Universal Style Transfer via Feature Transforms

E4040.2018Fall.HSYG.report
Hanying Gan hg2470, Shu Liu sl4234, Yiqing Su ys3060
Columbia University

Abstract

In this project, we experiment a universal style transfer method proposed for transferring arbitrary visual styles to content images. We conduct an Encoder-Decoder training on Microsoft COCO dataset. Afterwards, we apply the whitening and coloring methods introduced in the paper for style transfer. Experimental results demonstrate our performance. And we demonstrate the effectiveness of our algorithm by generating high-quality style transferred images with comparisons the results in the original paper.

1. Introduction

Style transfer is an essential task in computer vision with wide applicability in image editing, reconstruction, and generation. The key challenge for is task is to represent styles via disentangle features. Although Gram matrix[3] or feed-forward methods[4] have demonstrated their effectiveness, how to produce a generalized and efficient style transfer algorithm is still an open question.

To remedy this issue, paper [2], introduce a universal style transfer method for proper and effective style feature extraction and transfer. In this work we implemented the proposed method in the paper to unfold the image generation process by training an auto-encoder for image reconstruction, and integrate the whitening and coloring transforms (WCT) in the feed-forward passes to match the statistical distribution and correlations between the intermediate features of content and style.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

In this paper, a decoder network is firstly pre-trained to invert different levels of VGG features. Then it performs a style transfer through WCT based on both VGG and the pre-trained decoder. During this step, WCT is applied to one layer of content features let the covariance matrix matches that of style features. After the complementation of single-level stylization, the same method is further extended to multi-level stylization to match the statistics of all level styles. A control parameter is also introduced to represent the style transfer degree to enable a free choice of the balance between stylization and content preservation for users.

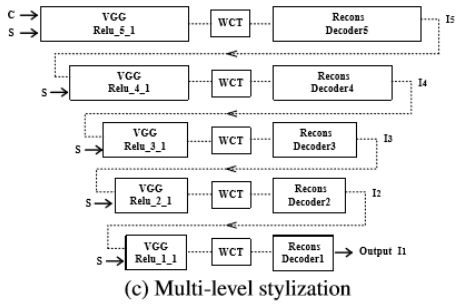

Fig 1. Method process flow chart in the paper

2.2 Key Results of the Original Paper

The original paper[2] implement a single and multi-level stylization pipeline and the stylization framework for texture analysis. The experiment results show a good performance in generalizing to arbitrary styles. In comparison with other methodologies, the approach adopted in this paper has received quantitative results with a covariance matrix difference (L_s) equals 6.3 and preference% equals 30.3, which outperforms other methods.

3. Methodology

In this section, we introduce methods for reproducing the multi-level stylization in Microsoft COCO. We first present the objectives in this work. Afterwards, main obstacles are introduced. Finally, we show the formulation and project design for our work.

3.1. Objectives and Technical Challenges

The objectives of our work are to reproduce the multi-level coarse-to-fine stylization presented in the paper. Given features from both lower and higher layers, we aim at capturing the characteristics of target style.

Our major formulation of multi-level stylization is composed of an image reconstruction process with whitening and coloring feature transformation. We firstly construct encoders and five decoders based on the pretrained layers in VGG 19 model. Then WCT is applied on Relu_5_1 features to gain a rough stylized result and is put as new content images to update features in lower layers.

There are several technical challenges that we need to address during our implementation process:

1. The first layer of decoder in the reconstruction model is insufficient in non-linearity, which leads to weakness in capturing features.
2. Encoder-Decoder model requires deep network structure for satisfaction results, which increase the complexity in the landscape of loss.

For first challenge, we improve the result of decoder 1 by training through hundreds of thousand iterations. As for the initial poor performance of decoder 5, we have experimented with losses of different weights, and finally employ the sum of feature loss, pixel loss and variance loss as the total loss to obtain a better result.

3.2. Problem Formulation and Design

The pixel reconstruction loss and feature loss are used for reconstructing an input image. The formulation is as follows:

$$L = \|I_0 - I_i\|_2^2 + \lambda \| \emptyset(I_0) - \emptyset(I_i)\|_2^2,$$

where I_0 and I_i represent input and reconstructing output image, and $\emptyset$ is the encoder of VGG with extraction of Relu_X_1 features, and λ is the weights to balance the two losses. The decoder will be fixed and used as feature inverter after our training.

