tasks, we need to take a closer look at VGG19. VGG19 is consists of two parts:

- **vgg19.features**: Convolutional and pooling layer.
- **vgg19.classifier**: Last three layers for output.

On the grounds of the mentioned items, it is clear that for style transfer we need only features portion, so we will load in that and freeze the weights. [R.10]

4.1.2 Alternations in VGG19 Architecture for Our Goal

In order to get the best out of VGG19, some alternations are done.

Replacing Max-Pooling with Average Pooling

As has been suggested in [R.7] and [R.9], replacing every max-pooling layer with average-pooling layer leads upto better results. [R.16]

Skip Connections

On the accounts of the detail in the paper regarding the skip connections from specific layers of encoder, the VGG19 is sliced into four part. With the output of these parts in hand, the aim of having realiable information from the encoder to be used in decoder can take place. [R.17]

Adaptive Instance Normalization

In chapter 2 section 2.4.1 the reason behind using AdaIN instead of other methods such as batch normalization or instance normalization is explained. However here, we want to take a closer look on what this method is, and how it is utilized in here.

Adaptive Instance Normalization is a normalization method that aligns the mean and variance of the content features with those of the style features. Unlike BN or IN, AdaIN has no learnable affine parameters. Instead, it adaptively computes the affine parameters from the style input. [R.18]

Intuitively, let us consider a feature channel that detects brushstrokes of a certain style. A style image with this kind of strokes will produce a high average activation for this feature. Moreover, the subtle style information for this particular brushstroke would be captured by the variance. Since, AdaIN only scales and shifts the activations, spatial information of the content image is preserved. [R.8]

Hence, the usage of AdaIN results in better image. However, for our case, due to the lack of resources, AdaIN is only used between the output of encoder and the input of decoder, and in both of these modules the implemented BN is employed. [R.19, R.20]

4.2 Decoder

In contrast to the encoder module, this module needs to be implemented and trained thoroughly. To do so, first of all, a unit consisting conv2d with kernel size 3 and stride 1 with padding set as 1, batch normalization as regualizer, and ReLU as activation function is implemented. With this unit in hand, four slices of decoder are created.

The decoder, same as the encoder, consists of four slices. Each slice, get a concatinated input. This input consists of the output of convolution layer from encoder and the output of the up-sampling layer.

The size of each kernel and the number of input and output channels of each convolutional layer in decoder are set based on its corresponding value in encoder. Besides, in order to reverse the average pooling layers in encoder, upsampling layers are implemented.

FIGURE 4.2: The result of employing AdaIN between then encoder and decoder. [R.18]

Chapter 5

Colorization Sub-Net

Consider the grayscale image in Fig. 5.1. At first glance, hallucinating its colors seems daunting, due to the fact that so much of the information (two out of the three dimensions) has been lost.

FIGURE 5.1: A grayscle image

On the other hand, looking more closely, it can be noticed that in many cases, the semantics of the scene and its surface texture provide ample cues for many regions in each image: the grass is typically green, the sky is typically blue, and the ladybug is most definitely red.

Of course, these kinds of semantic priors seem right and in hand, they do not work for everything, e.g., the croquet balls on the grass might not, in reality, be red, yellow, and purple (though it is a pretty good guess). In addition, as was mentioned before, there is no correct answer in the task of colorization, since for instance, a flower can be red, yellow, pink, and so forth.

Hence, as a solution, in many cases the goal is not necessarily to recover the actual ground truth color, but rather to produce a plausible colorization that could potentially fool a human observer. Looking at the problem with this lens makes this task much more achievable.

