REPRODUCIBILITY CHALLENGE: ADJUSTABLE REAL-TIME STYLE TRANSFER

Rohan Bungre, Carl Richardson, Charlie Steptoe & Alexander Thomas

ABSTRACT

'Adjustable Real-Time Style Transfer' (ICLR 2020) addresses the inclusion of a secondary conditioner network to adjust hyperparameters without retraining a style transfer network, otherwise requiring hours of computation. This document attempts to recreate the paper's findings, by reproducing figures and describing the difficulties and nuances in reproducing the described network. Relevant code can be found at the Real-Time-Adjustable-Style-Transfer *repository* (git.io/JsT1E)

1 INTRODUCTION

This report evaluates the reproducibility of Babaeizadeh & Ghiasi (2018). Neural Style Transfer traditionally requires retraining of networks whilst varying hyperparameters over many hours to produce sensible results. This recent work optimises a new network that extends the input to include these hyperparameters, thus allowing for real-time tuning to produce variable and more suitable results without retraining. Result reproduction utilised an existing code base for similar problems.

2 REPRODUCTION OF CODE

2.1 ADAPTING EXISTING CODE

To understand the established structure of code for style transfer problems, it was useful to find an existing implementation for previous, similar papers. Investigation yielded an implementation using the PyTorch library, from Nikita Prudnikov. During this work, it was realised that although the basic implementation follows the intentions of the original paper, minute implementation details were overlooked. Incorrect VGG-19 layers were used, with inconsistent use of instance normalisation and exponential moving average compensation on content loss, so manual scale correction was required, the removal of which is the intention of this paper.

Figure 1: Initial Results Comparison

Overall, the original code was found to prioritise style loss and the conditioner network responsible for hyperparameter replacement was less effective than expected, meaning the content image became lost. Regardless, this implementation provided a base to amend and more accurately follow the intended implementation. Example stylised images can be seen in Figure 1, for comparison against the amended version.

2.2 IMPLEMENTATION DETAILS

All references to equations and figures made in this section are with respect to Babaeizadeh & Ghiasi (2018). Details of the network architectures were provided in the appendix. The conditioner was

comprised of 11 dense layers; however, the output function of each layer was not explicitly stated (like it was for the stylizing network). After our first attempt of replicating the model failed, we assumed ReLU activations were used on all but the output layer, as standard for dense layers. The full architecture of the stylizing network was explicitly stated. The conditional instance normalisation, as given by equation 6, was applied to the output of each activation function following all convolutions.

Equation 1 shows the loss was computed by passing images into a pre-trained network and extracting specific features. There were several discrepancies in this section. From how the loss was detailed, it seemed like equation 1 included a mistake and the content loss at layer l should be between the activation of the stylized and content images. Furthermore, figure 2 indicates VGG-16 was utilised whereas section 5.1 stated VGG-19. The implementation details and VGG-19 was implemented. Finally, the implementation details describes the features extracted as 'the last feature set of conv2, conv3 and conv4 layers'. After downloading the pre-trained VGG-19 network from Pytorch, we found these labels didn't map to those used in the module description and we were unable to confidently identify the correct layers and to which 'the last feature set' corresponded. We opted to use the $conv2\_2$, $conv3\_3$ & $conv4\_3$ convolutional layers from the network.

The model was now ready to be trained using the loss and optimisation details specified throughout the paper and in the appendix. However, before this, the training images/data had to be prepared. The paper provided no details of how the images were pre-processed so we decided to re-use the transformations from the existing code. The content images, before input to the stylizing network were resized to 256x256, centre cropped and each pixel value was scaled by 255. This ensured all content images were formatted identically. The style image, before being standardised, had each pixel value scaled by 255. Before passing any image into VGG-19, all pixel values were divided by 255 and each input channel was standardised using the ImageNet means and variances. Regarding the training data, section 5.1 and the appendix provided necessary details, where the appendix details were interpreted as a single epoch using 200,000 batches of size 8.

2.3 Training, Initial Results & Loss Values

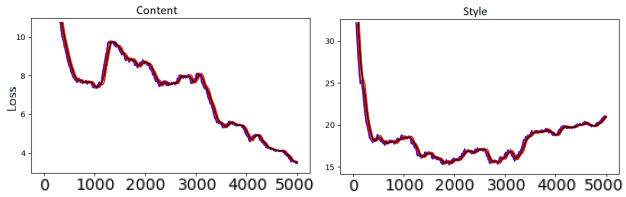

Figure 2: Loss Function Saddle Point at Batch $\approx$ 3000

In contrast to the existing code, when re-implemented, the networks trained steadily, with a decreasing loss value for both style and content. However, at approximately batch 3000 of training, the loss function appeared to reach a saddle point. Beyond this, the content loss dropped dramatically and the style loss rose, as seen in Figure 2. This may be due to sharing of VGG-19 layers between content and style. As a result, supervision was required to ensure this did not occur.