The next important step are whitening and coloring transforms process, which is aimed at adjusting vectorized VGG feature maps of content image f_c with respect to the statistics of that of style image f_s, where $f_c \in R^{C\times H_c\times W_c}$, and $f_s \in R^{C\times H_s\times W_s}$. For whitening transform, we conduct a linear transformation for f_c to obtain $\widetilde{fc}$ by,

$$\tilde{f}c = E_c \times D_c^{-1/2} \times E_c^T \times f_c,$$

to make the feature maps are uncorrelated. D^c is a diagonal matrix with eigenvalues of the covariance matrix $f_c f_c^T$, and E_c is the corresponding orthogonal matrix of eigenvectors, with $f_c f_c^T = E_c \times D_c \times E_c^T$.

For coloring transform, which is the inverse of the whitening step to linearly transform $\tilde{f}c$, we manipulate on f_s to obtain $\widetilde{fs}$ by,

$$\widetilde{fs} = E_s \times D_s^{-1/2} \times E_s^T \times \tilde{f}c,$$

where D^s is a diagonal matrix with eigenvalues of the covariance matrix $f_s f_s^T$, and E_s is the corresponding orthogonal matrix of eigenvectors. Then, $\tilde{f}cs$ was re-centered with the mean vector m_s of the style, $\tilde{f}cs = \tilde{f}cs + m_s$.

Finally, $\tilde{f}cs$ was blended with the content feature f_c, and then we feed it to the decoder to provide users with an adjustable balance of stylization effects:

$$\tilde{f}cs = \alpha \times \tilde{f}cs + (1-\alpha) \times f_c,$$

Where α represents the style weight for users to control the transfer effect.

The detailed engineering process of implementation encoder-decoder and style transfer are included in the following section 4.

4. Implementation

In this section, we introduce the pipeline and implementation process including the training of multiple decoders and style transfer. And we provide models and results in detail in Bitbucket[1].

4.1. Deep Learning Network

4.1.1 Decoder Training

The model is trained on the Microsoft COCO dataset. For the multi-level stylization approach, we use a pretrained VGG19 model as encoder, and train five reconstruction decoders separately for features encoded in VGG19 relu_X_1 (X=1,2,3,4,5) layer.

The structure of encoder and decoder X is symmetric, and the flow chart below is an example of reconstruction decoder X (X=2):

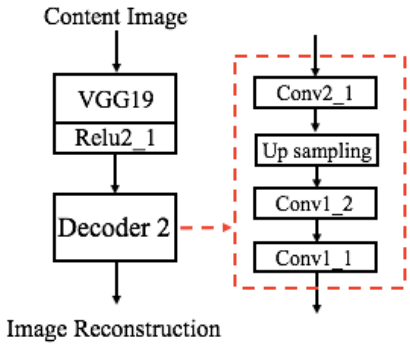

Fig 2 Sample structure of an encoder-decoder. (Right) There is a ReLU activation function after every convolution layer. And variables in the red dash line block are trainable.

The loss function is defined as the sum of reconstruction loss and feature loss. And we further implement another loss into the loss function, the variation loss. Weight can be assigned to the three kinds of loss. In default, weight of both reconstruction loss and feature loss is 1 and that of variation loss is 0. To further improve the result, we conduct experiments on different weight of loss functions. And the default optimizer is Adam optimizer.

In the training step, the algorithm randomly read a set of images with batch size of 100 from training set in every iteration and crop it from size of 512 into size of 128. The performance on test set is evaluated every 10 iterations. This approach is due to the limit of storage resource. Besides, randomness in selecting and cropping

training data also effectively augment the training set and reduce overfitting.

Another technique to prevent overfitting in the training step is early stopping. We check the loss on validation set and save the best model. Besides, we look back to the past several thousand iteration. If the result doesn't improve, the algorithm will automatically stop.

