Real-time Arbitrary Style Transfer for Selfie image with Facial Structure Preservation

CAO Anh Quan
Ecole Polytechnique
anh-quan.cao@polytechnique.edu

Guillaume DUFAU
Ecole Polytechnique
guillaume.dufau@polytechnique.edu

Damien HENRY
Google Arts & Culture
damienhenry@google.com

Abstract

In this project, we proposed three real-time arbitrary style transfer approaches that can preserve better the facial structure of selfie images using depth estimation and face segmentation. Our methods are built on top of AdaIN. Hence, they inherit the arbitrary style transfer capacity from AdaIN. Our approaches require additional computational time during training, whereas they keep the same real-time inference speed of AdaIN. The algorithms allow us to produce new images that combine the content of a selfie with the style of an art image while preserving the structure of the face. Our results demonstrate the effectiveness of our approaches to maintain the facial structure of the selfie images.

1. Introduction

The purpose of this project is to develop a style transfer method for a mobile device. The use case is that the user uses the mobile application to take their selfie image and select an arbitrary art image. Then, the mobile application output a stylized image that has the same content as the selfie image and the style similar to the style of the art image. Hence, we need to have a method to extract the content representation from the selfie image and the style representation from the art image. Thanks to the advance in deep convolutional neural network [1], we can extract good content and style representations of an image.

Many neural style transfer techniques were proposed Gatys *et al.* [2], Huang and Belongie [3] and Johnson *et al.* [4]. However, most of these techniques are designed for generic images. If we apply to the selfie images, they will often distort the structure of the face. Hence, we need a new style transfer technique that can preserve the structure of the face. Our code is available at `https://github.com/caoquan95/AI-VIS-transverse-project`.

In this project, we presented a real-time, arbitrary style transfer technique for selfie images that can preserve the facial structure. We leverage the existing [3] technique so that our method can have real-time inference and transfer arbitrary style. To maintain the structure of the face, we proposed three approaches to preserve the facial structure of the selfie image: preserving the depth map, preserving the face segmented distribution map, and the combination of both methods.

2. Related work

Gatys *et al.* [2] proposed an algorithm that uses a convolutional neural network to generate a new image that matched the style of the style image and the content of a content image. However, their algorithm requires computationally expensive optimization that needs a significant amount of time to generate one image. In the later work, Johnson *et al.* Johnson *et al.* [4] proposed a new method that use convolutional neural network to transform the original image to the stylized image by minimizing the loss function used in Gatys *et al.* [2]. Their method can run in real-time, but the trained neural network is fixed to only one style image. Huang and Belongie [3] presented the AdaIN method that can transfer arbitrary style without training a separate network. Since most of the recent real-time, arbitrary style transfer techniques are the improvements of AdaIN, we decided to develop our methods based on AdaIN.

3. AdaIN

The AdaIN method [3] generate stylized image $\hat{I}$ by combining the content from a content image I_c and the style from a style image I_s as followed

$$\hat{I} = g(AdaIN(f(I_c), f(I_s))).$$

where $f(\cdot), g(\cdot)$ are the encoder and decoder respectively. The $\mathrm{AdaIN}(x, y)$ is a variant of instance normalization [5] with the instance normalization parameters derived the style image representation. We define $x = f(I_c), y = f(I_s)$, $x \in \mathbb{R}^{C\times H\times W}$ and style image $y \in \mathbb{R}^{C\times H\times W}$. Then $\mathrm{AdaIN}(x, y) \in \mathbb{R}^{C\times H\times W}$ and is defined as

$$\mathrm{AdaIN}(x, y)_{cij} = \sigma_c(y)\left(\frac{x_{cij} - \mu_c(x)}{\sigma_c(x)}\right) + \mu_c(y), \quad c = 1, 2, ...C. \tag{1}$$

$$\mu_c(x) = \frac{1}{HW}\sum_{i=1}^{H}\sum_{j=1}^{W} x_{cij} \tag{2}$$

$$\sigma_c(x) = \sqrt{\frac{1}{HW}\sum_{i=1}^{H}\sum_{j=1}^{W}(x_{cij} - \mu_c(x))^2} \tag{3}$$

The style transfer architecture employs the encoder-decoder transform network (fig. 1). The encoder is a fully-convolutional part (until relu4_1 layer) of VGG network, pre-trained on ImageNet dataset [6]. We fix the weights of the encoder in the decoder training. The decoder $g(x)$ is symmetric to the encoder: every max-pooling operator is replaced with double upsampling nearest neighbors layer. All convolution-ReLU blocks are replaced with the same block, except the last one, which has no ReLU layer after it. Hence, the output of the decoder has the same dimensionality as the content image I_c.

Since the AdaIN layer transfers the mean and standard deviation of the style features, we can define the style loss to match these statistics.

$$\mathcal{L}_s = \sum_{i=1}^{L}\|\mu(\phi_i(g(t))) - \mu(\phi_i(s))\|_2 \quad + \quad \sum_{i=1}^{L}\|\sigma(\phi_i(g(t))) - \sigma(\phi_i(s))\|_2 \tag{4}$$

where each ϕ_i denotes a layer in VGG-19 used to compute the style loss. In our experiments, we also use the `relu1_1`, `relu2_1`, `relu3_1`, `relu4_1` layers with equal weights.

4. AdaIN with Depth Preservation

In this section, we introduced our first approach, named Depth AdaIN, that maintains the facial structure of the selfie image by preserving its depth map. We added to the architecture of AdaIN a depth estimation network which generates the depth maps of the content and the output images. Then, we compute the depth loss between the resulting depth maps of the content and output images. The total loss is the weighted sum of our depth loss and the style loss and content loss from AdaIN. The weights of the depth loss, content loss, and style loss are 100.0, 1.0, and 10.0, respectively. The architecture of Depth AdaIN is shown in figure 2.

Similar to AdaIN, Depth AdaIN only needs the Encoder, AdaIN, and Decoder to transfer style. The Encoder and depth estimation networks are pre-trained, and their weights are fixed during training. We only need to train the Decoder. Since Depth AdaIN is built on AdaIN, it inherits the arbitrary style transfer capacity from AdaIN. The depth estimation network is only required during training so that Depth AdaIN has the same real-time inference speed as AdaIN.

4.1. Depth Estimation Network

We decided to design our version of the depth estimation network architecture. To design the network, we first selected the dataset LineMOD as the benchmark dataset. The LineMOD dataset contains RGB images of many small objects with the corresponding depth map, as shown in figure 4.1. We chose this dataset because the slight depth deviation between the small objects is similar to the depth deviation of the face, such as the depth deviation between the nose, mouth, and eyes.

We trained a supervised neural network to predict depth from the RGB images on the rest of the objects in the LineMOD dataset, which is not used for object pose prediction. The input is the RGB image, and the output is the corresponding depth map.

We implemented three versions of the depth estimation network. The baseline is DepthV1, which use the Pyramid Parsing Network with logL1 loss. After that, we implemented the DepthV2 by replacing the Pyramid Parsing Module in the network by the Up-projection block proposed in [7], which results in a considerable improvement in the performance. Finally, we created the DepthV3 model by extending the DepthV2 model. The DepthV3 model contains multi-scale feature extraction and multi-scale feature fusion to integrate the information at different scales. We also implemented the new loss function, which includes three components: LogL1 loss, gradient loss, and surface normal loss.

