Machine Learning for Image Colorization

Ligeng Zhu*, **Zeyu Zhao*** **and Chong Guo***
School of Computing Science
Simon Fraser University
(* denotes equal contribution)
{lykenz, zza100, armourg}@sfu.ca

Abstract

Given a gray-scale image as input, we experimented using SVR and 3 different deep neural networks to colorize it. We trained and tested our models on Microsoft COCO dataset [1] and the deep neural networks show a competitive result while taking reasonable training time. Among all approaches we tried, the **Stylization Network** gives the best performance in COCO dataset and we studied whether it can be further improved by using different numbers of *conv* layers and different training parameters. We enhanced two of our models on Places365-Standard dataset [2], which includes 1.6M images and the fine-tuned models can colorize images with the same level of visual quality compared to the state-of-the-art approaches.

1 Introduction

Automated image colorization can be utilized to restore old photos and enhance gray-scale videos. Recently, this topic has become popular in machine learning communities [3, 4, 5]. In our project, we consider the image colorization problem as a task to predict UV values given Y values in YUV color space. However, the colorization problem is much more involved compared to the regression problems we have seen in the lectures since no clear features can be utilized to train models.

Our first thought was to use support vector machine regression (SVR). However, this approach doesn't give a satisfying result. Therefore, we started to explore previous researches about image colorization and found out that nearly all research on this topic utilizes deep neural networks. We choose and implement Residual Encoder Network [5] and Multi-level Feature Network [3], not surprisingly, they show much better results compared to SVR.

Earlier this year, Johnson et al. [6] proposed a deep neural network for image stylization task. Intuitively, our colorization problem can be viewed as a type of stylization task since colors are mapped to pixels according to their features (Fig. 1). In a high level, both stylization and colorization tasks can be taken as image-to-image pixel-wise prediction. So we think this stylization network is also effective for the image colorization task with some modifications. There are some previous research [4] utilizing similar network structure, but best to our knowledge, we are the **first group** adapting Stylization Network [6] for image colorization. In addition, we tried different numbers of *conv* blocks and different training batch sizes to see whether this approach can be further improved.

Since the Microsoft COCO dataset [1] lacks diversity, we later enhance our multi-level feature network and stylization network on Places365-Standard dataset [2], which includes 1.6 million images. The fine-tuned models can render results as good as that of the state-of-the-art models [4]. These two deep neural networks are also convolutional layer oriented, which means they can handle images of any size.

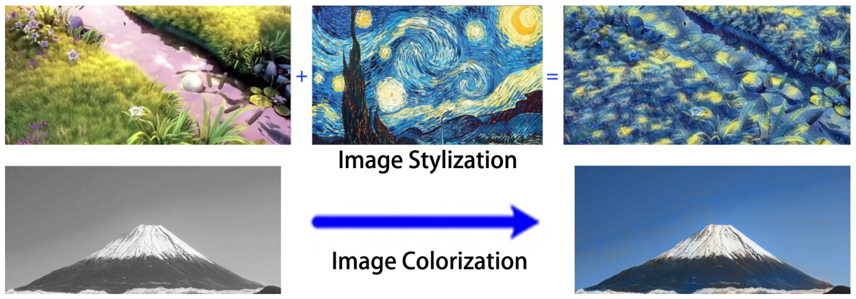

Figure 1: Stylization task and colorization task

Our source code and pre-trained models for all aforementioned approaches are published on GitHub. Readers can download our code and pre-trained models to colorize their favorite images or train their own models. The URL is:

`https://github.com/Lyken17/Image-Colorization`

Our main work in this project are: (1) Came up with and implemented HOG + SVR approach. (2) Implemented the residual encoder network [5] and the multi-level feature network [3]. (3) Adapted stylization network [6] for image colorization task. (4) Trained and tested all aforementioned models in Microsoft COCO dataset [1] and compared their performances. (5) Tried different numbers of "shave blocks" and different training batch sizes for stylization network to see whether the result can be further improved. (6) Fine-tuned both multi-level feature network and stylization network in the Places365-Standard dataset [2].

The rest of this report is organized as follows: Section 2 introduces different approaches and network structures we utilize. Section 3 provides our experiment details and the result analysis. Conclusions and discussions are given in Section 4. Finally, Section 5 lists contributions of individual members.