4.1.2 Style Transfer

Decoder training provides us with multi-level stylization models that include trained encoders and decoders. In the style transfer stage, both content and style images are fed into encoders, transferred through whiting and coloring (WCT), and then decoded in order to reconstruct the image. The structure of WCT is shown in Fig. 3:

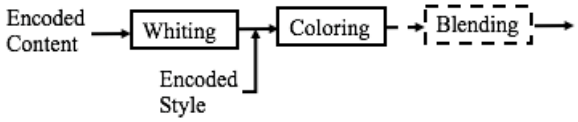

Fig 3 The structure of style transfer. Blending in the dash line block is an optional transferring step.

In the implementation, we also test the output with different α, by which we blend the WCT-processed image with origin content image.

4.2. Software Design

4.2.1 Decoder training

Below is the algorithm for encoder-decoder implementation:

Table 1 Algorithm for Encoder-Decoder

Algorithm 1: Encoder-decoder Implementation

```
def encoder(input,target layer):
While layer is not target layer:
    If type name is Reflect Padding:
        Add a reflect padding step
    If type name is Convolution:
        Read weight and bias from VGG19 model
        Add a non-trainable convolution layer
    If type name is ReLU:
        Add a ReLU activation function
    If type name is Max Pooling:
        Add a max pooling layer
return output, last layer

def decoder(pretrained_VGG19,start layer):
While layer is not start layer:
    If type name is Convolution:
        Read shape of weight and bias from VGG19 model
        Add a reflect padding layer
        Add a trainable convolution layer
    If type name is Max Pooling:
        Add an up sampling layer
return output
```

The encoder-decoder implementation is rather intuitive. However, we design a wrap-up function to combine encoder with decoder so that in every training step and style transfer, we're able to get the model given the target layer.

Table 2 Algorithm for Wrapping up Encoder-Decoder

Algorithm 2: Wrap up Encoder-decoder

```
def get encoder decoder(input, target layer):
    encoded, end layer = encode(input, target layer)
    decoded = decode(encoded, start layer = end layer)
    second encoded, _ = encode(decoded, target layer)
    ed = (input, encoded, decoded, second endcoded)
return ed
```

To combine all techniques discussed in section 4.1.1, we provide pseudo code of training step as below:

Table 3 Algorithm for Training Step

Algorithm 3: Training step for decoders

```
def train_decoder( target_decoder, train_img, val_imgs, batch_size, iters, look_back_period, val_per_iter, feature_weight, pixel_weight, variation_weight, learning_rate, optimizer):
    tensor_list = get_encoder_decoder([target_decoder])
    var_list = [v for v in trainable_variables() if decoder_name in v.name]
    train_gen = Batch_Generator(train_imgs)
    val_gen = Batch_Generator(val_imgs)
    inp = tensor_list[0]
    loss = Train_utils.loss(tensor_list, feature_weight, pixel_weight, variation_weight)
    out = tensor_list[2]
    step = Train.utils.train_step(loss, learning_rate, optimizer)

In the training session:
for itr in range(iters):
    training_batch_x = train_gen.get_batch(batch_size)
    _, cur_loss = sess.run([step,loss], feed_dict = {inp: training_batch_x })

    If not itr % val_per_iter:
        val_batch_x = val_gen.get_batch(batch_size*8)
        val_loss = sess.run([loss], feed_dict = {inp: val_batch_x})
        if best_loss > val_loss:
            best_at = itr
            best_loss = val_loss
            Save the model
```

```
elif itr >= look_back_period + best_at:
    Stop the training step
```

4.2.2 Style Transfer

The algorithm of style transfer is straightforward, which is explained in detail in section 3.2. However, we find three points we should pay attention to in implementation:
(1) A small constant (10e-8) should be added to covariance matrix before performing inverse operation on its eigenmatrix.
(2) tf.svd is reported to be slow on GPU. So we decide to operate it on CPU.
(3) We force the minimum value after SVD to be 1e-5 in case of the instability in the following steps.

5. Results

5.1. Project Results

5.1.1 Result of decoders

We train five decoders separately, and below are the loss on validation set for every decoder. And the loss function is defined as the sum of reconstruction loss and feature loss.