In fact, one can model enough of the statistical dependencies between the semantics and the textures of grayscale images and their color versions in order to produce visually compelling results. [R.21]

5.1 Method One[1]

To satisfy the goals of colorization, an encoder-decoder structure is implemented which gets an image grayscale format as its input and outputs a colorful image. As was discussed in chapter 3 section 3.2.1, other than representing images in RGB

[1] [R.21]

format, one can represent them in another space as LAB. (In that section, the details behind this space was discussed thoroughly, so here we only employ them to walk through this chapter's goal.) In fact, for the sake of the described network, images represented in L space are fed into the network, and it is responsible to figure out the two other a and b parts. Therefore, with all these three segmentations of the LAB space, one can have a colorful image.

As any other networks, in order to train colorization network, one needs to define a loss function. In the case of this problem, to define such function, it is essential to pay attention to what function it is. For instance, some chose L_2, meaning that the distance between the predicted image's channels and the original image is calculated. As appealing it seems, due to the fact that this problem is a multi-modal one[2], this norm is not a robust choice.[3]

To tackle this obstacle, in [R.21] multinomial cross-entropy loss is suggested. With the help of this loss, for a specific object, the possibility of various colors can be checked and the best one can be selected.[4] Method one's structure is depicted in Fig.5.2:

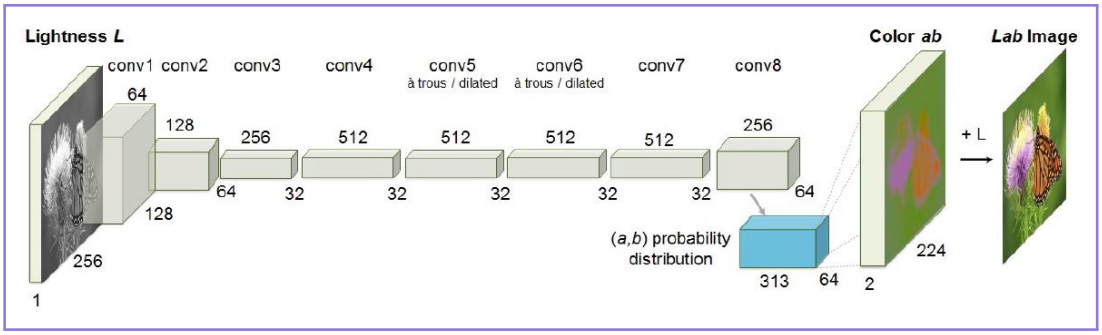

FIGURE 5.2: Method one's architecture.
Each conv layer refers to a block of 2 or 3 repeated conv and ReLU layers, followed by a BatchNorm layer. The net has no pool layers. All changes in resolution are achieved through spatial down-sampling or up-sampling between conv blocks.

5.2 Method Two[5]

In another paper, [R.22] human intevention is added. In more details, the network asks people to color some specific points, then, with the guidance of these points, the other parts of the image is colored. Unlike other methods, in this procedure, U-Net is employed and for training, since human interaction cannot be added for many images, a new idea is implemented.

In this idea, same as the others, channel L is fed into the network, afterwards, based on the ground truth, it is masked randomley. For masking, about 80-90 percent of the image is hidden and the remaining part creates the input of the network. The remaining parts of the image which is not mask acts like the user-guided colorization.[6]

[2]The reason behind this fact is that each part of the image may have more than one choice for coloring, hence the multi-modal charactristics.

[3]In these cases L_2 takes the average of choices and colors the image with the average value. This results in not acceptable coloring.

[4]As it is clear, instead of solving a regression problem, with the help of this loss, a classification problem is answered.

[5][R.22]

[6]This idea is noted as "Peek" in this paper.

Besides all the mentioned differences, *Smooth* L_1 loss is used to train the network which also solves the problem of L_2 loss.

