Object Recognition and Image Understanding Project Style transfer for images and videos

Florian Kleinicke

July 30, 2018

1 Introduction

As neural networks advance and become more useful it's getting interesting to model the internal information insight of a network. The here used effect uses this internal representation of two images, the content and the style image, in a neural network.
These internal representations in the different layers are also displayed in figure 2. This representation alone is already very interesting to gain insights into neural networks. But here these methods are used not only to gain understanding, but furthermore to create artistic visualizations.
The outcome is a stylized image, that is in certain aspects interpreted similarly by the neural network, then the two original images. The two input images and a corresponding outcome can be seen in figure 1.

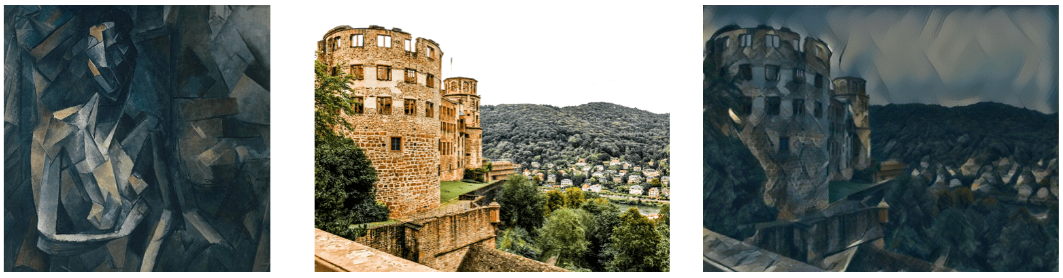

Figure 1: The stylized image (right) is composed out of the style image (left) and the content image (middle).

I used the algorithm from the paper "A Neural Algorithm of Artistic Style" by Gatys et al. [3] to compute such a style transfer. Later on applied the same algorithm with small modifications on video as described by Ruder at al. in "Artistic style transfer for videos" [6].