4.1.1 Loss for depth estimation

Following [8], the loss is the weighted sum of three components: LogL1 loss, gradient loss and the surface normal loss. Let us denote the estimated and ground truth depth maps as d and g. The d_i and g_i are the estimated depth and ground truth depth for the pixel i. The number of pixels is n. The LogL1 loss is defined as

$$l_{logL1} = \frac{1}{n}\sum_{i=1}^{n}\log(|d_i - g_i| + 0.5)$$

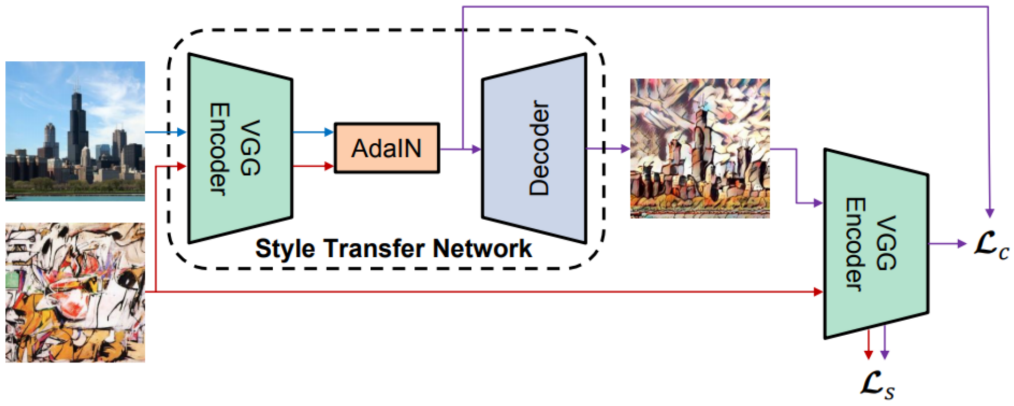

Figure 1. AdaIN style transfer architecture [3].

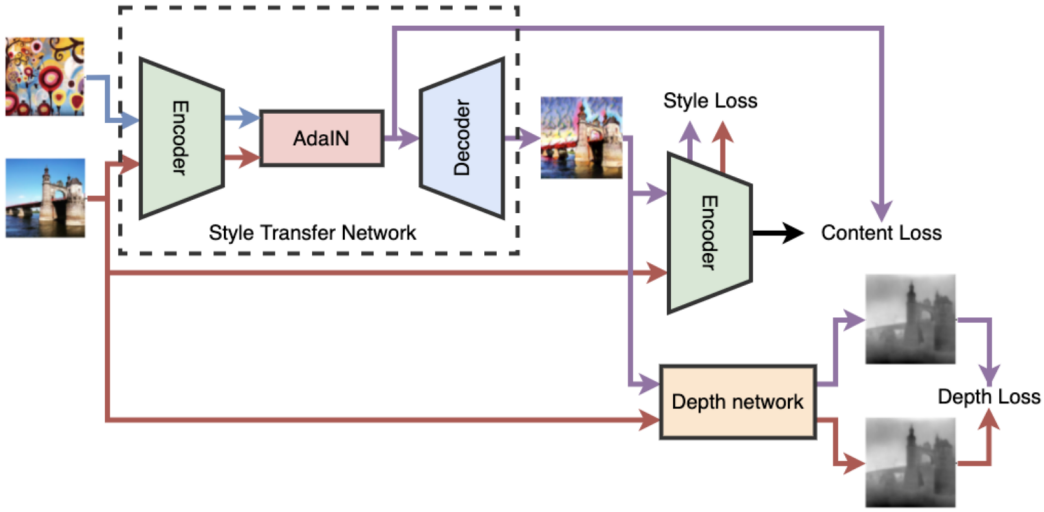

Figure 2. Depth AdaIN.

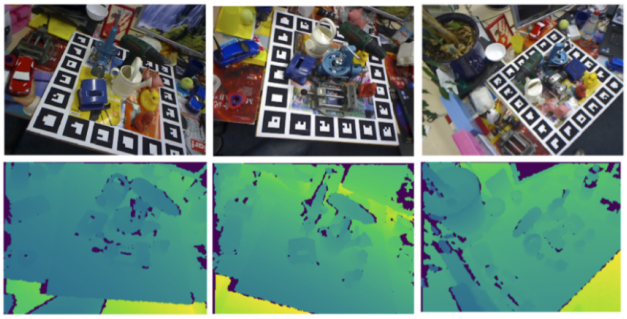

Figure 3. LineMOD dataset

If we only used the LogL1 loss, the edges in estimated depth map are often blurry and distorted. Thus, we need the gradient loss

$$l_{grad} = \frac{1}{n}\sum_{i=1}^{n}[\log(|\nabla_x(d)_i| - \nabla_x(g)_i|) + \log(|\nabla_y(d)_i| - \nabla_y(g)_i|)] \quad (5)$$

where $\nabla_x(.)_i$ is the gradient value in the x direction at pixel i. However, the gradient loss is only effective with the strong edge, it cannot deal with the high frequency undulation of a surface. We need the surface normal loss

$$l_{normal} = \frac{1}{n}\sum_{i=1}^{n}\left(1 - \frac{n_i^d \cdot n_i^g}{\sqrt{n_i^d \cdot n_i^d}\sqrt{n_i^g \cdot n_i^g}}\right)$$

where $n_i^d = [-\nabla_x(d)_i, -\nabla_y(d)_i, 1]^T$ and $n_i^g = [-\nabla_x(g)_i, -\nabla_y(g)_i, 1]^T$ are the surface normal of the esti-

mated depth map and ground truth depth map respectively. Finally, the total loss is the sum all the loss above

$$l_{total} = l_{logL1} + l_{grad} + l_{normal}$$

4.1.2 Depth estimation architecture

In general, our network employs encoder-decoder architecture. The encoder is the modified ResNet that produces the representation with a large number of channels but small size. The decoder is the stack of four up-sampling layers that increases the size of the representation but decreases the number of channels.

DepthV1: Pyramid Parsing Network In the first version, we use ResNet18 He *et al.* [9] as the encoder. The decoder is the stack of multiple Pyramid Pooling Modules that were proposed in [10]. The idea is to capture both global and local information by fusing the features under different pyramid scales when performing up-sampling. The input is divided into grids with different sizes. Then, the pooling operation is performed in each cell of the grids. The output from each grid is upsampled and concatenated. Finally, the concatenated representation is fed into a convolution layer. Figure 4.1.2 illustrates the architecture of the Pyramid Parsing Module.

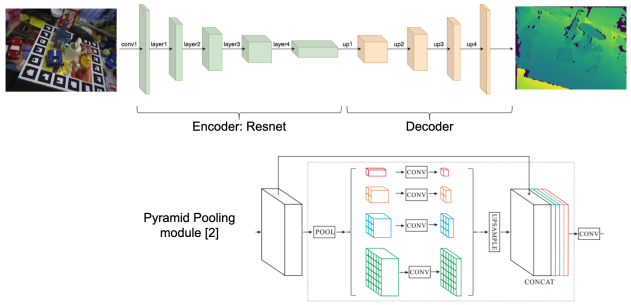

Figure 4. Pyramid Parsing Module

DepthV2: Up-projection block In this version, we replaced the decoder by the stack of multiple Up-Projection Blocks proposed in [7]. The approach is double the size by mapping each entry into the top left corner of $2x2$ (zero) kernel and followed by a $5x5$ convolution. Thus, it is applied to more than one non-zero element at each location. The architecture of the up-projection block is shown in figure 4.1.2.

DepthV3: Multi-scale feature ensemble Depth estimation requires good context information for a good results [11]. The context information can be captured efficiently by a multi-scale feature ensemble. Hence, we updated our

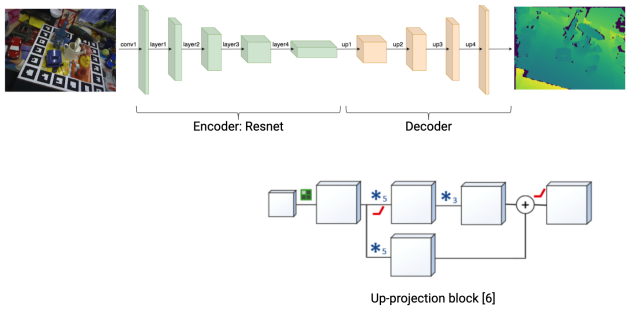

Figure 5. Up-projection block

network design following multi-scale network architecture proposed by Godard *et al.* For each block, the representation is up-scaled, concatenated together, and fed to the last convolutional layer.

