A Neural Algorithm for Artistic Style

E4040.2017Fall.XKCD.report
Vinay Kale vk2392, Siyu Liu sl4262, Yue Wen yw2892
Columbia University

Abstract

Since its creation, Neural Networks have been used for predictive models, object detection and natural language processing. Yet in our paper, we used it to capture the essence of art and recreate images and videos with different styles. One main challenge is how to reserve the original content structure while applying the feature/style obtained from VGG-19, an object recognition pre-trained model. These models are of high perceptual quality, demonstrating the generative power of neural networks trained in purely discriminative fashion. We have successfully explored transferring artistic styles to images, to segmented parts of images, and to videos.

1. Introduction.

Inspired by the movie *Loving Vincent*, which is a biographical animated film about Vincent van Gogh, we wanted to implement the knowledge of Convolutional Neural Networks to create images and videos with specific artistic styles.

A Neural Network transferring artistic styles to another image does not work in the same way as common predictive models. We do not train our network to learn the best weights to give an outcome. Instead, we desire the back propagation to minimize the total loss function to produce an image that has both the stylish feature of the style image and the overall structure and contours of the content image. As *Gatys[2]* pointed in his paper, the goal of matching the content with specific style can be solved through an optimization process. Following his research paper, we first extract the features using VGG-19 pretrained model, define loss functions to denote content representation and style representation as an optimization problem. We explored different optimizers, varying balance of weights, limiting the stylish transfer to specific part of the content image, and extended the application to artistic style processing for videos.

The movie *Loving Vincent* was invested with 5.5 million budget, and its box office is 5.1 million, because of the high expense of a team of 115 painters. A potential use case of our project might be able to help decrease the work of human labours significantly and promote creativity, since the movie can be recorded in traditional form and modified using the Deep Neural Networks later.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

The aim of this paper is to reconstruct two images A and B through generating a new image X which captures the content of A and style of B. The author used output of different layers in VGG19 to reconstruct the content and style.

The pre-trained VGG19 model was trained on more than a million images and has been used in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). The structure of the VGG19 is shown in Figure 1. Gatys suggested we should not use the fully-connected layers of the VGG19 model.

.

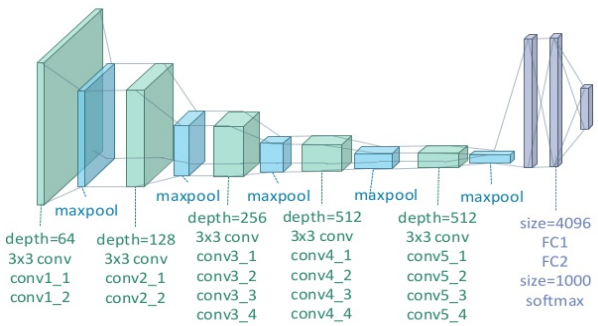

Fig. 1 VGG 19 graph

Each layer in the VGG19 has a group of nonlinear filters. In layer l, there are N_l feature maps with size M_l. Therefore, the information can be stored in a matrix $F^l \in R^{N_l \times M_l}$. Gatys captured content representation by directly calculating the loss while use Gram matrix, which is the inner product space to the style. As shown in Figure 2, Gatys calculated the losses of both of the content and style image and updated the white noise picture to minimize the difference between the input two images. The five pictures in the top line of Figure 2 show the style representation of different layers. We can see that style represented by different layers from the VGG19 yields different level/size of styled features. The images in the bottom of Figure 2 shows re-constructing the input image from different layers.

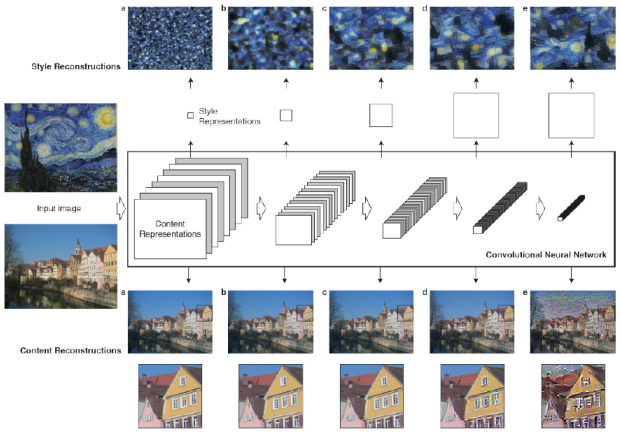

Fig. 2 Structure of Style Transfer Network

2.2 Key Results of the Original Paper

Gatys et al successfully transferred picture off to different artistic styles by finding an image that simultaneously matches the content representation of the photograph and the style representation of the artwork. Demonstrated in Figure 3, Gatys applied five different artistic styles from Turner, Van Gogh and other famous painters to a scenic photograph of Neckarfront in Tubingen, Germany.