The trained revised network produced considerably more convincing results, though it is arguable that it learned a content loss that was too low, resulting in less variety. The two implementations' outputs can be seen in Figure 1. Each new style image used requires 200,000 epochs to adhere strictly to the ICLR paper ($\sim$ 25 hours), so it was decided to use only the most common style.

3 Figure Reproducibility

3.1 Layer Loss

Figure 7 in Babaeizadeh & Ghiasi (2018) encapsulates main motivation of the original ICLR paper. It is clear that by adjusting the input $\boldsymbol{\alpha}$ vector, the respective loss of that layer of the VGG-19 network should decrease when compared against a style image. This behaviour facilitates the *real-time adjustable* nature of the network. The network was passed an image and the input $\boldsymbol{\alpha}$ vector elements varied individually between zero and one, whilst keeping the other elements constant at

zero or one. The reproduced plot using our implementation can be seen in Figure 3. Note that, unexpectedly some layer losses, in fact, increase with their respective α value. It is not clear why this is the case, but it may correlate with findings of Section 3.3.

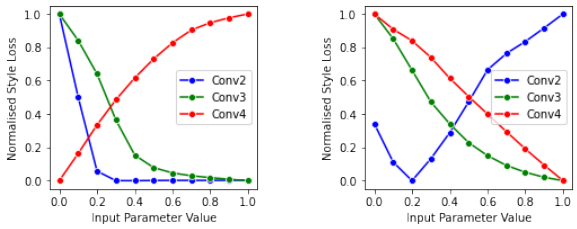

Figure 3: Layer Loss Plots w/ Remaining α_s kept at 0 (Left) and 1 (Right)

3.2 Varying the Image Style

When reproducing Figures 3 and 6 from Babaeizadeh & Ghiasi (2018), several issues as no source images were defined, nor their scale. Testing found that the size of the input image affected the styling of the output image. Using reverse image search, the original test images were found and scaled down to a size of (300x300) for optimal results. The first experiment was to reproduce the effect of adjusting input parameters on stylisation (Figure 3). The stylised output was observed with a given layer weight increasing from 0 to 1, with the rest fixed to 0. The details of each stylisation should vary with weights as deeper layers represent larger features of the style image.

Using Prudnikov's model, the input image was stylised, and intensity of stylisation did increase as with increased weights, shown in Figure 4. However, it is unclear from this reproduction if deeper layers used larger features of the style image to stylise the content. A major issue with this model was that when all style weights were 0, content image reconstruction was imperfect. Using the group's model, the input image was stylised and again, stylisation did increase with increased weights as shown in Figure 4. Unlike the previous model, the deeper layer (conv_3) does use larger features to style the image, when compared to (conv_2). The problem with the group's model was that the deepest layer (conv_4) was not contributing any style to the image.

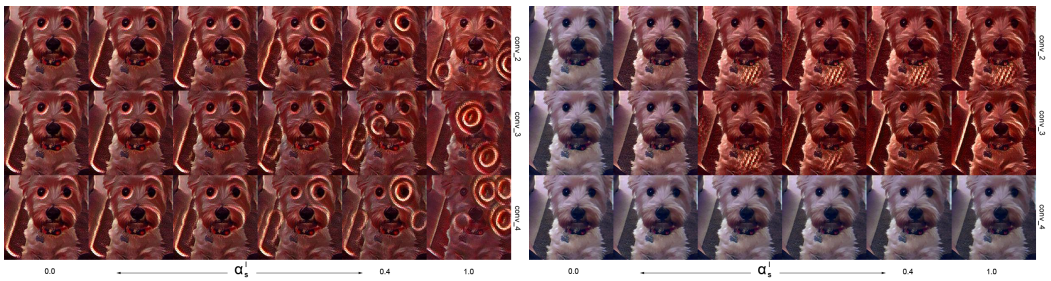

Figure 4: Truncated reproduction of Figure 3 using NP's (left) and our own implementation (right)

The second experiment was to reproduce the effects of combining the styling from several convolution layers (Figure 6). There is no specific result here, however the styling should look different after combining the effects from different layers. Using both Prudnikov's model and the group's model, the input image was stylised based on a combination of layers as shown in Figure 5. Both models did produce distinct stylised images, with Prudnikov's model opting to favour circular style features, whilst the group's model favoured linear features.