Table 4 Decomposition of Loss of Decoders

	Feature Loss	**Reconstruction Loss**	**Total Loss**
Decoder1	0.016	0.00098	0.017
Decoder2	0.024	0.00069	0.024
Decoder3	0.15	0.0018	0.16
Decoder4	0.081	0.0013	0.08
Decoder5	1.87	0.014	1.88

And here we visualize the performance for every decoder:

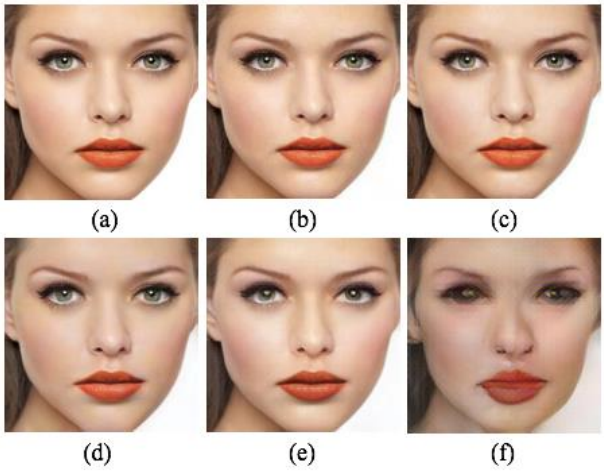

Fig 4 Visualization of the encoder-decoder performance.(a) the original content image; (b)-(f) output processed by decoder 1 to decoder 5 respectively.

5.1.2 Single level style transfer

Here we use encoder-decoders independently to conduct style transfer, and the result is shown below:

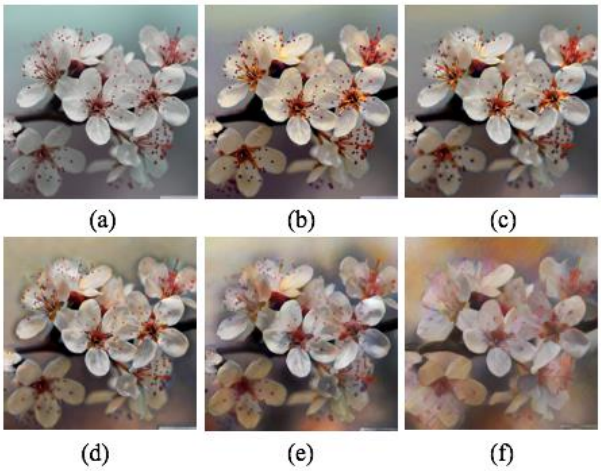

Fig 5 Single level style transfer.(a) the original content image; (b)-(f) style transfer processed by encoder-decoder 1 to encoder - decoder 5 respectively. α in default is 0.6.

5.1.2 Multi-level style transfer

Here we use encoder-decoders in sequence to construct multi-level style transfer. And the figure below shows the output in every transfer step:

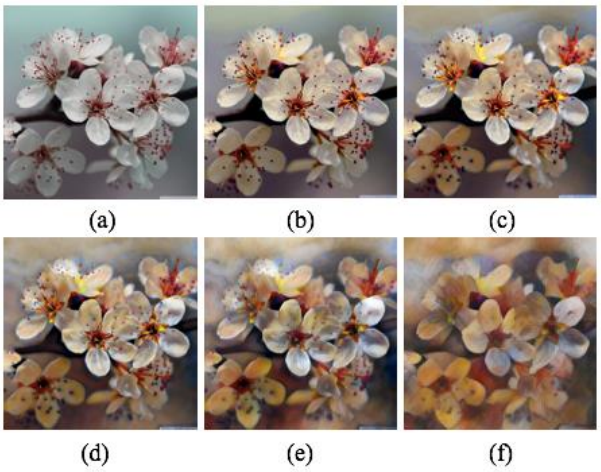

Fig 6 Multi-level style transfer. (a) the original content image; (b)-(f) style transfer processed by encoder-decoder 1 to encoder - decoder 5 in sequence. α in default is 0.6.