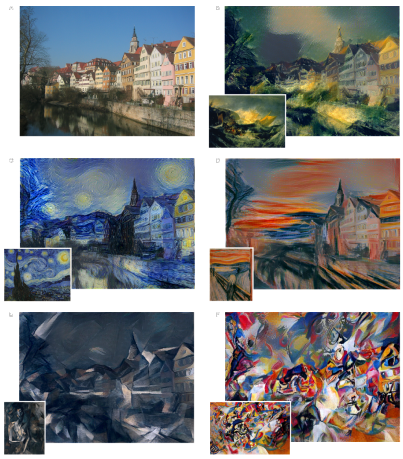

Fig.3 Transferred image of a photograph (top-left image is the original photograph)

Gatys examined how the style representation can be captured with different layers in the pre-trained network. He found that localized structure was better captured with deeper neural networks, as shown in Figure 4. From top to bottom the pictures show increasing depth of layers he used for the style image. The bottom line shows when using all the five convolutional layers, the artistic style captures larger sized features than using parts of the convolutional layers. From left to right the pictures show increasing content/ratio.

Fig.4 Matching styles with increasing layers

In his paper, Gatys also concluded that when he set the content/style ratio to be 0.001 or 0.0001, the image that matches both constraints of the content and style is relatively balanced.

3. Methodology

Based upon the model of the original paper**[2]** , we built our model on the convolution and pooling layers of VGG-19 and tried different versions of style and component values by taking different combinations of convolution layers while calculating the gram matrix and feature maps respectively.

The key finding of the original paper **[2]** is that an image can be represented separately in terms of content and style. Following the paper, we have used convolution layers to extract the style and content feature mappings of the image. As mentioned in the paper, extracting style representation from up to the higher layers of the CNN gives smoother images. Hence, conv1_1, conv2_1, conv3_1, conv4_1 and conv5_1 layers are used for style representation of an image. conv4_2 is used for content representation. Now, a random image is generated with pixel densities distributed according to Normal (0,1) distribution. Using this random image and optimization, an image is generated to have style of one image (art_image) and content of other image (content_image).

Inspired from the work, with an intent to take it a step further, we have also tried to implement the style transfer segmentation for specific parts of an image (ex. face). We try to capture only the required part of the stylized image using a face contour identifier and overlap the image with original content image.

In the techniques above, we actually are trying to maximize the statistical dependencies for feature maps between images. We initially observed our result images to be a bit noisy. Hence, we decided to add a third loss *Total Variation Loss.* It imposes local spatial continuity between the pixels of the combination image, giving it visual coherence. The idea is to maximize the correlation between the images to result in a smooth, coherent generated image. So building upon this idea, we tried to define new loss function which adds the third loss with its own weight to adjust.

$$L_{content} = \alpha L_{content} + \beta L_{style} + \gamma L_{TV}$$

3.1. Objectives and Technical Challenges

The objective of this project was to present an algorithm for combining the content of one image with the style of another image using state of the art convolutional neural networks. Most importantly, the algorithm should allow the user to trade off the relative weight of the style and content reconstruction terms. To do so, we start with a white noise image and solve a minimization problem, discussed in the following section, to find another image that matches the desired content and style. We replicated the stylized image generation model of the original paper [2] and tried to apply the model on various art and content images to ensure its generalizability.

VGG network is a convolutional neural network which rivals human performance on a common visual object recognition benchmark task. Training of this network involves huge time, memory and monetary resources. Hence to get comparable results, we had to load weights of pre-trained VGG-network into Tensorflow. An important challenge is the selection of the hyperparameters which balance the style and the structure in the output image. The performance of the model is rather subjective in nature and requires repetitive experimentation with varying model complexities and parameters to get visually plausible results. Also the challenge lies in the fact that there are endless combinations to check with no clear-cut direction in which to proceed while parameter selection and complexity determination.