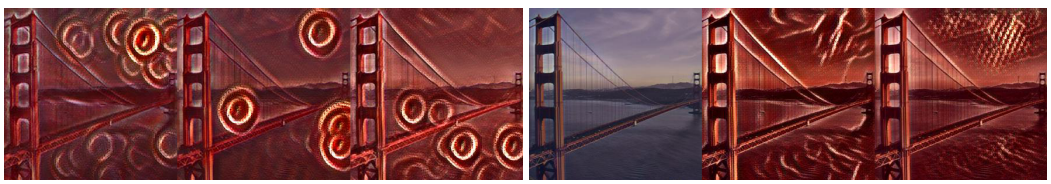

Figure 5: Truncated reproduction of Figure 6 using NP's (left) and our own implementation (right)

3.3 EFFECT OF NOISE

Figure 5 from Babaeizadeh & Ghiasi (2018) demonstrates the effect on the images produced by adjusting parameters of the model, adding a few points of Gaussian noise and doing both of these things. The paper specifies that the noise should be produced by multiplying the input image with a white mask, in which fewer than 10 randomly-chosen pixels have been changed to 0 (black). An inverse Gaussian filter is then applied to this mask. Efforts to reproduce the figure using this method using Prudnikov's code were unsuccessful. The tiny amounts of added noise did not affect the final result as much as the paper suggests, and all produced images were identical. The most likely explanation is that the transformations using Prudnikov's code are not sensitive enough for such small changes to make a difference. Experimenting by randomly adding larger noise spots did produce different results, as shown in Figure 6. Reproduction of the figure using this technique demonstrates that the addition of noise to the input image does affect the image produced, though the images produced are not particularly similar to those produced by the original author and the method is different. The same process was carried out using the new model, as shown on the right in Figure 7. While the inverse Gaussian method of adding noise again didn't make any noticeable difference, there was a substantial difference when adding the black spots as described above. The effect of adding the randomisation is generally less visible using the new model, due to its avoidance of the other model's tendency to obscure details by adding swirls.

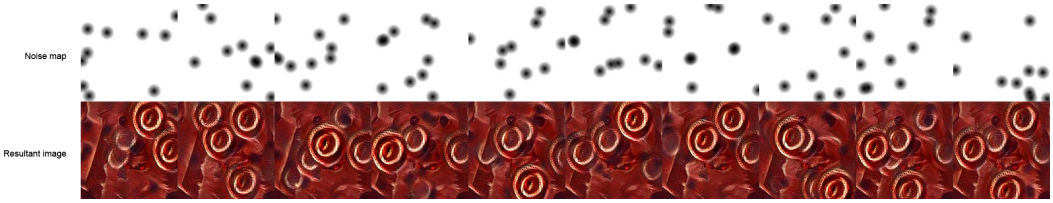

Figure 6: Effect of adding noise to the input image

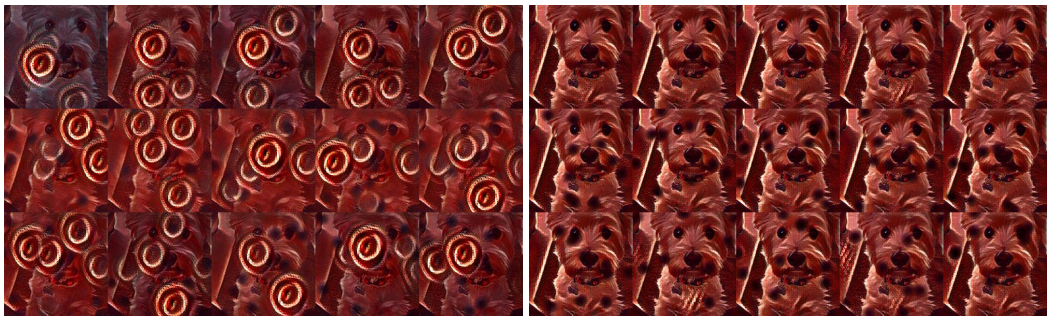

Figure 7: Truncated reproduction of Figure 5 using NP's (left) and our own implementation (right)

4 CONCLUSION

Using the methodology provided by Babaeizadeh & Ghiasi (2018), an Adjustable Real-Time Style Transfer model was implemented, reproducing results to an acceptable level. Likely due to the under-specification in the paper, the reproductions were not completely representative of the original, however they still showed that stylisation of an image can be varied using weights, without retraining the network. Due to computation time to train one style network, testing capabilities were limited when compared to the original paper's.

REFERENCES

Mohammad Babaeizadeh and Golnaz Ghiasi. Adjustable real-time style transfer. *arXiv preprint arXiv:1811.08560*, 2018.