5.1.3 Impact of α

The function of α is to determine by what percentage we need to blend the encoded origin content image with WCT processed image. $\alpha = 1$ means that we only use WCT processed image. We conduct experiments on performance of different selection of α and the results are shown below:

Fig 7 Multi-level style transfer with different selection of α

5.2. Comparison of Results

5.2.1 Comparison of decoders performance

Below we visualize the performance of decoder models provided by Li:

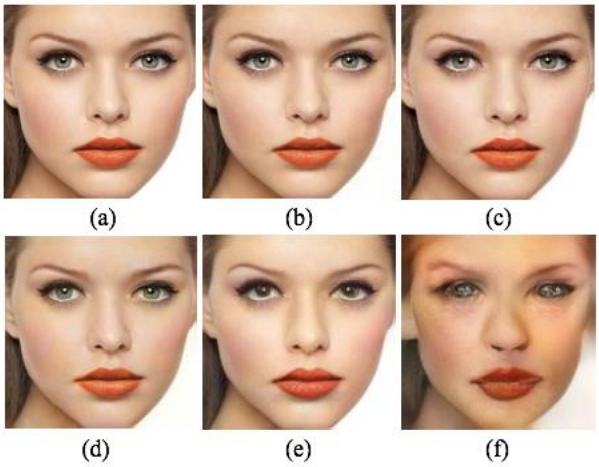

Fig 8 Visualization of the performance of Li's decoder models. (a) the original content image; (b)-(f) output processed by decoder 1 to decoder 5 respectively.

Comparing our decoder models with Li's model, we find there's little difference in performance of decoder 1 to decoder 4. However, our decoder 5 slightly worse than Li's in terms of the clearness of the contour of eyes and the restore of color. And underperformance of decoder 5 could also be observed from validation loss in table 1.

5.2.2 Comparison of single level style transfer

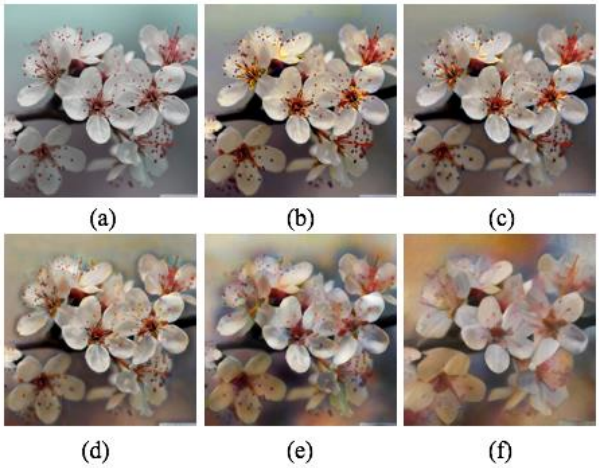

Fig 9 Single level style transfer using Li's model. (a) the original content image; (b)-(f) style transfer processed by encoder-decoder 1 to encoder -decoder 5 respectively. α in default is 0.6.

Comparing Fig 5 and Fig 9, we can barely observe any difference in style transferred images except for image processed by decoder 5. And we can hardly tell which one is better comparing Fig 5(f) and Fig 9(f) because it's a subjective judgement. One difference is that our model image restores more details from the original content while image generated from Li's model is more smooth. And that comes from the difference between decoder 5 of our models and that of theirs.

The result proves the effectiveness of our implementation.

5.2.3 Comparison of multi-level style transfer

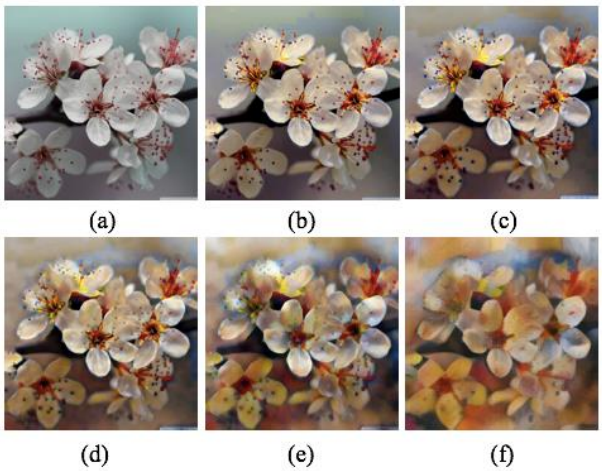

Fig 10 Multi-level style transfer using Li's model. (a) the original content image; (b)-(f) style transfer processed by encoder-decoder 1 to encoder -decoder 5 in sequence. α in default is 0.6.