3.2. Problem Formulation and Design

We follow the same methodology as [2] to formulate the minimization problem.

Using the VGG-19 as [2] proposed, we removed the fully connected layers, keeping only

the 5 pooling layers and the 16 convolutional layers. The trained weights of this network are publicly available online.

This network is used to encode an image at each convolutional layer using its filter response. In this way, as previously mentioned, a layer l that has N_l filters, will have feature maps each of size M_l which corresponds to height x width of the feature map. We can store all the responses of layer l in a matrix $F^l \in \mathbb{R}^{N_l \times N_l}$, so $F^l_{i.j}$corresponds to the activation of filter i th at position j and layer l .

Let p and x be the original image and the generated image, and p^l and F^l the respective feature representations at layer l, in order to generate the content of the original image, we need to minimize:

$$L_{content}(p, x, l) = \frac{1}{2}\sum_{i.j}(F^l_{i.j} - P^l_{i,j})^2$$

where the gradient with respect to x can be computed using back propagation.

Having defined how to generate the content of an image, a way to generate a style representation is needed. To do this, the correlation between different filter responses is computed, taking the expectation over the spatial extent of the input image. This is given by the gram matrix $G^l \in \mathbb{R}^{N_l \times N_l}$, where $G^l_{i,j}$ is the inner product between the feature maps i ,j in vector form, in layer l

$$G^l_{i,j} = \frac{1}{2}\sum_{l=0}^{L} w_l E_l$$

Having defined this, let α and x be the original image and the generated image, and A^l and G^l the respective style representations at layer l , in order to generate the style of the original image, we need to minimize:

$$L_{style}(a, x) = \frac{1}{2}\sum_{l=0}^{L} w_l E_l$$

where w_l is the weighting factor of the contribution of each layer and E_l is the contribution of layer and it's defined as:

$$E_l = \frac{1}{4N_l^2 M_l^2}\sum_{i,j}(G^l_{i,j} - A^l_{i,j})^2$$

Here gradient with respect to x can be computed using back propagation as well. The original paper chooses $\frac{1}{layers\ to\ use}$for the convolutional layers. We decide to use the same for the included layers and 0 for the layers we don't want to use.

Now that we have a way to generate an image that separately matches content of one image and style of another image, we want to bring these two components together to create an output image that combines both content and style. We achieve this by jointly minimizing the distance of the random image x to the styling image a and the content image p.

Finally, in addition to content and style loss, we include an additional total variation loss, also called TV loss. This loss allows to denoise the outcome image by ensuring a smooth variation between adjacent pixels. The expression for the TV loss is:

$$L_{TV} = \sum_{c=1}^{3}\sum_{i=1}^{H-1}\sum_{j=1}^{W-1}(x_{i,j+1,c} - x_{i,j,c})^2 + (x_{i+1,j,c} - x_{i,j,c})^2$$

Thus, the overall loss we aim to minimize is:

$$L_{content} = \alpha\, L_{content} + \beta\, L_{style} + \gamma\, L_{TV}$$

Through back propagation, after several iterations this yields an output image x that combines content from p and style from a. The following image summarizes how to generate an image mixing content and style.

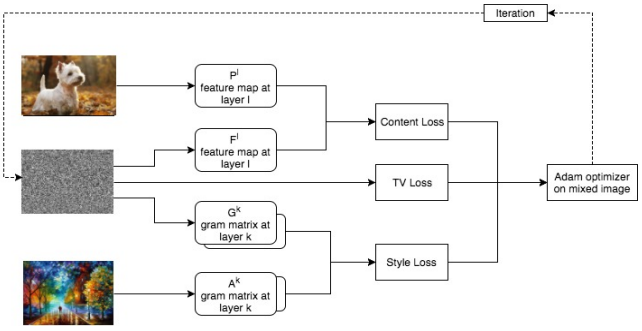

Fig.5 Summary flowchart

4. Implementation

In the following section, we describe the deep learning architecture, then we describe the overall design of our implementation, and details about challenges and considerations of it.