Method two's structure is depicted in Fig.5.3:

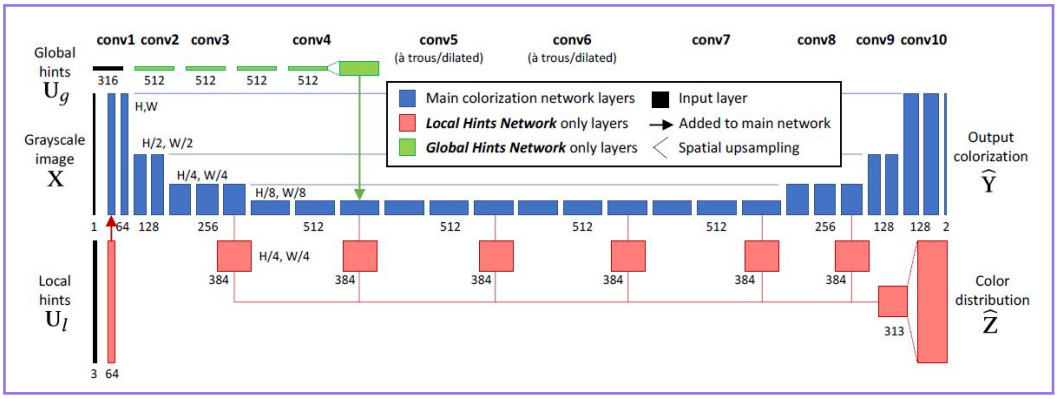

FIGURE 5.3: Method two's architecture.
Two variants of the user interaction colorization network are trained. Both variants use the blue layers for predicting a colorization. The Local Hints Network also uses red layers to (a) incorporate user points U_l and (b) predict a color distribution $\hat{Z}$.
The Global Hints Network uses the green layers, which transforms global input U_g by 1×1 conv layers, and adds the result into the main colorization network. Each box represents a conv layer, with vertical dimension indicating feature map spatial resolution, and horizontal dimension indicating number of channels.
Changes in resolution are achieved through sub-sampling and up-sampling operations. In the main network, when resolution is decreased, the number of feature channels are doubled. Shortcut connections are added to up-sampling convolution layers.

5.3 The Given Paper's Method

In our given paper, the same structure as [R.22] is implemented. However, there are some distinctive improvements. One of which is that instead of masking the ground-truth, it masks T_{ab}. In fact, it concatinates this map with the lightness of the image. (channel L)

Another difference is that it utilizes weighted *Smooth* L_1 loss. This is a notable idea since it can pay more attention to the pixels which are not masked. Therefore, loss is calculated as:

$$L_c = L_h((1 + \lambda M) \odot F_c(x), (1 + \lambda M) \odot y)$$

5.3.1 Network's Architecture

The architecture of this network shares features with U-Net, so it consists of ten blocks. The first four layers double the number of channels and halves H and W[7], while in the last four layers, the opposite procedure takes place. In layer number five and six, the stated dimensions are fixed, and dilated convolutional layers with factor two are used. Mostly, the convolutional layers are 3×3.[8]

[7] Resolutions

[8] The receptive field is not altered but more information is transfered through bottleneck

In addition to the mentioned symmetric layers, there are some short-cut connections as well.

This method's structure is depicted in Fig.5.4:

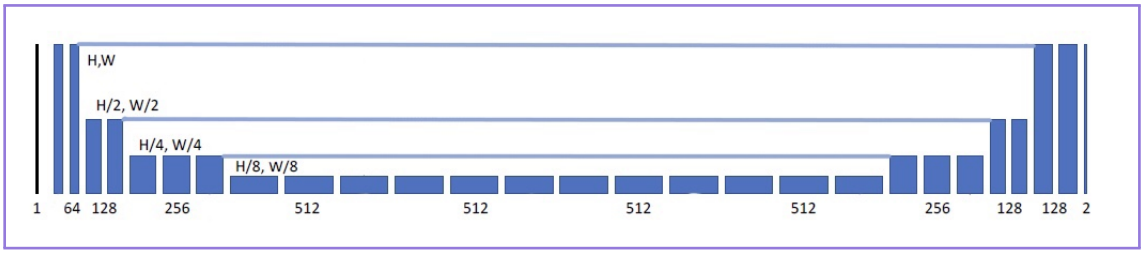

FIGURE 5.4: This method's structure. As described, it has the blue part of Fig.5.3.