Comparing Fig 6 and Fig 10, we could draw same conclusion as we discuss in section 5.2.2. And the comparison again proves the effectiveness of our model.

5.3. Discussion of Insights Gained

5.3.1 Training decoders

In the original paper, the authors didn't disclose training details of decoders. So we explore ways by ourselves to tune hypermeters in order to achieve better performance. Based on our experience, we conclude some insights in training decoders, some of which are accorded with our expectation as we explain in section 3.1:

(a) It's difficult to train decoder 1 and decoder 5. The former is because there's too few nonlinearity lies in decoder 1. And the latter is because it's a very deep neural network.

(b) In the training step for decoder 1, we face the issue of obtaining unstable loss on validation set. And this problem could be solved simply by increasing the number of iteration.

(c) In the training step for decoder 5, it's difficult for the algorithm to learn encoded features and reconstruct the image. An efficient method to mitigate this problem is to train this decoder based on the decoder 4.

However, we observe that the result could still be improved. A feasible solution that we would like to purpose is training the convolutional layers between decoder 4 and decoder 5 separately. It means that after we obtain decoder 4, we simply add one convolutional layer and train the new decoder. And we repeat this approach until we construct the entire decoder 5.

(d) Random crop of input images, small batch size and large training set (over 11 thousand images) prove to be effective in preventing overfitting in this case.

(e) Feature loss is in the dominant position among the components in total loss. If we increase the weight of reconstruction loss, the algorithm will focus on the detail of reconstructed image. However, increasing the weight of feature loss will lead to more abstract output, so the selection of weight might be tricky. And we find the default setting has already provided satisfying results.

5.3.2 Style transfer

From the single level style transfer shown in Fig 5, we could observe the function of every decoder. In Fig 5(b)-(d), there're a few features in the style image added to the content image other than general color features. In Fig 5(f), more color features are added while the output image preserves much detail in the original content image.

However, comparing Fig 5(f) and Fig 6(f), we could conclude that multi-level style transfer is better than single level style transfer in terms of the ability of learning the general style of style image rather than colors.

Fig 7 clearly demonstrates how the choice of α affects the output image. The larger the α, fewer details of content image are preserved. So α is proven to be a strong tool to tune the output image, since the judgement on images is rather subjective.

6. Conclusion

In this project, we reproduce a feature transformation method for universal style transfer. With informative encoder-decoder network trained from scratch, we introduce whitening and coloring transforms (WCT) to learn disentangling feature representation for both single level and multi-level style transfer between arbitrary visual styles and content images. Experiments have demonstrated the effectiveness and efficiency of our implementation in term of the quality of style transferred images. And we further summarize insights in training decoders, compare our results with those of original paper and purpose future improvement.

6. Acknowledgement

We'd like to express our thanks to Evan Davis, whose implementation is great help in the beginning, and Yijun Li et al. who offers us with this exciting topic to work on. We'd also express our thanks to Prof. Zoran Kostic and all TAs. Without their dedicated work, we wouldn't be able to get into the fascinating world of neural network and deep learning.

7. References

Include all references - papers, code, links, books.

[1]https://bitbucket.org/ecbm4040/2018_assignment2_sl4234/src/master/HSYG.project.hg2470.sl4234.ys3060/

[2] Li, Yijun, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal style transfer via feature transforms. In NIPS, 2017.

[3] L. A. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis using convolutional neural networks. In NIPS, 2015.

[4] Jun-Yan Zhu*, Taesung Park*, Phillip Isola, Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. arXiv, 2017

8. Appendix

8.1 Individual student contributions in fractions - table

	ys3060	sl4234	hg2470
Last Name	Su	Liu	Gan

Fraction of (useful) total contribution	1/3	1/3	1/3
What I did 1	Build the model	Build the model	Build the model
What I did 2	Test model	Test model	Test model
What I did 3	Final report	Final report	Final report