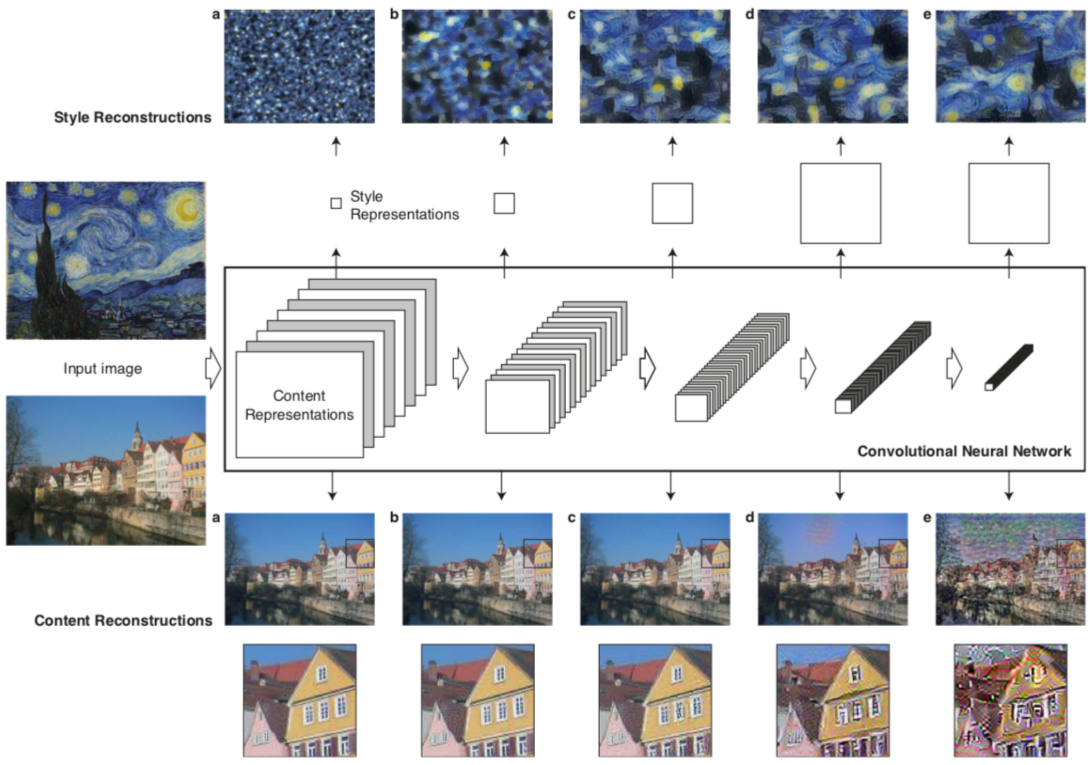

Figure 2: The information from the different layers compared. In the upper row, the style is computed for the first 5 layers. The first layer still contains quite detailed style. In the latter layers, the style getting more focused on the big recurring stylistic objects. The bottom row displays the remaining style of the layer. Its shown how much of the content of the original image remains in the certain layers. The image is getting further compressed into it relevant parts. In the fourth layer, the main content is still available, while the irrelevant details (style) are getting less important. The images here were created by applying the (ref) proposed loss function only for one single layer and only for the style or the content loss.

2 Image style transfer

Summary The goal of a style transfer is to generate an image, that displays the scene of a content image with a changed style. The new style is given by a style image. To achieve this I used the method of Gatys et al. [3]. In a brief review, the internal representation in a neural network of a style image, content image, and a generated image are compared.

From the difference, a content loss function is formulated. For the style loss in multiple convolutional layers, the correlation of the activations is computed and compared for the generated image and the style image. In the end, the two loss functions are added with some weight and the generated image is optimized with back propagation to achieve a minimal loss function. This way a stylized image, that contains the style of one image and the content of another image is generated.

But now in more detail For initialization, the generated image (later stylized image), is just random noise, or a copy of the content image. As a neural network, a pretrained VGG19 network [7] is chosen. To compare the difference between the generated image and the content image in the VGG 19 network, the activation in a selected layer is directly compared.

$$\mathcal{L}_{content} = \frac{1}{N_l M_l} \sum_{i,j} (F^l_{ij} - P^l_{ij})^2 = MSE(F^l, P^l)$$

With F^l representing activations of the l'th layer in the network of the stylized image and P^l the activations of the l'th layer of the content image. The dimensions of these layers are $N_l \times M_l$. The here used layer is the `conv4` layer of the VGG19 network, since there is still enough detail left, but not the less relevant information of the image is already removed. This is because the network creates an internal representation of the image, and focuses more and more on the relevant details. What is considered a relevant detail, depends on the training. Here the network is trained to detect certain objects, so the relevant information what these objects look like, is still represented in the deeper layers. The comparison of the different layers can be seen in figure 2.

In the same figure, you can see To compute the loss function for the style, that the style of an image is stored in the correlation between the single neurons of a layer. Therefore the Gram matrix is calculated.

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$

F^i stands for the filter response of layer i, so F^i_{jk} is the activation of the j-th filter at position k. The Gram matrix is computed for the style image and the generated image. In the equation below the matrix is called G for the generated image and A for the style image and has the dimension N times M. Each layer is weighted and the results are summed up for the total style loss function.

$$\mathcal{L}_{Style} = \sum_{l=0}^{L} \frac{w_l}{4N_l^2 M_l^2} \sum_{i,j} (G^l_{ij} - A^l_{ij})^2 \underset{\mathrm{w_l=4}}{=} \sum_{l=0}^{L} MSE(G^l, A^l)$$

I used the first 5 convolutional layer to compute the style loss. To be exact, it were the activations of the layer `conv1`, `conv2`, `conv3`, `conv4` and `conv5` of the VGG19 network. To compute the final loss, the two content and the style loss are added.

$$\mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{Style}$$

Here I choose $\alpha = 1$ and $\beta = 1000000$.
The generated image can be improved using backpropagation and the here derived loss function. Depending on the resolution of the image, a few iterations already lead to ok results when initiating the generated stylized image with the content image. For the resolution of 350 times 525 pixel 300 iterations lead to good results, while the results usually don't improve anymore after 1000 iterations. At 300 iterations some important style details might still be missing.

3 Video style transfer

Several improvements can be made to adapt this technique to videos. The techniques used here were proposed by [6].