4.1. Deep Learning Network

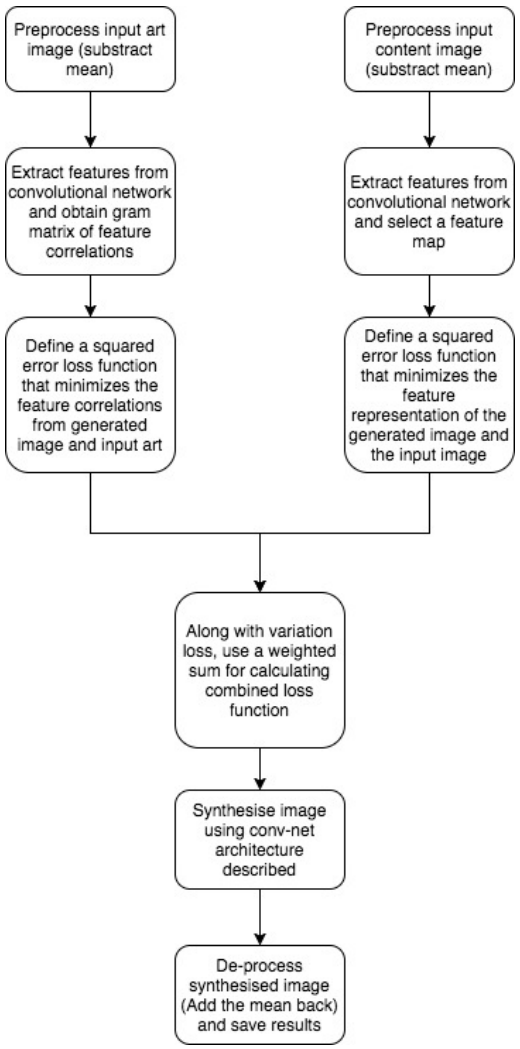

Fig.6 Architectural block diagram

The architecture of VGG-net is shown in the appendix. For SGD based gradient descent updates, we employed the ADAM algorithm for training the convolutional networks

Adaptive Moment Estimation (Adam) is an optimisation method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients similar to momentum. The gradients are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method. The pseudo code for the method is written in the next section.

Regarding the data, we have used different styles such as well as content picture. Besides, we also used a short video as input data.

4.2. Software Design

a). Pseudocode for ADAM

Algorithm: ADAM

Input:
Step- size α*;*
decay rates ρ_1 *and* ρ_2*;*
constant ε*;*
Initial parameter θ

Algorithms:
Initialize 1^{st} *and* 2^{nd} *moment variables*
$s = 0, r = 0.$
Initialize time step
$t = 0$
While stopping criterion not met do:
Sample a minibatch of m training set examples $\{x_1, x_2, \ldots, x_m\}$

Set $g = 0$

for $i = 1$ *to m do:*

compute gradient:
$g \leftarrow g + \nabla_\theta L(f(xi, \theta), yi)/m$
end for
$t \leftarrow t + 1$
get biased first moment:
$s \leftarrow \rho_1 s + (1 - \rho_1) g$
get biased second moment:
$r \leftarrow \rho_2 r + (1 - \rho_2) g_2$
compute biased- correct first moment:
$\hat{s} \leftarrow \frac{s}{1-\rho_1{}^t}$
compute bias-corrected second moment:
$\hat{s} \leftarrow \frac{s}{1-\rho_2{}^t}$
compute update:
$\Delta\theta \leftarrow - \alpha \frac{s}{\sqrt{\gamma} + \varepsilon} \odot g(element - wise)$
Apply update:
$\theta \leftarrow \theta + \Delta\theta$
end while

It is proposed by the authors of Adam that the values of 0.9 for ρ_1 , 0.999 for ρ_2, and 10^{-8} for ε. They showed empirically that Adam works well in practice and has advantage over other adaptive learning method algorithms.