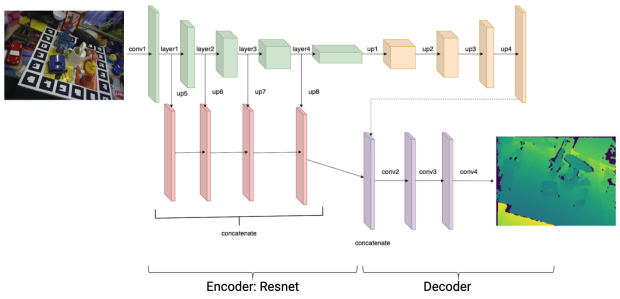

Figure 6. Multi-scale feature ensemble

4.1.3 Depth filling using Colorization method

Since the depth map contains many missing values, we need to handle these values when computing the loss. The first natural solution is just not included in these pixels in the loss. However, we cannot use this solution for the gradient loss and surface normal loss because the calculation involves the neighbor pixels. Therefore, we used the colorization method from [12] to fill in the missing depth map. The method is based on the assumption that the nearby pixels in space-time that have similar gray levels should also have similar colors. In our case, we replace the colors by the depth values. The results are illustrated in figure 4.1.3.

4.2. Implementation details

We augmented the data using color jitter that scales the brightness, contrast, and saturation values of the RGB image by $c \in [0.6, 1.4]$. We followed the same procedure as the one in [13]. The RGB-D images have a size of $480x640$ pixels. We trained each of the proposed networks for 20 epochs. The encoder is initialized by a model pre-trained with the ImageNet dataset. All the other layers are initialized randomly. We used Adam optimizer with the learning

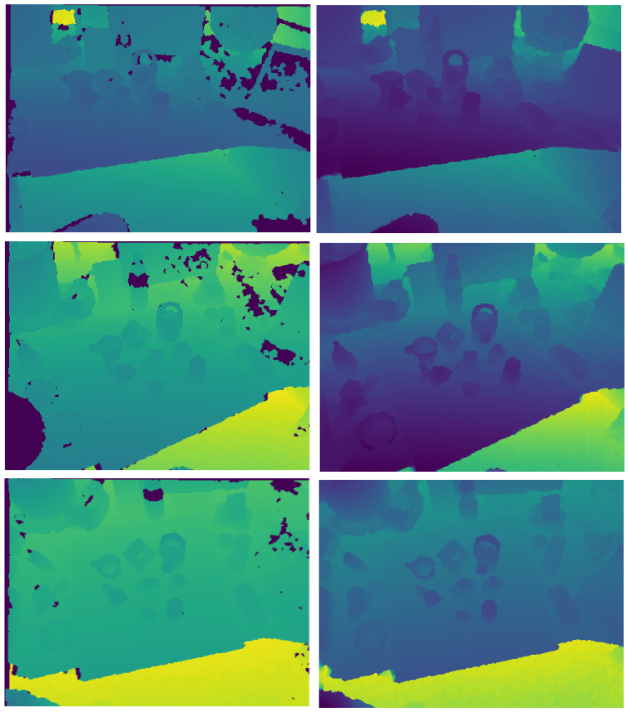

Figure 7. Depth filling using Colorization method. The left column is the original depth map. The right column is the filled depth map.

rate of 0.0001 and reduced to 30% if the test loss plateaued. All the experiments are conducted in PyTorch with a batch size of 8.

4.3. Performance comparison on benchmark dataset

I used the same performance measures from the previous works [7], [8]. Let us denote the number of pixels in all evaluated images by T. The performance measures are

- Root mean squared error (RMS): $\sqrt{\frac{1}{T}\sum_{i=1}^{T}(d_i - g_i)^2}$
- Mean relative error (REL): $\frac{1}{T}\sum_{i=1}^{T}\frac{||d_i - g_i||}{g_i}$
- Mean log10 error (log10): $\frac{1}{T}\sum_{i=1}^{T}||\log_{10} d_i - \log_{10} g_i||$
- Thresholded accuracy: Percentage of d_i such that $\max\left(\frac{d_i}{g_i}, \frac{g_i}{d_i}\right) = \delta <$ threshold

Table 1 show the quantitative performance measurements of the three architectures: DepthV1, DepthV2, DepthV3. We can see that the use of the Up-projection block in DepthV2 outperforms the DepthV1 with the Pyramid Parsing Module. The DepthV3 has the performance thanks to the multiscale architecture and the inclusion of

Method	RMSE	REL	log 10	$\delta < 1.25$
DepthV1	423.693	0.208	0.1	0.582
DepthV2	266.554	0.122	0.053	0.87
DepthV3	**142.974**	**0.064**	**0.027**	**0.966**

Table 1. Comparison of the the three architecture on the LineMOD dataset.

gradient and surface normal loss. The mean relative error (REL) of the DepthV3 is 0.064, which means that on the average difference between the predicted and ground truth values is 6.4%. I want to compare my implementation with the state-of-the-art method, but I don't have time since the training is quite long, and I have limited Google Cloud credit. The qualitative results are shown in figure 8.

4.4. Face depth estimation

To train the depth estimation network, we first used the cropped faces Pandora dataset (Borghi *et al.* [14]) (figure 4.4). The dataset contains 250k RGB and depth images (100x100 pixels). However, since all the images are captured in a lab setting with a similar environment, the trained network cannot generalize to arbitrary images in the wild. Therefore, we solved this problem by mixing the Pandora with RGB-D Scenes [15] dataset so that the depth estimation network can learn more depth patterns. The RGB-D Scenes Dataset contains 300 household objects, which was recorded using a Kinect style 3D camera. The performance of the depth estimation network on the images in the world increase reasonably after training on the mixed dataset. However, the results are not perfect, but we think they are still good enough for style transfer tasks since we can still realize the shape of the face and different parts of the face in the depth map. The results of the depth estimation network trained with different datasets on arbitrary images are shown in figure 11.

4.5. Training the Depth AdaIN

In order to train the full Depth AdaIN architecture, we first pre-train AdaIN following the original paper using 80k images from the MS-COCO dataset as content images and 10k images collected from WikiArt as style images. We trained it for about 15 hours with 200,000 iterations and 8 images per iteration. The depth estimation network is pre-trained on the mixed RGBD dataset, and the training took about 10 hours. Finally, we train the full architecture using the face images from the Helen dataset [16] that contain 3k face images in the wild as content images and 10k images collected from WikiArt as style images. We fixed the weights of the Encoder and depth estimation network. We only train the decoder part of the network. The full training took 35 hours with 100,000 iterations and 8 images per iteration. All the training is done using a Tesla V100 GPU.