2 Approach

All our approaches try to predict UV values of a given gray-scale image. We define *UV loss* for a $W \times H$ image as:

$$\mathcal{L}_{UV} = \frac{\sum_{(i,j,k)\in\mathbf{UV}} \left(\mathbf{UV}^{predict}(i,j,k) - \mathbf{UV}^{true}(i,j,k)\right)^2}{2 \times W \times H}$$

where $\mathbf{UV}$ is the matrix containing UV values for all pixels and $size(\mathbf{UV}) = 2 \times W \times H$.

Our goal is to minimize $\mathcal{L}_{UV}$. For all deep neural networks, since the U value and the V value are within the range $(-1, 1)$. We utilize $Tanh$ as the activation function for the output layer:

$$h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \in (-1, 1)$$

Values of U and V are further normalized to exactly $[-1, 1]$. It's fine that $h(x)$ only becomes ± 1 when $x = \pm\infty$ since it is rare for U and V to take extreme values. We let the networks predict U' and V' (normalized UV values) instead of U and V. The loss function is also changed to $\mathcal{L}_{U'V'}$ at training time.

Next, we will introduce all five approaches we tried. We compare and rank these methods according to their testing errors (i.e. average $\mathcal{L}_{UV}$ on testing dataset).

2.1 HOG + SVR

An intuitive approach for this task is predicting U and V values pixel-by-pixel. However, U and V values for a certain pixel are dependent on the region it belongs to. To consider the local features near the pixel for which we want to predict U and V values, we introduce the HOG [7] feature. For each pixel, we extract a 16×16 image block around this pixel and compute the HOG for this image block. In our experiment, we divide the 16×16 image block into 4 cell-blocks and the number of bins for each histogram is 9. Therefore, a total number of $4 \times 9 = 36$ values can be obtained as a descriptor for this image block. Fig. 2 shows an example of HOG extraction.

Support vector machine regression (SVR) is then used to predict the U value and V value. For each pixel, we construct a 37-dimensional feature vector (Y value and HOG feature), and then U-model is utilized to predict U value and V-model is utilized to predict V value.

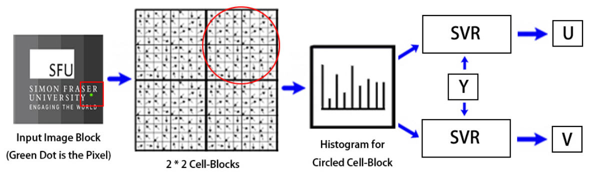

Figure 2: HOG + SVR Colorization

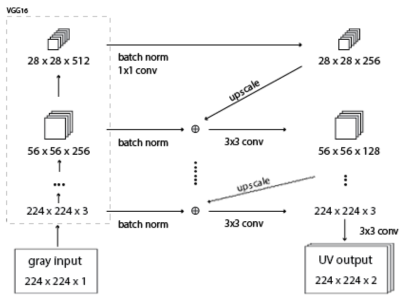

Figure 3: Residual Encoder Network

2.2 Residual Encoder Network

The residual encoder network has an encode-decoder-like structure with residual connections (Fig. 3). The first half structure is mainly adopted from VGG-16 [8], and in the second half, upscaled feature maps are combined to fuse information. The network keeps doing so until a $224 \times 224 \times 2$ feature map is obtained, which is exactly the UV values we want to predict.

2.3 Multi-level Feature Network

This network [3] is end-to-end utilizing both local and global features, with no limit on input image size. After compressing feature maps, Multi-level feature network later recovers them by fusing both global and local information. This network is mainly adopted from [3] with several modifications to significantly reduce training time while keeping the same level of performance. The network structure is shown in Fig. 4 and Fig. 6.