The trivial approach would be to apply the style transfer to each individual frame, initiated with random noise. Since there is already a stylized frame from the previous frame, I was able to initialize the stylized new frame with the stylized last frame. This is easy to implement and will improve the quality of the result, assuming that not much changes between the two frames. The second generated frame requires fewer iterations because the initiation is already very good and some information about how the last image was stylized is transferred to the next image. This helps to reduce the change between to consecutive stylized frames. If it is generated from scratch each time, optimizing the loss can lead to completely different results. This could mean that a flickering of suddenly changing styles becomes visible at certain points.

Optical flow Another technique to further improve the results uses the optical flow that was introduced in our lecture. The flow describes the movement of similar looking pixels from one frame to the next frame. With this flow the old stylized frame that is used as an initiation can be warped to get a new, better initiation, that will represent the expected stylized result better. For the next task not only the forward flow is computed, but also the backward flow. Therefore the flow from f.e. the second frame to the first frame is computed.

Temporal loss The third improvement I used was a temporal loss. During the training, this loss ensures that the new frame changes only within some limits compared to the previous stylized frame. Therefore the style should not change too much at certain places. This method can only be applied, when there are no cuts in the video. For cuts or too strong movements, an additional function is proposed. I'll check whether the forward flow of the last and the backward flow on the current frame are similar, and save the information in a consistency mask. The updated loss function now reads.

$$\mathcal{L}_{total} = \alpha\mathcal{L}_{content} + \beta\mathcal{L}_{Style} + \gamma\mathcal{L}_{temporal}$$

With the temporal loss

$$\mathcal{L}_{temporal} = \frac{1}{D} \sum_{k=1}^{D} c_k (x_k - w_k)^2$$

The c_k is an element of the consistency mask for the flow, x_k a pixel of the generated image and w_k a pixel of the warped stylized image of the last frame. The loss sums over all D pixel of the image, and ignores the pixels, where the flow isn't consistent.

4 Implementation

I implemented the algorithm using pytorch. There is a good tutorial with an explained code on the pytorch website about style transfer [2]. I took this code as a base for my further experiments and saved it under the name `stylize.py`. It makes use of the pretrained VGG19 network, that is part of `torchvision.models`. I added scripts and functions to apply the algorithm to the images I had. Therefore I updated the way, that arguments are passed, how images are loaded and that the content and style images are internally reshaped to the same size. The output isn't shown with matplotlib anymore but directly saved to the hard drive. This way it's also easier to run it on a server. The style transfer for images is started with the bash script `fotostyle.sh`.
I tried several configurations, different styles and contents, different resolutions and number of iterations.

Video style transfer The original paper was implemented using torch as a framework. [5] Therefore it was possible to see in the code what they were doing, but due to the different platforms and functions that can be used not really helpful most of the time. Another implementation I found was with tensorflow [8]. This implementation was clearer in most parts, but still used a lot of functions that aren't provided with pytorch.
The main problem while implementing the paper was, that it was described what should be done, but not actually how to do it. The most time consuming was the implementation of the image warping, and to figure out how to compare the forward and backward flow. Even the mentioned preexisting implantations weren't helpful in these points since they chose existing functions or really complicated approaches to solve these problems.
Since the in the paper proposed method to compute an optical flow wasn't available for pytorch I used OpenCV [1]. To run the video style transfer I used the script `transferStyle.sh`. In this script, first, the resolution of the output video is determined. Then the original video is split into its frames and the optical forward and backward flow is computed. To compute the optical flow the file `create_flow.py`, that uses the methods of the file `flowmethods.py`. If you run this file, forward and backward flow files are created and saved.
Afterward, the style transfer for the first frame is computed. Depending on the settings, the stylized image for all the other frames are computed and at the same time the

optical flow is applied to the stylized frames (in `apply_flow.py`, so they can be used for initiation and temporal loss.
In the end, all the stylized frames are unified to a video.