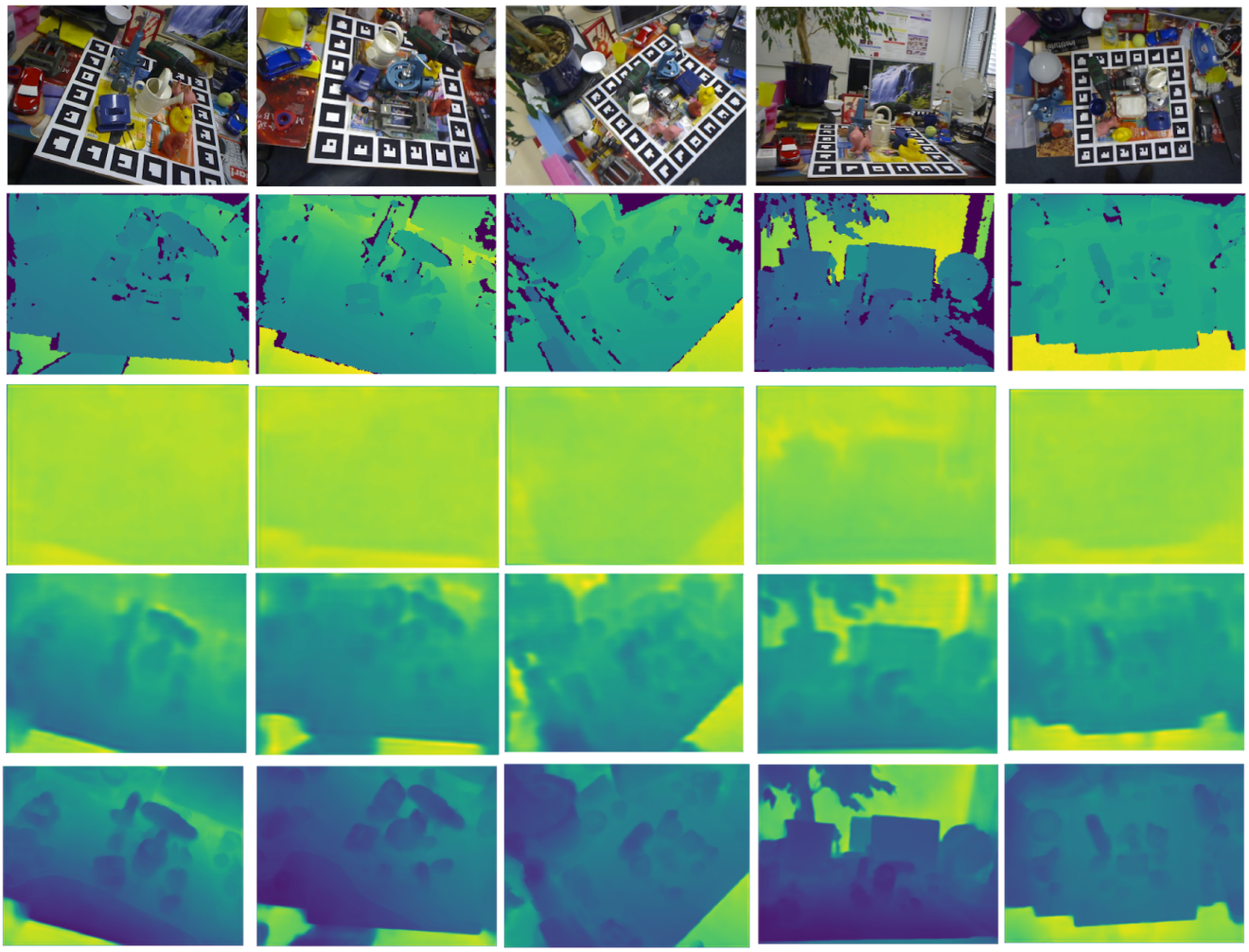

Figure 8. Qualitative results of the depth estimation networks. From top to bottom: RGB image, Ground-truth, predicted depth from DepthV1, predicted depth from DepthV2, predicted depth from DepthV3.

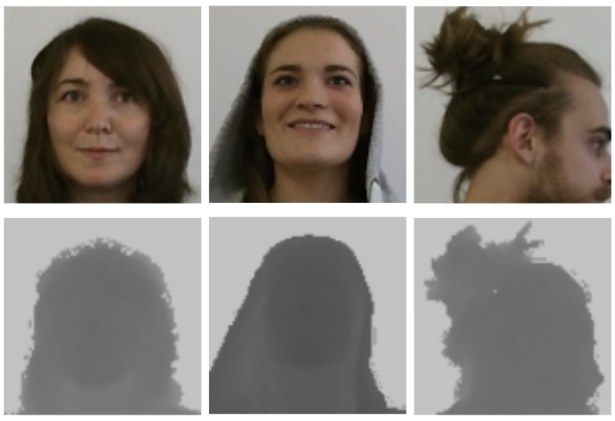

Figure 9. Pandora dataset.

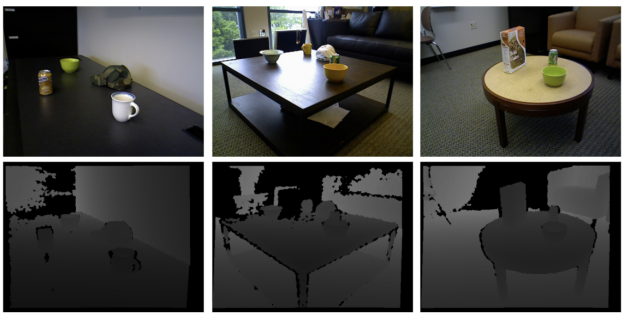

Figure 10. RGB-D Scenes dataset.

4.6. Depth AdaIN Results

Figure 12 illustrate the good results from our Depth AdaIN method compare to the results from the original AdaIN method. The network does not see all the style and

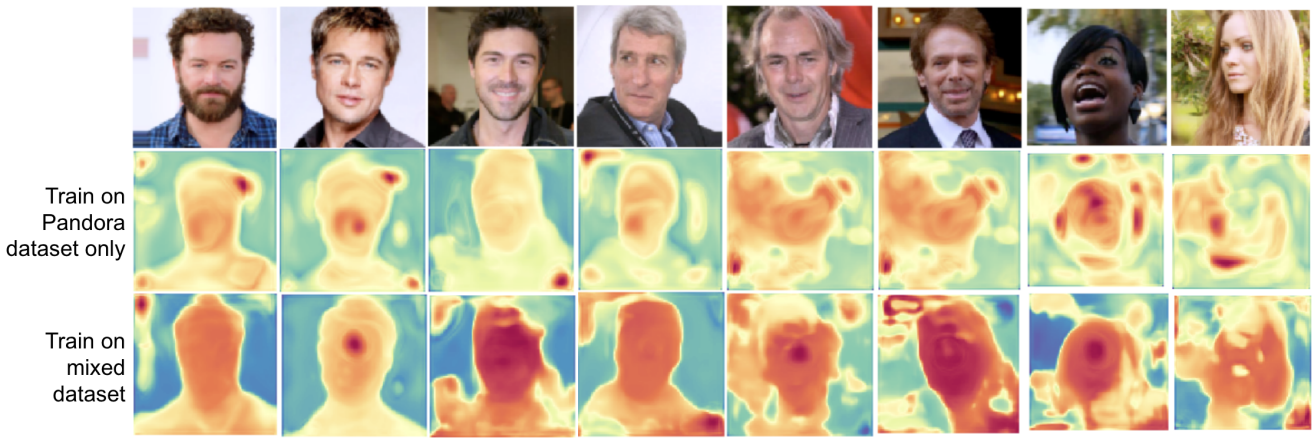

Figure 11. Results of the depth network on images in the wild

content images during training. We can see that the output of the stylized images from our approach has a much better facial structure than the original outputs from AdaIN. The shadow of the face, the chin, the neck, the cheek, and the nose is much clearer and better. The background and the face are less noisier.

From the output depth maps of the stylized images, it seems that the network can recognize quite well the overall depth of the face, which results in good output images. However, If the content is too complicated and different from the mixed depth dataset that is used to train the depth estimation network, the resulting image will have bad quality. The quality of the output stylized images also depends on the quality of the depth map output from the depth estimation network and the style transfer powers of AdaIN.

We noticed the Depth AdaIN failed with several kinds of art images. We believe the reason comes from the style transfer capacity of AdaIN. Figure 13 shows an example when the result is bad due to the limited style transfer strength of AdaIN.

5. AdaIN with Face Segmentation

Our next approach is to replace the depth estimation network by a face segmentation network. Instead of preserving the depth map of the selfie image, we preserve the probability map of the segmented face map. Hence, we called this approach as Face Segmentation AdaIN. The architecture of the Face Segmentation AdaIN is shown in figure 14. The face segmentation network predicts the probability face segmented maps of the selfie and the output image. Then, the face segmentation loss is the L2 loss between these two probability face segmented maps. The total loss is the weighted sum of the three losses: style loss, content loss, and face segmentation loss. The weights of the Face Segmentation loss, content loss, and style loss are 30.0, 1.0, and 10.0, respectively.

Same as Depth AdaIN, Face segmentation AdaIN only requires Encoder, AdaIN, and Decoder for style transfer. The encoder and face segmentation network are pre-trained and fixed during training. Only the decoder is trained. Since Face Segmentation AdaIN is an increment of AdaIN, it can transfer arbitrary style. The face segmentation network is only required during training. Hence, we maintain a real-time inference speed of AdaIN.

5.1. BiSeNet: Bilateral Segmentation Network

Instead of designing the face segmentation network from scratch, we used an existing architecture named BiSeNet proposed by Yu *et al.* for our face segmentation network. The architecture of BiSeNet is shown in figure 15.

Semantic segmentation requires rich spatial information (low-level) and context information (high-level) to achieve high performance. The convolutional neural network encodes high-level semantic information by consecutive down-sampling operations, which destroys the spatial information of the image. However, spatial information of the image is crucial in semantic segmentation. The spatial information is maintained in modern convolutional neural networks using dilated convolution or sizeable receptive field. The context information is captured by enlarging the receptive field, fusing different context information, or multi-scale feature ensemble.