(a) Feature Extraction

type	kernel	stride	output
conv	3*3	2*2	64
conv	3*3	1*1	128
conv	3*3	2*2	128
conv	3*3	1*1	256
conv	3*3	2*2	256
conv	3*3	1*1	512

(b) Global Feature

type	kernel	stride	output
conv	3*3	2*2	512
conv	3*3	1*1	512
conv	3*3	2*2	512
conv	3*3	1*1	512
FC	-	-	1024
FC	-	-	512
FC	-	-	256

(c) Local Feature

type	kernel	stride	output
conv	3*3	1*1	512
conv	3*3	1*1	256

(d) Fusion Layer

type	kernel	stride	output
copy	-	-	256
fuse	-	-	512

(e) Conv and upsample

type	kernel	stride	output
conv	3*3	1*1	256
conv	3*3	1*1	128
up	-	-	128
conv	3*3	1*1	64
conv	3*3	1*1	64
up	-	-	64
conv	3*3	1*1	32
conv	3*3	1*1	16
up	-	-	16
conv	3*3	1*1	8
conv	3*3	1*1	2
up	-	-	2
Tanh	-	-	2

Figure 4: Details of Multi-level Feature Network

Input: Given one gray scale image, this network resizes its width and height to the multiple of 8 (in order to fit the size of convolutional layer) and then downscales it to 224×224 (*Scaled*) while keeping a copy of original image (*Origin*).

Feature Extraction: We feed *Scaled* and *Origin* to *Feature Extraction*, and then get *Local* and *Global* respectively (Weights and gradients are shared). Different from the end-to-end, one-step training of the original approach [3], we first train this part separately and then use the pre-trained weights as initial weights for faster convergence (fine-tune).

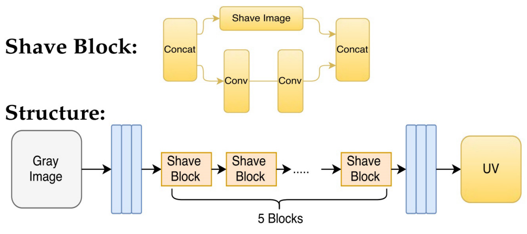

Figure 5: Stylization Network

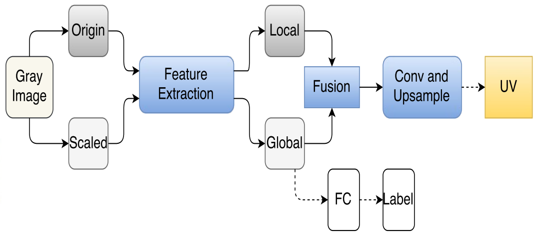

Figure 6: Multi-level Feature Network

Local features: In order to enforce *Feature-Extraction* to extract the proper features, a simultaneous classification procedure is attached after *Global*. Note this part is only required at training phase. When deploying the model, we don't need the classification procedure.

Global Features: The local features are obtained by further processing the extracted features from *Scaled* with two convolutional layers. Those 256 feature maps are later mapped into 256 numbers by 3 fully-connected layers, and replicated to $\frac{w}{8} * \frac{h}{8} * 256$ for later fusion.

Convolution and Upsample: Once global and local features are fused, they are later processed by convolutional and upsampling layers. Details of convolutional layers are shown in Fig. 4 and for upsampling, we use nearest neighbors. These layers are alternated until the output has the same size as *Origin* (different from half size as in [3]).

2.4 Stylization Network

The stylization network [6] is originally used to transfer artistic styles from a painting to an image (as explained in Fig. 1). It shows great effectiveness in stylization task, and we notice that its fully convolutional nature fits colorization task as well. Therefore we adapt this network for image colorization task. The structure of the stylization network is shown in Fig. 5 (details in Fig. 7). For colorization problem, only Y values of an image are known and therefore, we change the input channel dimension to 1. Similarly, output channel is set to 2 instead of 3.

(a) Encode

type	kernel	stride	output
conv	9*9	1*1	32
conv	3*3	3*3	64
conv	3*3	3*3	128

(b) Shave Block

type	kernel	stride	output
conv	3*3	1*1	128
conv	3*3	1*1	128
Shave	-	-	128
Concat	-	-	128

(c) Decode

type	kernel	stride	output
conv	3*3	2*2	64
conv	3*3	2*2	32
conv	9*9	1*1	2
Tanh	-	-	2

Figure 7: Details of Stylization Network

Shave Blocks: The Shave Blocks contain two *conv* layers and a *Shave* layer. *Shave* layer is similar to *identity* layer but the output is re-scaled to match the size of the output of two *conv* layers. The design of shave blocks allows the network to "skip" some *conv* layers.