In [R.22] it is stated that the first eight layers are pre-trained while the last two layer are trained based on the data that is desired. However, in our case, a complete pre-trained network is utilized. [R.23]

In order to alter the number of channels to two, a 1×1 convolutional layer is used.

5.3.2 Network's Input and Output

Input

The input has four channels:

- $\boldsymbol{L}$: L is the lightening channel representing the grayscale image.
- $\mathbf{T_{ab}}$: T_{ab} is a map which is the output of the transfer sub-net. It consists of colors and their places.
- **Masks**: Masks are a one-dimensional feature map which act as the human intervention.

Therefore, the result of concatinating them will be a four-dimensional input.

Output

As was mentioned earlier, the output is an image with two, ab channels.

Chapter 6

Training and Results

6.1 Transfer Sub-Net Training

As was discussed in chapter 4, this network has two main parts: encoder and decoder. In the same chapter under section 4.1, it is stated that the encoder module utilizes a pre-trained VGG19, hence there is no need to train such network. On the other hand, the decoder module needs training.

6.1.1 Loss

In [R.24] it is stated that the goal is to train and encoder-decoder network so that it can perform style transfer. The desired output is the one that has the content of the original content image with the style of the style image. Therefore, content, style, and output of the encoder network are fed into a pre-trained VGG19 network separately. Fig.6.1 represents the network implemented in this paper.

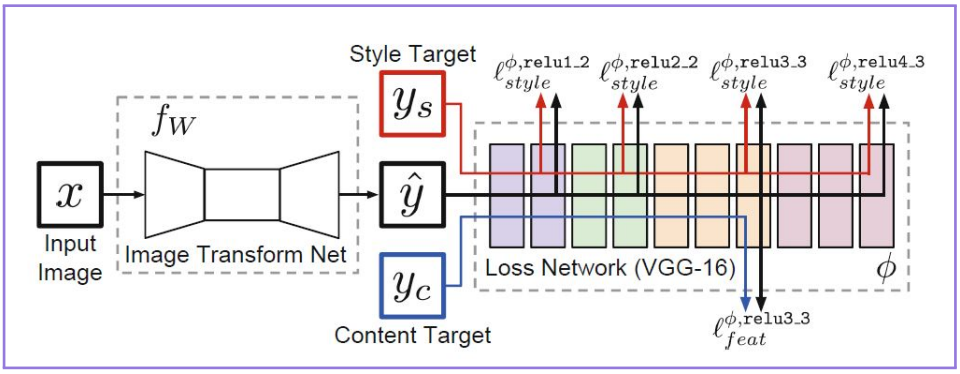

FIGURE 6.1: System overview.
An image transformation network is trained to transform input images into output images. A loss network pre-trained for image classification is employed to define perceptual loss functions that measure perceptual differences in content and style between images. The loss network remains fixed during the training process.[R.24]

Perceptual Loss

Feature Reconstruction Loss: Rather than encouraging the pixels of the output of encoder image to exactly match the pixels of the target image, here they are encouraged to have similar feature representations as computed by the pre-trained VGG19 loss network. Hence, the feature reconstruction loss is the (squared, normalized) Euclidean distance between feature representations of the output of encoder and the target image.

Mathematically speaking:

$$l_{feat}^{\phi,\,j}(\hat{y}, y) = \frac{1}{C_j H_j W_j} ||\phi_j(\hat{y}) - \phi_j(y)||_2^2$$

where ϕ is the pre-trained VGG19 loss network, $\hat{y}$ is the output image[1], y is the target image, in the j^{th} layer, and one can extract a $C_j \times H_j \times W_j$ feature map.[2]

Hence, using a feature reconstruction loss for training the image transformation network encourages the output image to be perceptually similar to the target image, but does not force them to match exactly.