Yu *et al.* proposed BiSetNet that captures effectively both spatial information and context information. There are three important components in their architecture. The first one is the Spatial Path, which consists of three layers. Each layer contains a convolution with $stride = 2$, followed by batch normalization[18] and ReLU [19]. Hence, it extracts the output feature maps, which is 1/8 of the original image. It encoder rich spatial information due to the large spatial

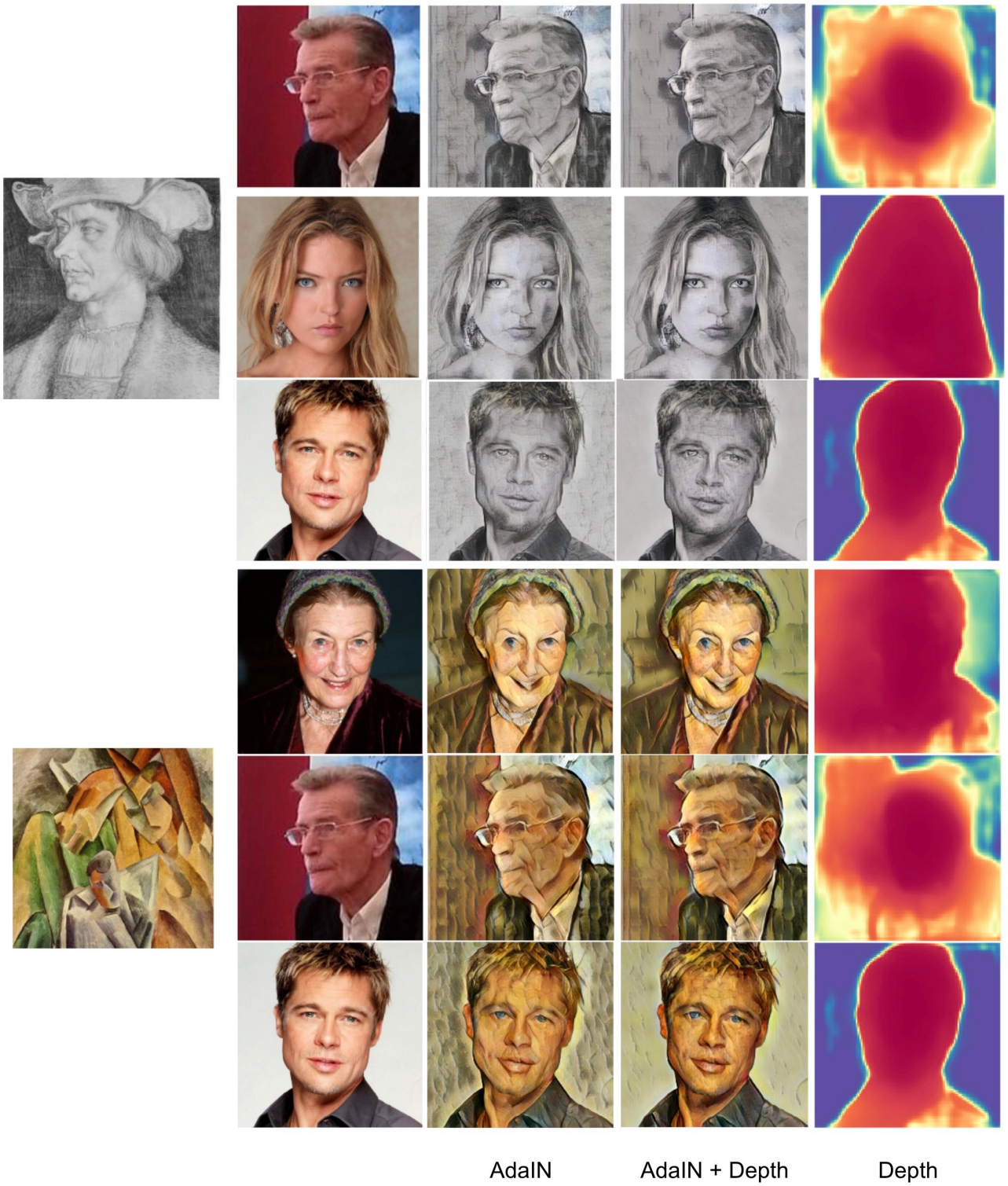

Figure 12. Depth AdaIN Good Results. On the left, we have the style images. The first column contains the original images. The second column is the output of the stylized images from the original AdaIN method. The third column is the results of our Depth AdaIN method. The last column is the depth map of the output of the stylized images from our Depth AdaIN method predicted by the depth estimation network.

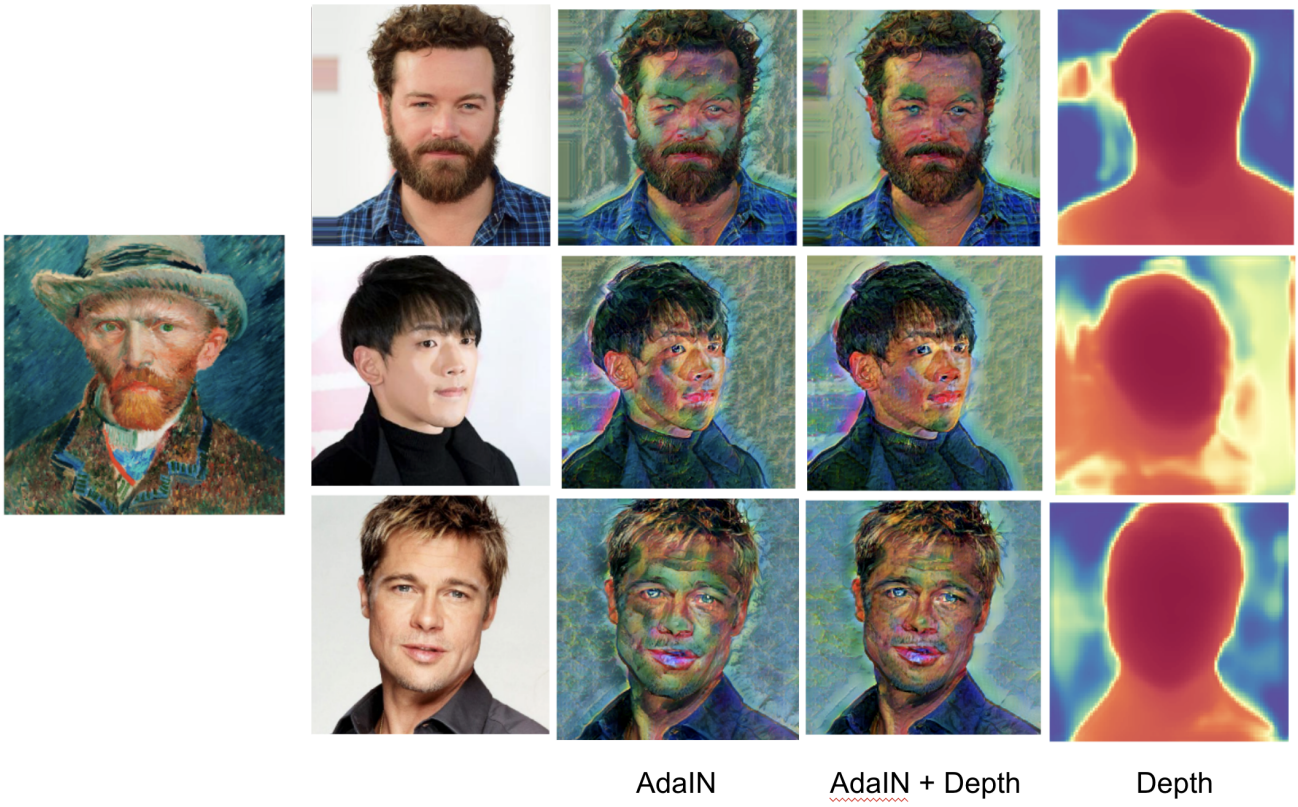

Figure 13. Depth AdaIN Bad Results. On the left, we have the style images. The first column contains the original images. The second column is the output of the stylized images from the original AdaIN method. The third column is the results of our Depth AdaIN method. The last column is the depth map of the output of the stylized images from our Depth AdaIN method predicted by the depth estimation network.