Output: After the last Shave block, the feature maps are enlarged to the size of the input images via the last three *conv* blocks. We changed the activation function of the output layer to $Tanh$ as explained in the beginning of this section.

3 Experiments

Except the fine-tuning part, all our models are trained in Microsoft COCO 2014 training dataset [1]. An "epoch" contains 82783 images (all training images in COCO dataset) and at every "iteration", we feed "batch size" images into the networks. All our deep neural networks are trained on NVIDIA TITAN X GPUs.

3.1 HOG + SVR

We extract HOG features as explained in Section 2.1 and use linear kernel for support vector regression. After training the model in COCO dataset, we observed that all weights w are near 0 while the bias b is nearly the average of UV values of all images in the training data. Therefore, we conclude that this model doesn't work. Due to the time limitation, we didn't test other kernels or parameters before switching to deep learning.

3.2 Residual Encoder Network

As mentioned in Section 2.2, we resize the input images to 224×224 to fit the VGG16 [8] model. The learning rate is set to be 10^{-6} and the optimizer is SGD. It takes around 32 hours to train this network on COCO dataset with batch size set to be 30.

3.3 Multi-level Feature Network

Compared with other two deep neural networks we experimented, this network is far more complicated. As mentioned in [3], it takes 3 weeks to train this network on NVIDIA TESLA K80, which is far beyond our resources. In order to train it in reasonable time without the loss of performance, we first train the classification part on ImageNet [9] for 34 epochs on 3 Titan X with learning rate $\eta = 0.1$ and momentum optimizer. The pre-trained weights (for convolutional layers only) are taken as initial weights for our colorization task. This trick makes training much faster and the rest of the training phase can be done within 3 days on a single Titan X.

We first train the network in COCO dataset for 30 epochs. We randomly shuffle these images and set the batch size to 32 (maximal batch size possible for a single Titan X), with initial learning rate to be 10^{-3}. The learning rate is reduced by half for every 3000 iterations and the Adam optimizer is used. Fig. 8 shows the training loss for the first 60000 iterations. In later training, we use Places365-Standard dataset [2], which consists of 1.6M training images. With the trick mentioned above, the whole training phase (7 epochs) for Places database can be completed in 3 days on a single Titan X.

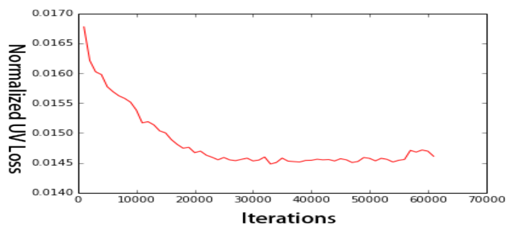

Figure 8: MLF Net Training Loss ($\mathcal{L}_{U'V'}$)

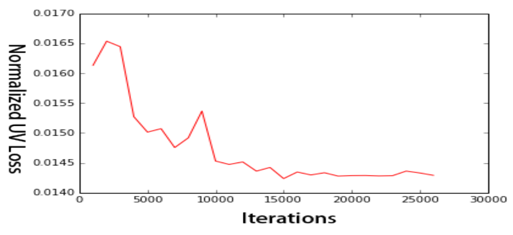

Figure 9: Stylization Net Training Loss ($\mathcal{L}_{U'V'}$)

3.4 Stylization Network

Neural networks for the stylization task are harder to converge at the training phase. So Johnson [6] set a tiny batch size. However, for colorization problem, a small batch size causes problem (converge in local minima) so we should increase the batch size. We also tuned the learning rate for training and introduced the learning rate decay function, where we reduce the learning rate by half every 3000 iterations.

We make all changes to the network as explained in Section 2.3 and train the network for 30 epochs on COCO dataset (with Adam optimizer). The learning rate is set to be 10^{-3} and every 3000 iterations, we reduce the learning rate by half (i.e. $\eta' = 0.5 * \eta$). We set the batch size to 30 and this is the largest batch size that our GPU can handle. It takes around 16 hours to train this network on COCO dataset. The training loss for the first 26000 iterations is shown in Fig. 9.