Style Reconstruction Loss: The feature reconstruction loss penalizes the output image $\hat{y}$ when it deviates in content from the target y. Plus, it is desired to penalize differences in style: colors, textures, common patterns, etc. To achieve this goal, *Style Reconstruction Loss* is defined.

$$l_{style}^{\phi,\,j}(\hat{y}, y) = ||Gj^{\phi}(\hat{y}) - Gj^{\phi}(y)||_F^2$$

where

$$G_j^{\phi}(x)_{c,c'} = \frac{1}{C_j H_j W_j} \sum_{h=1}^{H_j} \sum_{w=1}^{w_j} \phi_j(x)_{h,w,c} \phi_j(x)_{h,w,c'}$$

is *Gram matrix*. In fact, if we interpret $\phi_j(x)$ as giving C_j-dimensional features for each point on a $H_j \times W_j$ grid, then $G_j^{\phi}(x)$ is proportional to the uncentered covariance of the C_j-dimensional features, treating each grid location as an independent sample. It thus captures information about which features tend to activate together.

The style reconstruction loss is well-defined even when $\hat{y}$ and y have different sizes, since their Gram matrices will both have the same shape. Generating an image $\hat{y}$ that minimizes the style reconstruction loss preserves stylistic features from the target image, but does not preserve its spatial structure. Reconstructing from higher layers transfers larger-scale structure from the target image.

Hence, to calculate "Perceptual Loss", the weighted sum of both stated losses are participated. The weights are selected with experiments. Mathematically speaking:

$$Perceptual\ Loss = Content\ Coeff * Content\ Loss + Style\ Coeff * Style\ Loss$$

Reconstruction Loss

To calculate the reconstruction loss, the L_2 norm of the target which is the output of the network as well as the content is calculated.

Therefore, the total loss can be formulated as:

$$Total\ Loss = Perceptual\ Coeff * Perceptual\ Loss + Reconstruction\ Coeff * Reconstruction\ Loss$$

[1] $\hat{y} = f_W(x)$

[2] In [R.24], optimization is used to to find an image $\hat{y}$ that minimizes the feature reconstruction loss $l_{feat}^{\phi,\,j}(\hat{y}, y)$ for several layers j from the pre-trained VGG16 loss network ϕ. It is worth mentioning that while reconstructing from higher layers, image content and overall spatial structure are preserved, but color, texture, and exact shape are not.

6.1.2 Optimizer

ADAM optimizer is utilized to minimize the described loss while its learning rate is set as 0.001.

6.1.3 Other Hyper-Parameters

- **Batch Size**: 4
- **Number of Epochs**: 1
- **Style Weights**:
  - *conv*1_1: 1.0
  - *conv*2_1: 0.75
  - *conv*3_1: 0.2
  - *conv*4_1: 0.2
  - *conv*5_1: 0.2
- **Perceptual Coefficient**: 0.2
- **Reconstruction Coefficient**: 0.8
- **Content Coefficient**: 1
- **Style Coefficient**: 10^6

6.2 Colorization Sub-Net Training

As was discussed in chapter 5 section 5.3.1, a pre-trained network on ImageNet is utilized. The weights and network implementation were downloaded from [R.23]. However, after putting the transfer sub-net and this network together, it is fine-tuned with COCO dataset.

One other point which is essential is that this network is used as prior coloring for transfer sub-net. Hence, to force the network to learn the prior colorization task, once in a while, T_{ab} is set as none.

6.3 Total Network Training

6.3.1 Loss

For the total network, as was elaborated, *Smooth* L_1 loss is employed. To implement it with Pytorch, "nn.HuberLoss" is utilized as criterion while its reduction is set as mean and its delta hyper-parameter is set as 1.0.

6.3.2 Optimizer

ADAM optimizer is utilized to minimize the described loss while its learning rate is set as 0.00001.