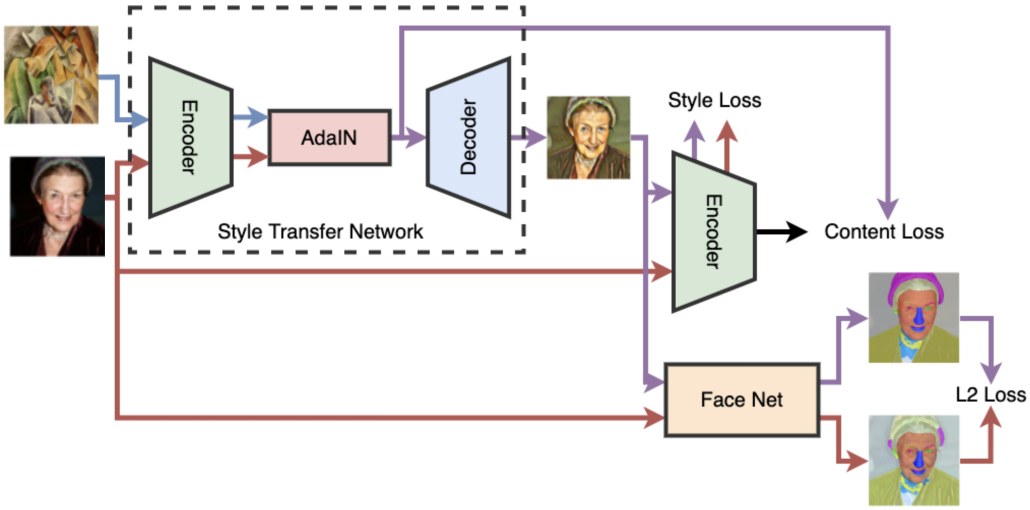

Figure 14. Face Segmentation AdaIN.

size of the feature map. The second important component is the Context Path. It is designed to have a large receptive field to capture sufficient context information. It consists of a lightweight model, such as Xception [20], which

have a large receptive field by downsampling the feature map very fast, and a global average pooling on the tail of the lightweight model that can provide the maximum receptive field with global context information. The last important component is the Attention Refinement Module, which implements the attention mechanism [21]. It computes the attention vector to guide feature learning.

5.2. Face Segmentation Dataset

To train the BiSeNet, we use the CelebAMask-HQ dataset [22]. It contains $30,000$ high-resolution face images selected from the CelebA dataset by following CelebA-HQ. Each image has a segmentation mask of facial classes corresponding to the CelebA. The masks of the CelebAMask-HQ were annotated manually with the resolution of 512x512 and 19 classes that contain facial components such as skin, nose, eyes, eyebrows, and mouth (figure 16).

5.3. Training Face Segmentation AdaIN

We trained the Face Segmentation AdaIN, similar to the Depth AdaIN. The Face Segmentation Network is pretrained on the CelebAMask-HQ dataset, which took about 9 hours.

5.4. Face Segmentation AdaIN Results

Figure 17 shows the results of the Face Segmentation AdaIN method compare to the original AdaIN. We can see that the resulting stylized images are much better than the ones from the original AdaIN. We believe the outputs have good quality since the resulting segmented face maps are quite accurate. We can see different parts of the face. However, in some regions, it seems that the face segmentation network reduces the strength of the style to preserve the facial structure. We think we can solve the problem by replacing the L2 Loss used in the Face Segmentation loss. We leave this issue for future work. Same as Depth AdaIN, the ability of style transfer is limited by the strength of AdaIN, which leads to worse results for some kind of art images.

6. AdaIN with both Face Segmentation and Depth Preservation

Finally, we combined both Face Segmentation network and Depth Estimation network with AdaIN, as shown in figure 18. We add the Face Segmentation Network and Depth Estimation Network to the architecture of AdaIN. We estimate the depth maps of the output image and the content image by the depth estimation network. Then, we compute the depth loss using the two depth maps as in the Depth AdaIN approach. We used the Face Segmentation Network to estimate the Probability Face Segmented maps of the content image and the output images. The Face Segmentation Loss is the L2 difference between the two Probability Face Segmented maps. The total loss is the weighted sum of the style loss, the content loss, the Face Segmentation loss, and the depth loss with the weights of $10.0, 1.0, 30.0$, and 100.0, respectively. Because we only need the Face Segmentation and Depth Estimation networks during train, our approach still has real-time inference speed as the original AdaIN.

We train the full architecture of AdaIN with both Face Segmentation and Depth Preservation, similar to the training process of Depth AdaIN and Face Segmentation AdaIN.

7. Results Comparison of the three approaches

We show the results of all approaches in figure 19. We can see that the outputs from the combination of both Face Segmentation and Depth Estimation are noticeably better than depth or face segmentation alone. With the face segmentation, the network can preserve different parts of the face such as the eyes, the skin, and the mouth, whereas, the depth estimation helps maintain the depth of the face such as the shadow of the face, the nose, and the hair.

8. Conclusion

In this project, we have successfully demonstrated a real-time arbitrary style transfer approach that can preserve much better Facial Structure. We proposed three approaches: Depth AdaIN, Face Segmentation AdaIN, and the combination of both of them. We saw a significant increase in the quality of the output images, but it is not always the case since the quality of the output images depends on the performance of the depth estimation and the face segmentation networks. If the output from these network has high quality, the stylized output images will have good quality. The latter approach that combines both Depth Estimation and Face Segmentation has the best results among the three approaches since it can maintain both the depth and the detail of different parts of the face.

9. Future works

Although our approaches show promising results, there is still a lot of work to do. First of all, we need a better method than AdaIN since we can see that AdaIN failed with several kinds of art images. Hence, the next step is to replace AdaIN by the approach proposed by Li *et al.* that is an improvement of AdaIN. The AdaIN module transfers the style by transforming the first-order statistics such as mean and standard deviation. In the new method [23], instead of transferring the first-order statistics, they train the network to learn a linear transformation matrix (figure 9) for transferring the style. They demonstrated a significant improvement in the quality of the output images comparing to AdaIN.

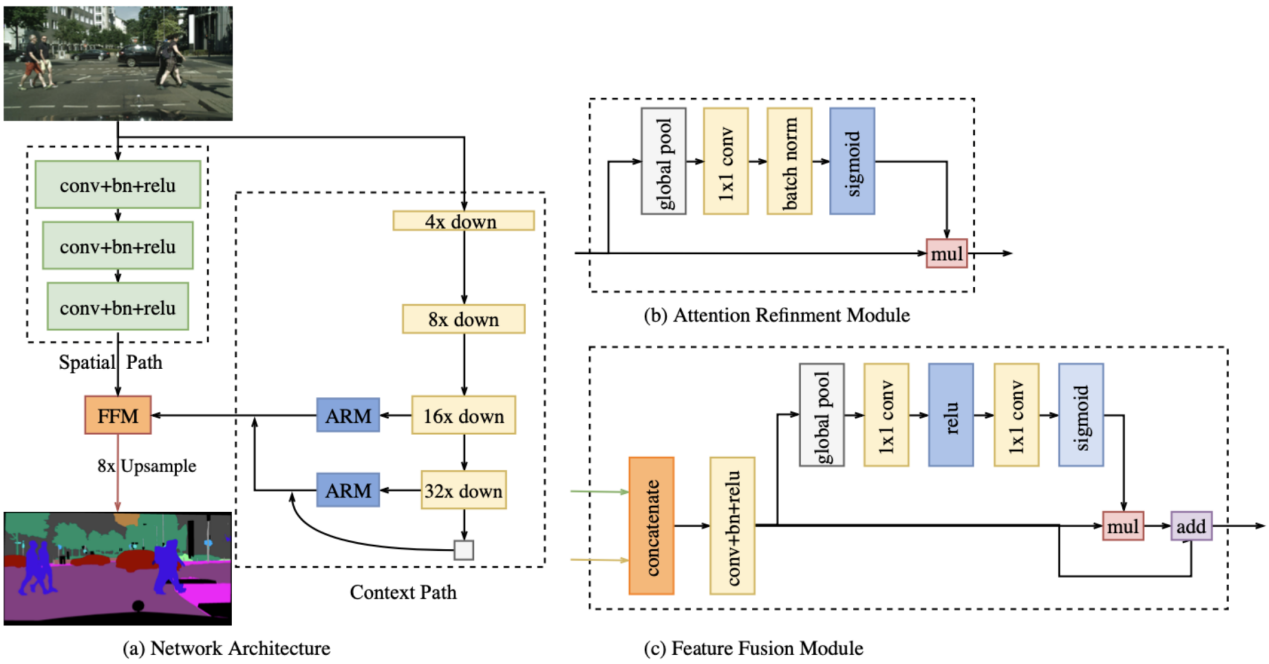

Figure 15. BiSeNet: Bilateral Segmentation Network.