5 Results

Images The results are really convincing. I applied the style of multiple well and less known paintings and images on photographs I took. Nevertheless there were some bigger problems. Since I used the cip-server, the graphics memory that could be used was limited to just 2 GB. When I chose the resolution too high, there was a graphics memory error after some time. Therefore I confined the resolution of the images to 525 to 350 pixel, and in some cases to 600 times 400 pixels.
The second problem was that the runtime was really long. For a 525 times 350 pixel image, the stylization took about 30 minutes.
After some testing, I noticed that the style has to fit the content. In figure 3 you can see, the effect of a style image with sharp lines. If the content image has some straight lines, these are amplified, which might lead to an impressive effect. If these straight lines are missing, then they are drawn at places, where they don't fit so well. Never the less the stylized image of the cat still looks really cool. Roundish styles, usually fit better with more content images.
During the training process, the content and style loss usually improve in the long run. But sometimes the stylized image gets very bad and essentially restarts with a noise image after a backward propagation. I didn't fully understand why this happens, but this happened when the optimum was reached for some time. Since I tried the first test with 2000 iterations, where these problems frequently occurred, I lowered the number of iterations to 1000, which slightly decreased the number of useless outcomes. A collection of stylized images I created, is handed in, in a folder next to the code. You can see eight of them in figure 4

Videos Since the approach in a good resolution is really time-consuming I focused on experiments on one video. In the video, a friend of mine jumps and after his landing the camera moves up. I chose this video, since it has a clearly moving person, with everything else staying in the same spot. In the end, the movement of the camera is challenging. This way the video contains an easy and a harder scenario. The video was cut to a length of 7 seconds and contains just above 100 frames.
The first step was to compute the optical flow. I made a video of the optical flow, called `opticalFlow.mp4`. To test the optical flow, and to visualize its weaknesses I applied it to the original video frames first. The video `contentFlow.mp4` looks like the original video but consists only of the original frames, that had been warped to represent the next frame. The warping had been always applied on the original frame and never on the previous warped frame. There are actually a few differences visible, compared to the original video. Whenever there is some movement, the objects around the moving object

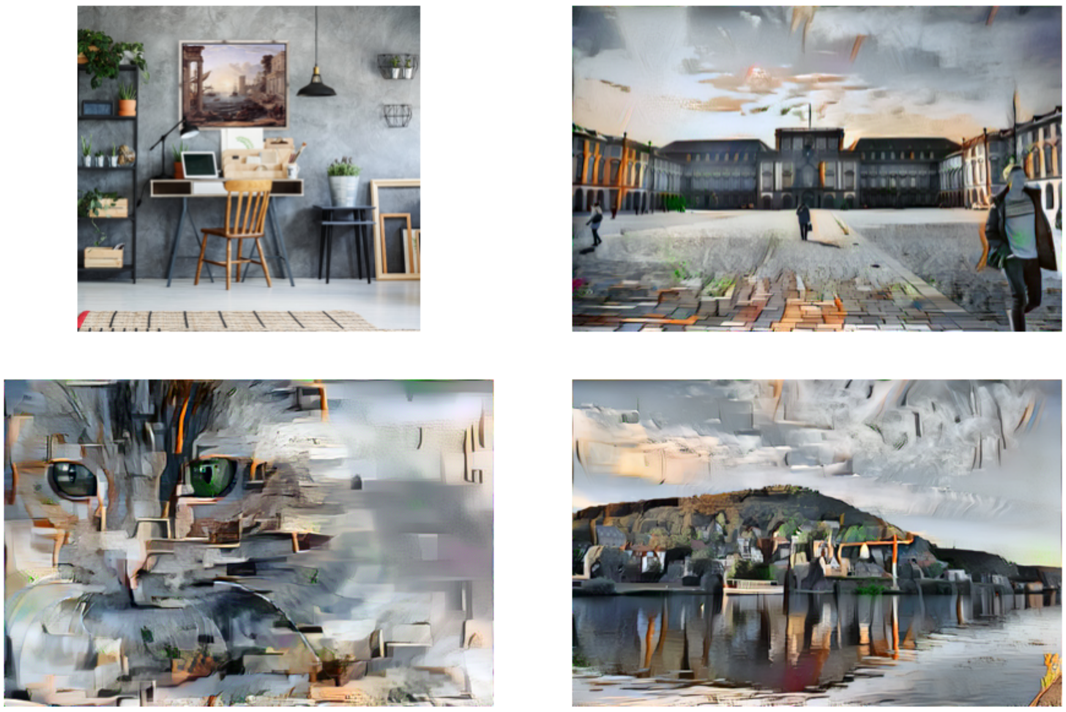

Figure 3: The style has to fit the content image. Here the horizontal and vertical lines of the style image (upper left) fit well to two of the content images (right side, Mannheim Castle, Neckarfront in Heidelberg). The application on the image with the cat has some interesting effects, since the different style adds straight lines into the image.