Here we will summarize the pseudocode for the complete algorithms.

b). Pseudocode for complete algorithm

Algorithm: Generate Image

Inputs:
p -- content_image
a -- style_image
alpha -- weight of the content loss
beta -- weight of the style loss
gamma – weight of the TV loss
Output:
x – generated image with content from image p and style from image a

vgg = VGG19() #create a vgg network with pretrained weights
p_layers = ['conv1_1', 'conv2_1', 'conv3_1', 'conv4_1', 'conv5_1']
a_layer = ['conv4_2']

p_vgg = vgg(p) # obtain conv. layers of p
a_vgg = vgg(a) # obtain conv. layers of a

p_feat = [p_vgg.output(layer) for layer in p_layers]
a_feat = a_vgg.output(a_layer)

x = generate_white_image()

x_feat = [vgg(x).output(layer) for layer in p_layers and a_layer]

p_loss = content_loss$(a\_feat, x\_feat', conv4\_2')$
a_loss = $\sum_{l \in p\_layers} style\_loss(a\_feat, x\_feat)$

loss = $\alpha\, L_{content} + \beta\, L_{style} + \gamma\, L_{TV}$
 # added variational loss too

$x = argmin\ loss$
minimize the loss function using the desired minimization algorithm

return x

The content_loss, style_loss and the variational_loss function are defined in Section 3.2.

5. Results
5.1. Project Results
a). Results presentation

In this part, we will present the trained pictures of extracting different styles of pictures and apply those styles to a picture of a dog

Fig.7 dog picture as content picture

Fig.8 style picture (shipwreck)

Fig.9 trained picture

Fig.10 style picture (ben giles)

Fig.11 trained result picture

We can see that the trained pictures are very decently generated and visually appealing from the right choice of the parameters. In later part in this paper, we will present how different choice of parameters might influence the final result.

Besides, iterations of the algorithms as well as the choice of optimization methods will also influence the results. Above pictures are generated from 1000 iterations and Adam optimization methods. We noticed that if we want to improve the quality of the final output, increasing the number of iterations might help, but usually 1000 will be sufficient given the size of the input pictures. We also want to mention that at some point, the Adam optimization algorithm might get stuck and it decreases the loss function in relatively small amounts.

b). Segmentation

In the previous work, we have applied the style we extracted to the whole picture. Besides that, we can only apply the style to part of the picture, which gives our algorithms more flexibility in terms of application.

So firstly, we segment one picture into two (or more) based on some certain edge (or other standard). Then we train the original whole picture. Finally we overlap part of the picture with the trained picture.

The results of segmentation are represented as followed:

Fig.12 style picture (the starry night)

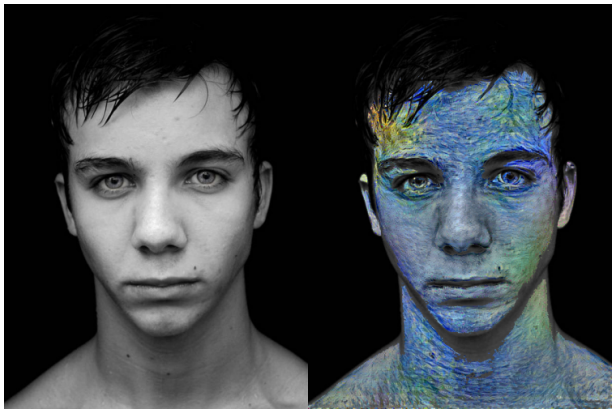

Fig.13 result of segmentation

Fig.14 style picture (the scream)

Fig.15 result of segmentation

As we can see from the result, we only changed the facial part and left the other part as original.

c). Video

Besides applying the model to a single picture, we can also implement it to a video.

The principle behind it is very simple, we use Python cv2 library to read frames (24 frames per second) from the video, and then run the algorithms for every single frame of the video.

Eventually, we combine those frames back to a modified video using the same library again.

For implementation, we chose a short video of around 30 seconds, and then trained around 700 pictures to get the final modified video using the famous painting of Monet.