After training on Microsoft COCO dataset, we further fine-tune the model on Places365-Standard [2] dataset. We load the model trained in COCO dataset and further train it in Places dataset for around 3 days (around 8 epochs) with learning rate fixed to 10^{-7}.

3.5 Model Comparison

We compare all our models trained in Microsoft COCO dataset. Since all deep neural networks are trained for 30 epochs. It is fair to compare those models. We test all models in COCO 2014 validation dataset (images scaled to 256×256) and the testing errors (average $\mathcal{L}_{UV}$ on testing dataset) are listed in Table 1.

Table 1: Testing error on Microsoft COCO dataset

Gray Output	**HOG + SVR**	**Residual Encoder Net**	**Stylization Net**	**MLF Net**
0.005163	0.004787	0.003956	**0.003758**	0.003782

3.6 Different Numbers of Shave Blocks and Training Batch Sizes for Stylization Network

As shown in Table 1, the Stylization Network gives the best performance in COCO dataset. So we further studied this network structure. We tried different numbers of "Shave blocks" (each shave block has 2 *conv* layers, see Fig. 5) and different training batch sizes. Here, due to time and resource limitation, we only run 15 epochs. Table 2(a) shows how different numbers of shave blocks (*conv* layers) impact the testing error (Batch Size = 25, Epochs = 15). Table 2(b) shows how different training batch sizes impact the testing error (Number of Shave Blocks = 5, Epochs = 15).

Table 2: Testing errors for different numbers of Shave Blocks and different batch sizes

(a) Different numbers of "Shave Blocks"

3	**4**	**5**	**6**
0.003979	0.003937	0.003937	0.003943

(b) Different batch sizes

10	**20**	**25**
0.004069	0.003982	0.003937

3.7 Analysis

We learned from textbook that deep neural networks are powerful on complicated problems where relevant features are hard to extract. In the colorization problem, we tried both SVR and deep learning and as shown in Table 1, deep neural networks greatly outperform the SVR approach.

Fig. 8 and Fig. 9 show both of our networks converge hence our training is correct.

Table 1 shows **Stylization Network** gives best performance. It makes sense since this network structure is more complicated than the residual encoder network and we simplified the multi-level feature network which might have negative impact on its performance.

Table 2(a) shows that more *conv* layers usually promise better results (networks with 4 and 5 shave blocks give the same testing error, but the network with 5 shave blocks gives better colorization result). However, the network with 6 shave blocks gives worse testing error compared to the network with 5 shave blocks. The reason might be that the network with 6 shave blocks is harder to converge.

Table 2(b) shows that larger batch size promises better training result and this observation is consistent with previous research. So when training the stylization networks, we should use batch sizes as large as possible.

3.8 Colorization Examples

To show our models work, we downloaded an arbitrary image from Google (SFU AQ) and the colorization result is shown in Fig. 10.

4 Conclusions and Discussions

In summary, deep neural networks outperform the SVR approach and the residual network [5] is the earliest network for the colorization task. Due to the simple structure, it always mis-predicts UV in