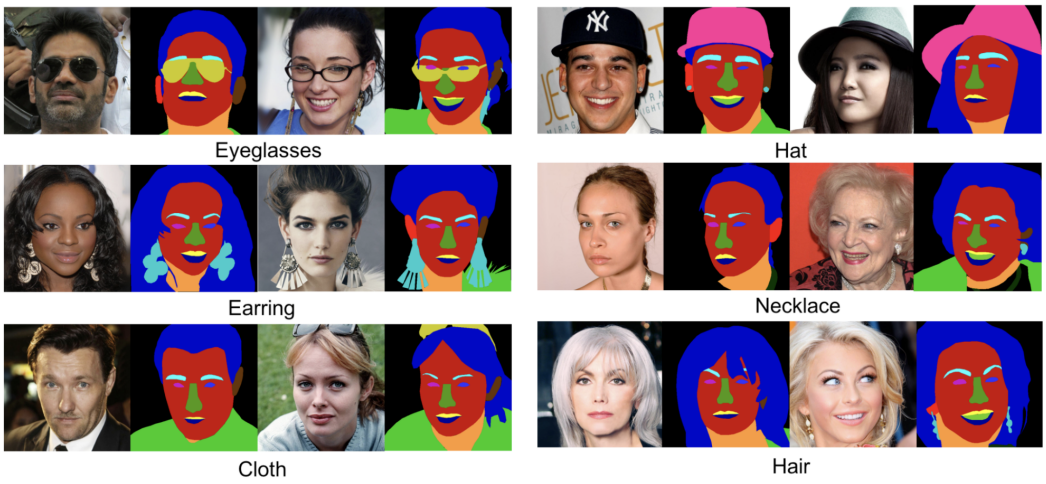

Figure 16. CelebAMask-HQ dataset.

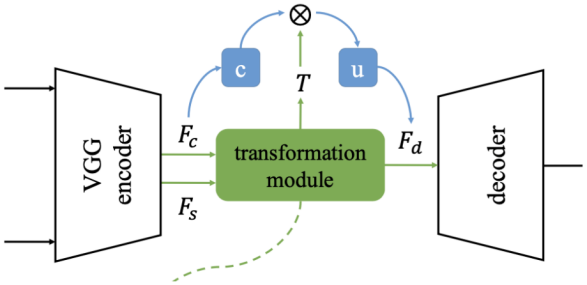

Figure 20. Transformation module.

The next question to ask is how much depth, face segmentation we should preserve? Since too strong preserving depth or face segmentation can result in reducing too much the strength of the style. We can solve this problem by finding a way to allow real-time interpolation between the amount of depth, face segmentation preserved, the strength of the style, and the content so that the user can select the sweet spot that produces the output image that he/she wants.

The last issue is the need for a better loss in the case of Face Segmentation as we can see that the use of L2 loss for

Face Segmentation can reduce the style significantly from the output images.

References

(1) A. Krizhevsky, I. Sutskever and G. Hinton, *Neural Information Processing Systems*, 2012, **25**, DOI: 10.1145/3065386.

(2) L. Gatys, A. Ecker and M. Bethge, 2016, pp. 2414–2423.

(3) X. Huang and S. Belongie, *Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization*, 2017.

(4) J. Johnson, A. Alahi and L. Fei-Fei, *Perceptual Losses for Real-Time Style Transfer and Super-Resolution*, 2016.

(5) D. Ulyanov, A. Vedaldi and V. Lempitsky, *Instance Normalization: The Missing Ingredient for Fast Stylization*, 2016.

(6) J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and F. F. Li, 2009, pp. 248–255.

(7) I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari and N. Navab, *Deeper Depth Prediction with Fully Convolutional Residual Networks*, 2016.

(8) J. Hu, M. Ozay, Y. Zhang and T. Okatani, *Revisiting Single Image Depth Estimation: Toward Higher Resolution Maps with Accurate Object Boundaries*, 2018.

(9) K. He, X. Zhang, S. Ren and J. Sun, *Deep Residual Learning for Image Recognition*, 2015.

(10) H. Zhao, J. Shi, X. Qi, X. Wang and J. Jia, *Pyramid Scene Parsing Network*, 2016.

(11) C. Godard, O. M. Aodha, M. Firman and G. Brostow, *Digging Into Self-Supervised Monocular Depth Estimation*, 2018.

(12) A. Levin, D. Lischinski and Y. Weiss, ACM SIGGRAPH 2004 Papers, Association for Computing Machinery, Los Angeles, California, 2004, pp. 689–694.

(13) C. Wang, D. Xu, Y. Zhu, R. Martín-Martín, C. Lu, L. Fei-Fei and S. Savarese, *DenseFusion: 6D Object Pose Estimation by Iterative Dense Fusion*, 2019.

(14) G. Borghi, M. Venturelli, R. Vezzani and R. Cucchiara, 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 5494–5503.

(15) K. Lai, L. Bo, X. Ren and D. Fox, 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 1817–1824.

(16) V. Le, J. Brandt, Z. Lin, L. Bourdev and T. S. Huang, Computer Vision – ECCV 2012, ed. A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato and C. Schmid, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 679–692.

(17) C. Yu, J. Wang, C. Peng, C. Gao, G. Yu and N. Sang, *BiSeNet: Bilateral Segmentation Network for Real-time Semantic Segmentation*, 2018.

(18) S. Ioffe and C. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, 2015.

(19) X. Glorot, A. Bordes and Y. Bengio, 2010, vol. 15.

(20) F. Chollet, *Xception: Deep Learning with Depthwise Separable Convolutions*, 2016.

(21) A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, *Attention Is All You Need*, 2017.

(22) C.-H. Lee, Z. Liu, L. Wu and P. Luo, *MaskGAN: Towards Diverse and Interactive Facial Image Manipulation*, 2019.

(23) X. Li, S. Liu, J. Kautz and M.-H. Yang, *Learning Linear Transformations for Fast Arbitrary Style Transfer*, 2018.