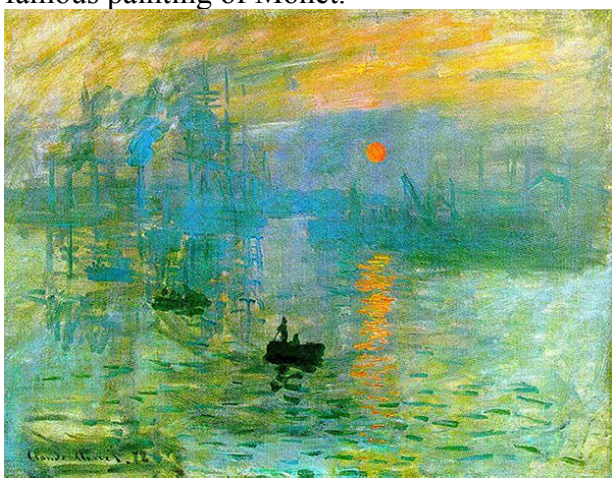

Fig.16 Monet style picture

It takes tens of hours using GPU in GCP to finish the training process. We uploaded the trained video to bitbucket.

5.2. Comparison of Results

a). Reproduce paper results

In this part, we recreate the example from the paper which used *The Starry Night* by Van Gogh as style.

The result of the original paper is displayed below:

Fig.17 the result picture from the original paper

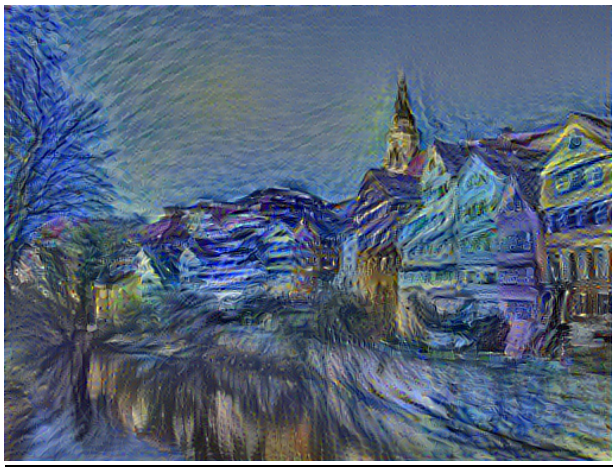

Fig.18 the picture we reproduced (with increasing style weight) first

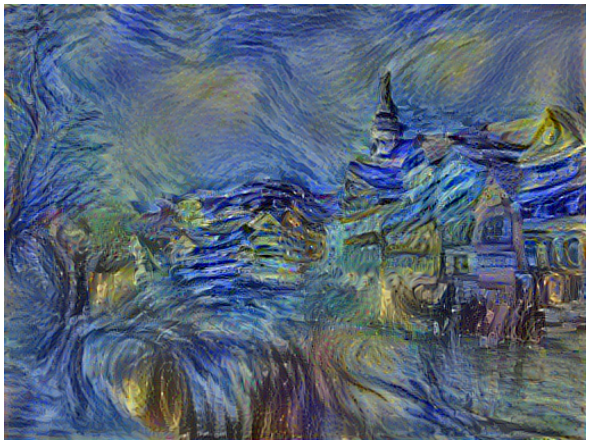

Fig.19 the picture we reproduced (with increasing style weight) second

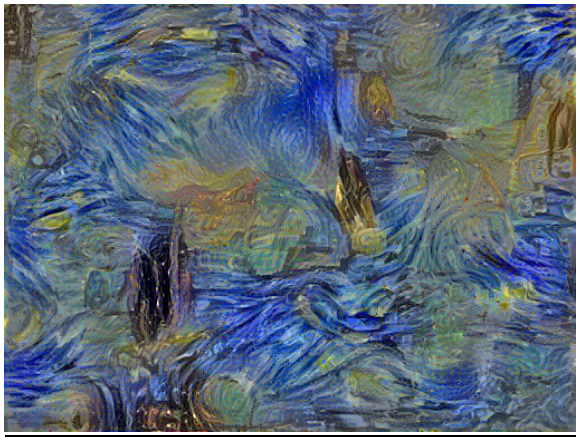

Fig.20 the picture we reproduced (with increasing style weight) third

We have presented the results with increasing style weights. We can see although it is not quite the same from the picture generated from the original photo, but the picture generated still seems to well capture the content and the style as well as to be visually appealing.

There could be several reasons that caused the difference from the original picture. Firstly, it can due to the different resolution of the input picture. Secondly, it can be caused by different iterations of the learning procedure. And lastly, and most probably, since we start the algorithm with a random white noise picture, as the optimization algorithms proceed, different initial value might lead to different local minimal points.

b). The results generated from different weights of style and content

In this part, we will present how different choice of hyper parameters will influence the final visual effect of the pictures. We sill use the dog picture we used in the previous part, and use *the starry night* as the style picture.