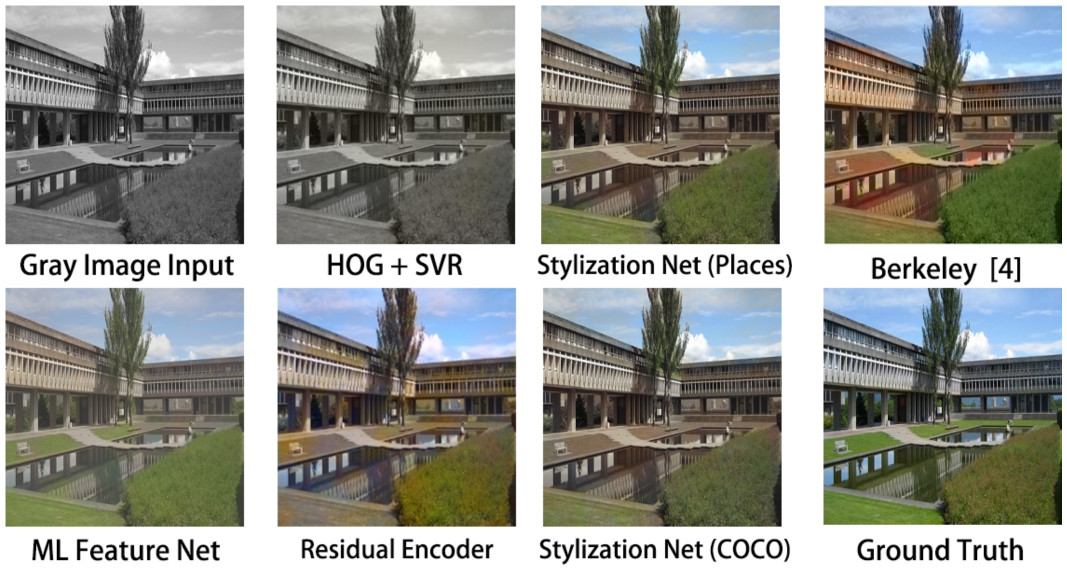

Figure 10: Colorization Example

complicated scenes. The multi-level feature network [3] can obtain rich information of colors and labels, and it is thought to be the state-of-the-art. But in our experiment, it doesn't give an obvious improvement compared to the Stylization Network. It may be caused by inadequate training (3 days on Titan X V.S 3 weeks on K80). We are the **first group** to adapt the stylization network for image colorization task, and surprisingly, even without additional label data, it shows a competitive result compared to the multi-level feature network [3].

The stylization network might be further improved by replacing Batch Normalization with Instant Normalization [10]. As shown in [10], Instant Normalization greatly improves the performance for image stylization task, which indicates that the Instant Normalization might also be effective for the colorization task.

Another thing we noticed during experiments is the loss. Though $\mathcal{L}_{UV}$ of 3 deep neural networks are close to each other, their performance on validation images varies a lot (as judged by human). Even for the same model, after loss converges, the results can be significantly improved by more iterations of training. Therefore, We guess the L2 loss on YUV color space may not be a good measurement for the colorization task.

Due to the page limitation, we are unable to give more colorization examples or cover every detail of our models and experiments. We therefore strongly recommend readers to visit our GitHub repository `https://github.com/Lyken17/Image-Colorization` where more details are provided and pre-trained models are released.

5 Contributions

In our project, **Ligeng Zhu** implemented, trained, tested and fine-tuned the multi-level feature network. He also set up the *Torch* environment in the GPU server for the whole group. **Zeyu Zhao** adapted, trained, tested and fine-tuned the stylization network. He also came up with and implemented the HOG + SVR approach. **Chong Guo** implemented, trained and tested the residual encoder network. He also set up the *Tensorflow* environment in the GPU server for the whole group. **Ligeng Zhu** and **Zeyu Zhao** made the poster for presentation and **Chong Guo** printed it. **All members** joined regular group meetings and wrote their corresponding parts in this report.

References

[1] Lin, T.Y. et al. (2014) Microsoft COCO: Common objects in context. *In ECCV* pp 740–755

[2] Zhou, B. et al. (2016) Places: An Image Database for Deep Scene Understanding. *arXiv:1610.02055*

[3] Satoshi, I., Simo-Serra, E. & Ishikawa, H. (2016) Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification. *In SIGGRAPH* **35(4)**

[4] Zhang, R., Isola, P. & Efros, A.A. (2016) Colorful image colorization. *in ECCV*

[5] Dahl, R. (2014) Automatic image colorization. `http://tinyclouds.org/colorize/`

[6] Johnson, J., Alahi, A. & Fei-Fei, L. (2016) Perceptual losses for real-time style transfer and super-resolution. *In ECCV*

[7] Dalal, N. & Triggs, B. (2005) Histograms of oriented gradients for human detection. *In CVPR*, pp 886–893.

[8] K. Simonyan & A. Zisserman (2016) Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556*

[9] Russakovsky, O. et al. (2015) ImageNet Large Scale Visual Recognition Challenge. *In International Journal of Computer Vision* **115(3)**

[10] Ulyanov, D., Vedaldi, A. & Lempitsky, V. (2016) Instance Normalization: The Missing Ingredient for Fast Stylization. *arXiv:1607.08022*