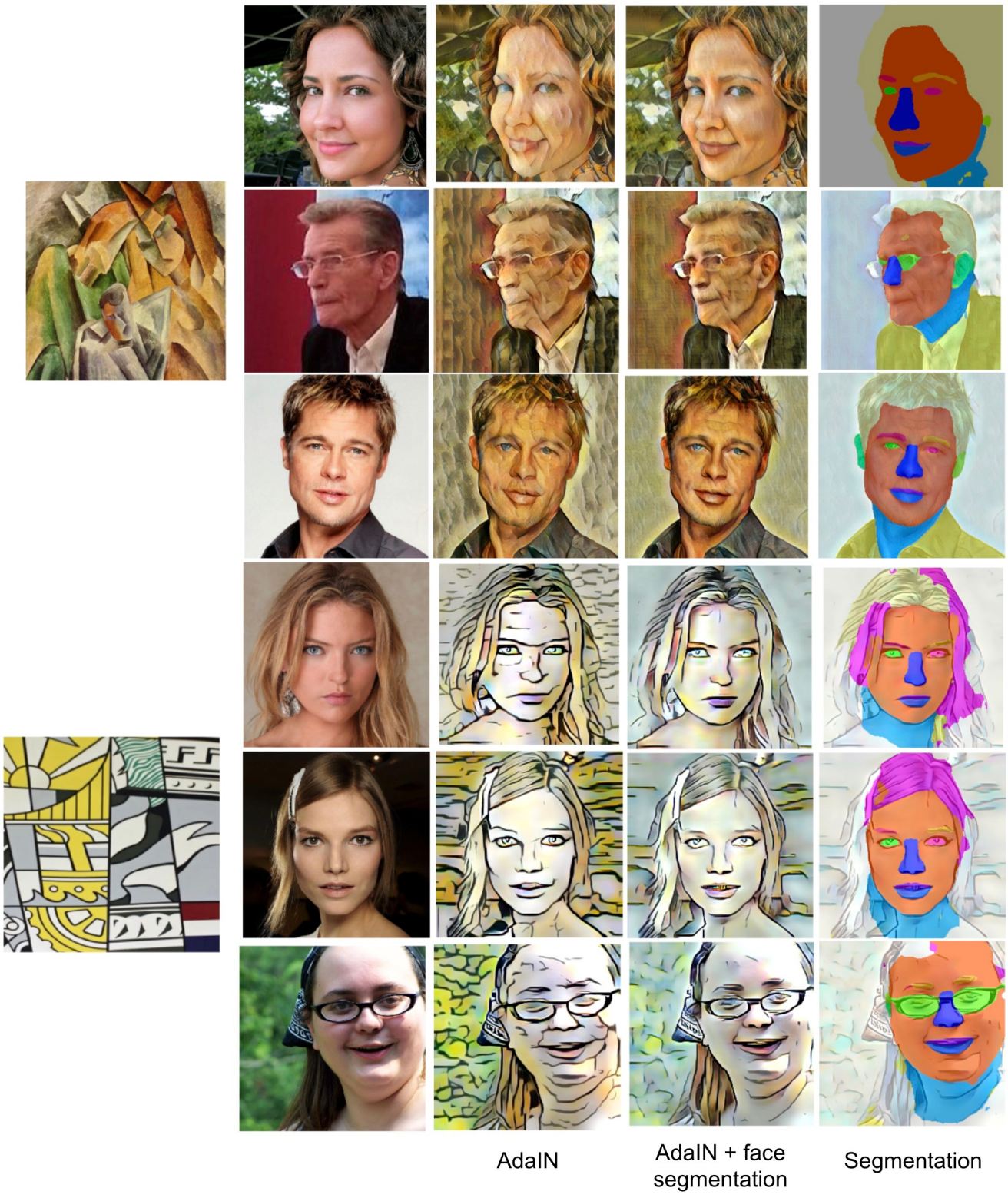

Figure 17. Face Segmentation AdaIN Results. On the left, we have the style images. The first column contains the original images. The second column is the output of the stylized images from the original AdaIN method. The third column is the results from our Face Segmentation AdaIN method. The last column is the Segmented Face Maps of the output of the stylized images from our Face Segmentation AdaIN method predicted by the Face Segmentation Network.

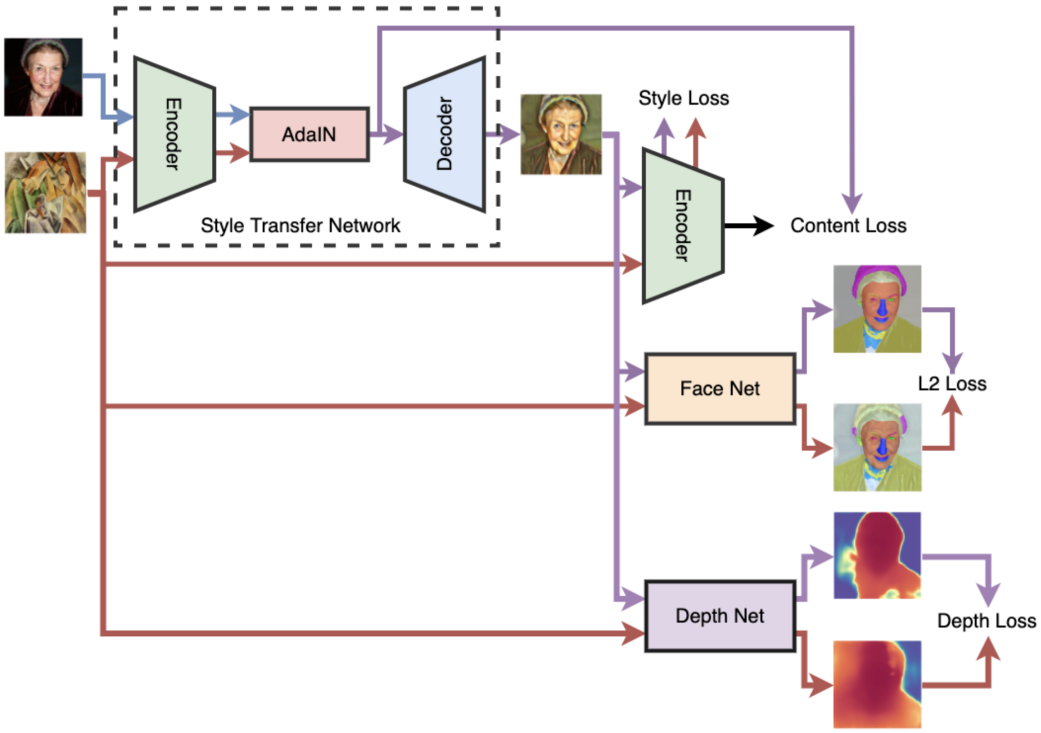

Figure 18. AdaIN with both Face Segmentation and Depth Preservation.

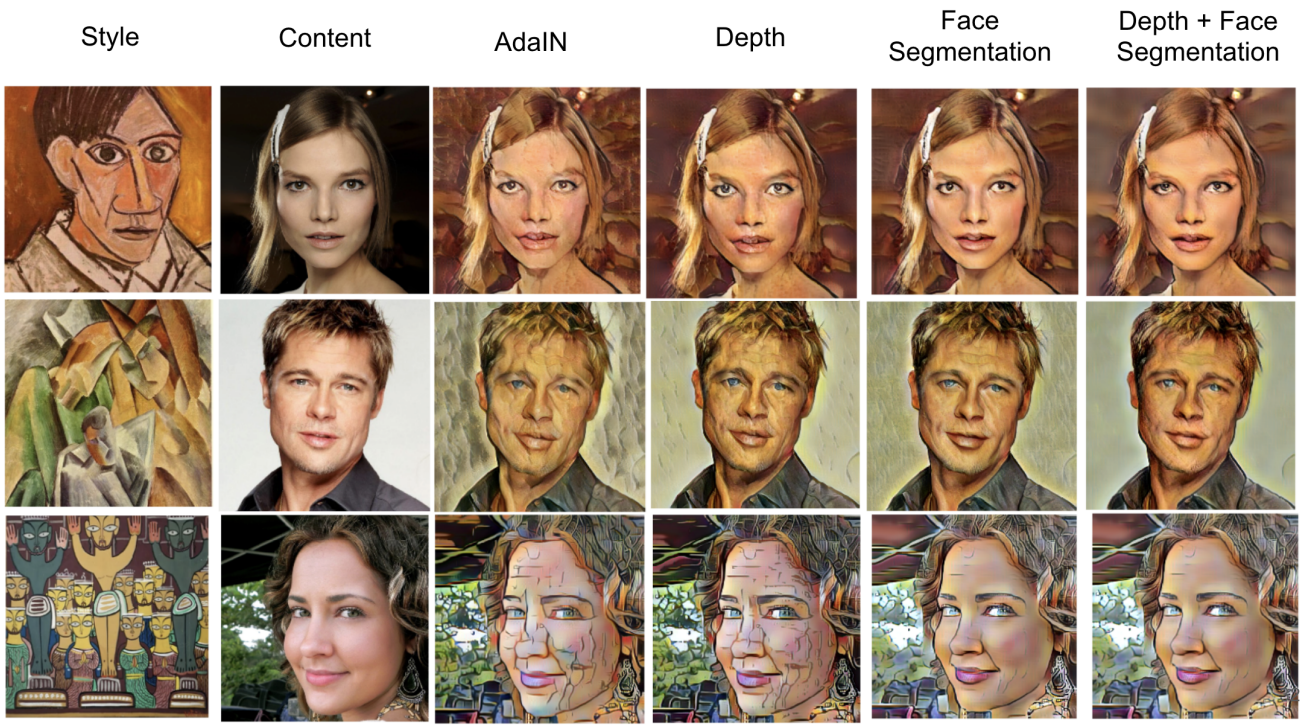

Figure 19. Comparison of results from all approaches. On the left, we have the style images. The first column contains the original images. The second column is the output of the stylized images from the original AdaIN method. The third column is the outputs from the Depth AdaIN. The fourth column is the results from our Face Segmentation AdaIN method. The last column is the outputs from the AdaIN with both Face Segmentation and Depth Preservation.