6.3.3 Other Hyper-Parameters

- **Batch Size**: 4
- **Number of Epochs**: 1
- **Mask Probability**: 0.99
- **Lambda**: 10

6.4 Results

6.4.1 Transfer Sub-Net Results

Based on the above explanations regarding the parameters for this network, the network is trained. The results are depicted in Fig.6.2.

FIGURE 6.2: Results of transfer sub-net on COCO dataset

As can be seen, we need another network to refine the color. Hence, the role of colorization network becomes clear.

Besides COCO dataset, we also ran the nework on CamVid dataset The results are depicted in Fig.6.3.

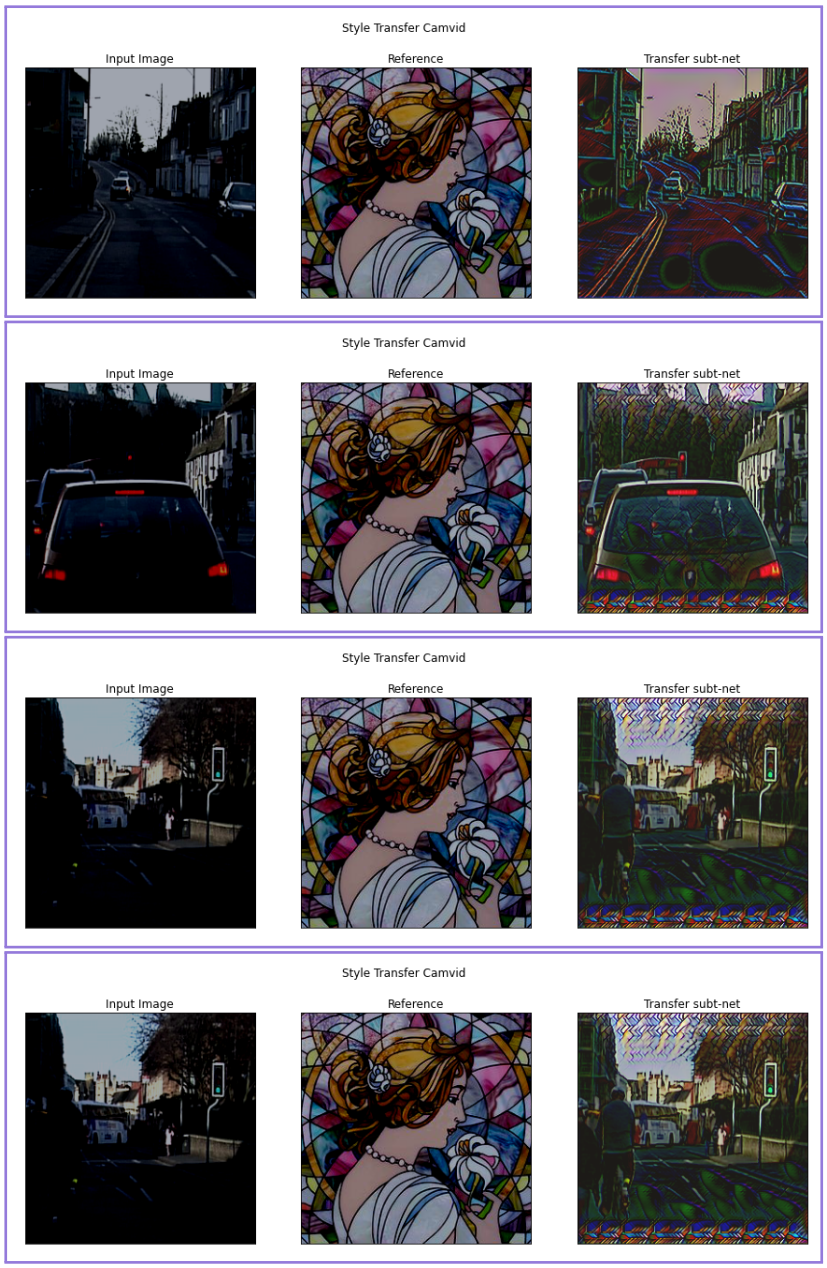

FIGURE 6.3: Results of transfer sub-net on CamVid dataset

6.4.2 Total Network Results

Based on the above explanations regarding the parameters for this network, the network is trained. The results are depicted in Fig.6.4.