Too much content(α： β = 1： 10)

Fig.21 picture generated with too much content

Content style balance(α： β = 1： 100)

Fig.22 picture generated with right ratio

Too much style(α： β = 1： 1000)

Fig.23 picture generated with too much style

As we can see from the previous three pictures, when we set α： β = 10, the result picture is able to capture both the style and retain the content of the dog. However, if the ratio is too high or too low, the results are less satisfying.

c). The results generated from using different number of convolutional layers to extract style.

As discussed in the paper, the five convolutional layers are used in extracting the style of the pictures. We are curious at the what results would look like if only we use two of them. The final results are showed below:

Fig.24 the lion content picture

Fig.25 the style picture (the portrait of doora maar)

Here is the picture generated using all the five layers.

Fig.26 the stylized picture using all the conv layers

Here is the picture generated using only two of the five layers.

Fig.27 the stylized picture using only two conv layers

As we can see from the picture, if we only use two layers, the algorithm did not extract too much style from the style picture. On the other hand, if we use all the layers, the algorithm would be able to extract more style of the style picture.

5.3. Discussion of Insights Gained

Since the project was not a traditional prediction problem, we could only apply the learnings of the course in a limited manner. Despite that, we have learned some important lessons while working on this project.

a). Computation cost

It's important to note that the time for training can grow rapidly with the desired size of the image. If we wish to get a very high-resolution image, we will have to minimize a loss function on millions of parameters. Hence, it's important to choose an algorithm that converges fast as well as using the GPU and cloud computing resource.

b). Balance of style/weight parameters

We also noticed that the choice of α and β can change significantly the result of the algorithm. For implementation, we can choose different parameters to control how much content we want to reserve.

c). The effect the total variance loss

Total variance loss helps makes the picture looks smoother. The idea is to maximize the correlation between the images to result in a smooth, coherent generated image. The total variation loss imposes local spatial continuity between the pixels of the combination image, giving it visual coherence.

d). Segmentation and video

Given the fact that we are able to combine the style of one picture and the content of another picture, we implemented our algorithm to all possible extent in the limited amount of time we have. The segmentation use-case can further be tuned to capture other parts than the image and

for the video, one can use the continuity between frames to give a more visually appealing scheme to overall video.

e). Style layers

We observed that we get better style representation as we include higher layers of the network for style feature calculation. This is primarily because the local image structure is better captured when including the contribution of higher layers of the network for calculating style features.

6. Conclusion

Using the methodology discussed in the original paper and a pre-trained VGG network, we are able to separate and recombine content and style of arbitrary images, to generate artistic images of high perceptual quality. Local image structures captured by the style representation increase in size and complexity when including style features from higher layers of the network. This can be explained by the increasing receptive field sizes and feature complexity along the network's processing hierarchy. We also implement the methodology to segments of the picture and video.

7. Acknowledgement

We would like to thank Professor Zoran Kostic for his indelible teachings and all the TAs of ECBM4040 (Fall 2017) for their constant guidance and support in coursework, assignment and project.

7. References

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

[1]https://YueWen1994@bitbucket.org/ecbm4040/2017_assignment2_yw2892.git

[2] A Neural Algorithm of Artistic Style. *Leon A. Gatys, Alexander S. Ecker, Matthias Bethge*. https://arxiv.org/abs/1508.06576

[3] Photo Style Transfer in Tensor Flow, *Aly Kane, Amelia Lemionet, Fjori Shemaj* http://cs231n.stanford.edu/reports/2017/pdfs/412.pdf

8. Appendix

8.1 Individual student contributions - table

	CUID1	CUID2	CUID3
Last Name	Kale	Liu	Wen
Fraction of (useful) total contribution	1/3	1/3	1/3
What I did 1	Write the code	Write the code	Write the code.
What I did 2	Train the pictures needed for the project.	Train the pictures needed for the project.	Train the pictures needed for the project.
What I did 3	Write the report.	Write the report.	Write the report.

8.2 VGG19 architecture