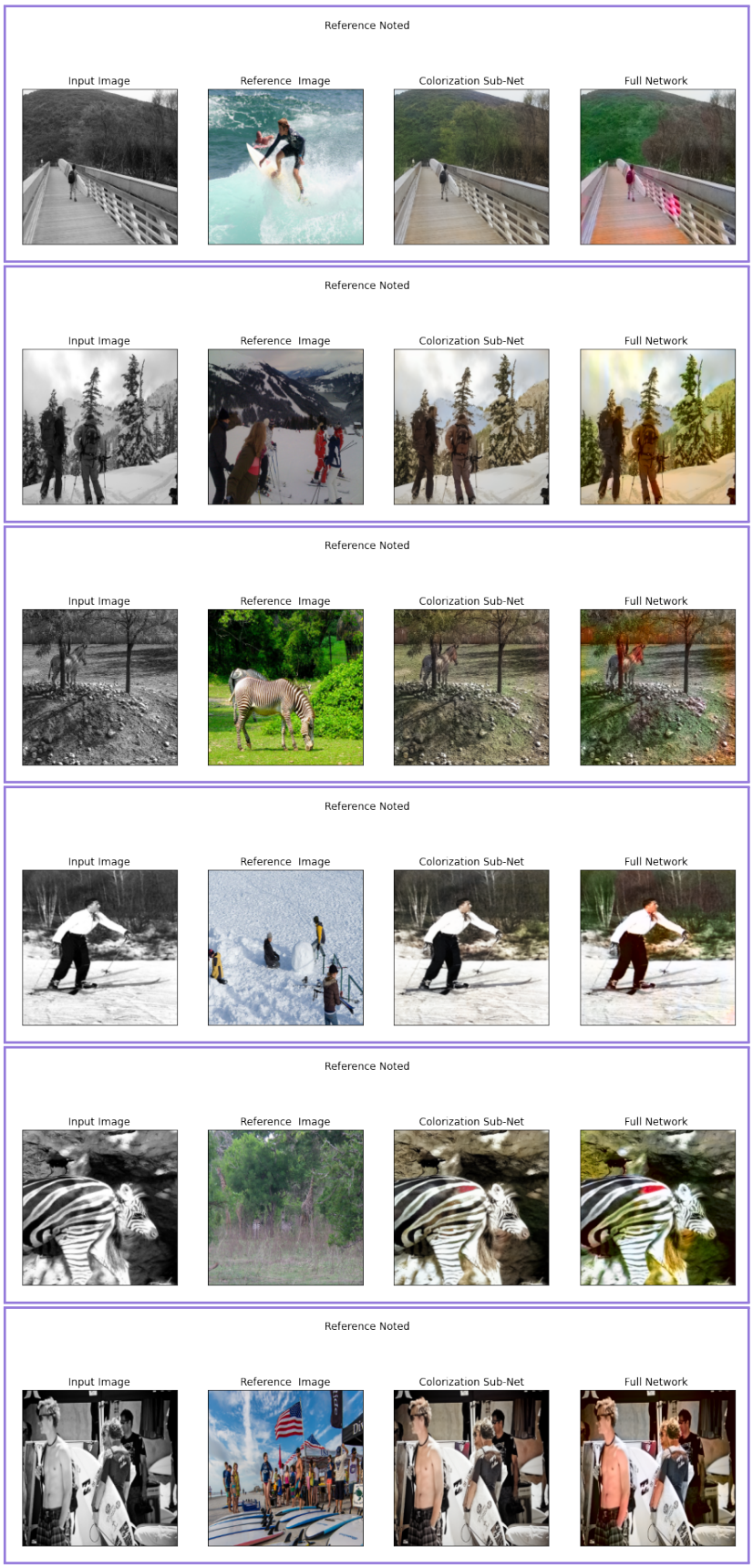

FIGURE 6.4: Results of total network on COCO dataset

Although there are still some problems in the colorization which are a result of

our limited sources[3], the results are encouraging. Fig.6.5 and 6.6 are indicators for the stated statement.

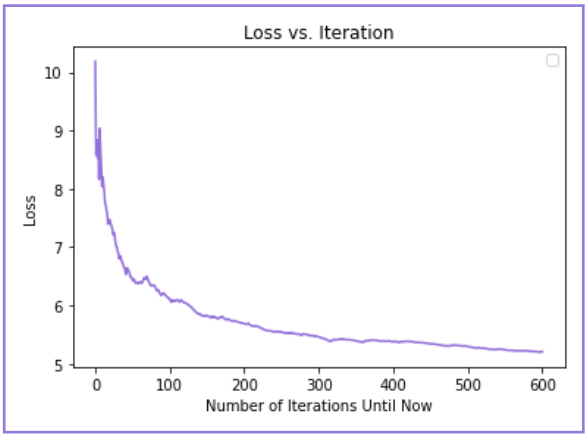

FIGURE 6.5: Training loss of transfer sub-net on COCO dataset

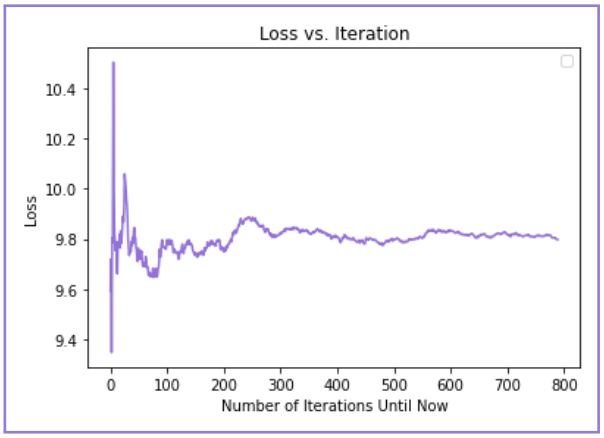

FIGURE 6.6: Training loss of total network on COCO dataset

[3]like removing AdaIn in the structure of encoder and decoder of transfer sub-net, training on a segmentation of COCO dataset, and so forth

Chapter 7

Reference

R.1 COCO Dataset

R.2 An Introduction to the COCO Dataset

R.3 Microsoft COCO: Common Objects in Context

R.4 Getting Started with COCO Dataset

R.5 Working with Color Images in Python

R.6 Convert an image RGB->Lab with python

R.7 Neural Style Transfer in Pytorch

R.8 Fast and Arbitrary Style Transfer

R.9 Image Style Transfer Using Convolutional Neural Networks

R.10 Style Transfer using Deep Neural Network and PyTorch

R.11 Why Does Neural Style Transfer Work Best With Old VGG CNNs' Features?

R.12 Understanding the VGG19 Architecture

R.13 Source Code for torchvision.models.vgg

R.14 Torchvision Models

R.15 Best way to get at intermediate layers in VGG and ResNet

R.16 Replacing a max pooling layer with an average pooling layer on a VGG model

R.17 Slicing VGG19

R.18 Adaptive Instance Normalization

R.19 AdaIN Usage in Network

R.20 AdaIN Implementation

R.21 Colorful Image Colorization

R.22 Real-Time User-Guided Image Colorization with Learned Deep Priors

R.23 Colorization Network Weights

R.24 Perceptual Losses for Real-Time Style Transfer and Super-Resolution