are warped too. This becomes clearly visible when he passes by the tree, is moving his hand before his face, or during his landing. At one point his arm seems to be really thin and his leg is really big. In figure 5 these movements are visible in the three images of the left. Therefore we expect that these small errors might lead to problems if the algorithms rely too much on the flow.

Stylized videos After these first tests, I applied the algorithm in multiple configurations on the video. For the first test, I used a resolution of 350 times 525 and put the number of steps for the first frame to 600 and for all the other frames to 300. I chose the resolution as high as possible, so there are more details left and the effects are clearly visible. Then I created the stylized videos with different configurations. In video `noFlowInit_noTemporalLoss.mp4` I didn't use the temporal loss and initiated each frame with the previous stylized frame, without applying the flow. In the video `noFlowInit_temporalLoss.mp4` I still used the previous stylized frame as the initiation image, but this time I also used the temporal loss. In the video `flowInit_temporalLoss`

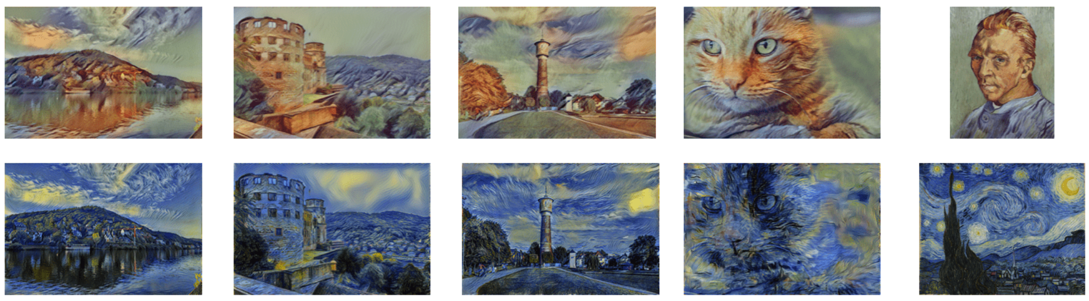

Figure 4: The resulting style transfers of two styles and 4 different content images. The style in both rows is from Vincent van Gogh. In the first row, it's from his self-portrait (right column), in the second row it's from his painting the starry night. In the first column, the style is the Neckarfront in Heidelberg, the second column is the castle of Heidelberg, the third column the tower of Ladenburg and the fourth column a cat.

I warped the initiation image with the flow and used a temporal loss.
A few things are visible. First the style much more present in the first frames, than in the last frames. In the beginning, there are still brush strokes visible, at the end they are completely gone. The yellow spots, that were applied by the style stay at the same spots, even when the camera is moving up. The camera pan is indeed the hardest task for the algorithm in the video. It looks kind of weird in all three cases. But there were also some unexpected results. Using the warped stylized frame as an initiation does seem to lead to worse results. A transparent version of the background seems to stay in the same position when the camera is moving. This is what originally should be prevented by the image warping. Another interesting observation was, the faster the person was moving, the more transparent he became. This may be because 300 iterations might not be enough to optimize the image at the changing positions. Therefore its really hard to judge the effect of each method, since they all perform different with different number of iterations and different resolutions.
In order to get to grips with these problems, I have run two more experiments. The first one with a slightly reduced resolution, but with 500 iterations at each frame. I used no warp on the initialization but used temporal loss. The result didn't differ by much to the previous run, as you can see the file `moreSteps.mp4`. The level of detail is still higher in the first frames.
In the end, I run another experiment called `noInit.mp4` where I initiated each frame with its content frame. The results are as described in the original paper. A strong flickering of different positions of the style is visible.

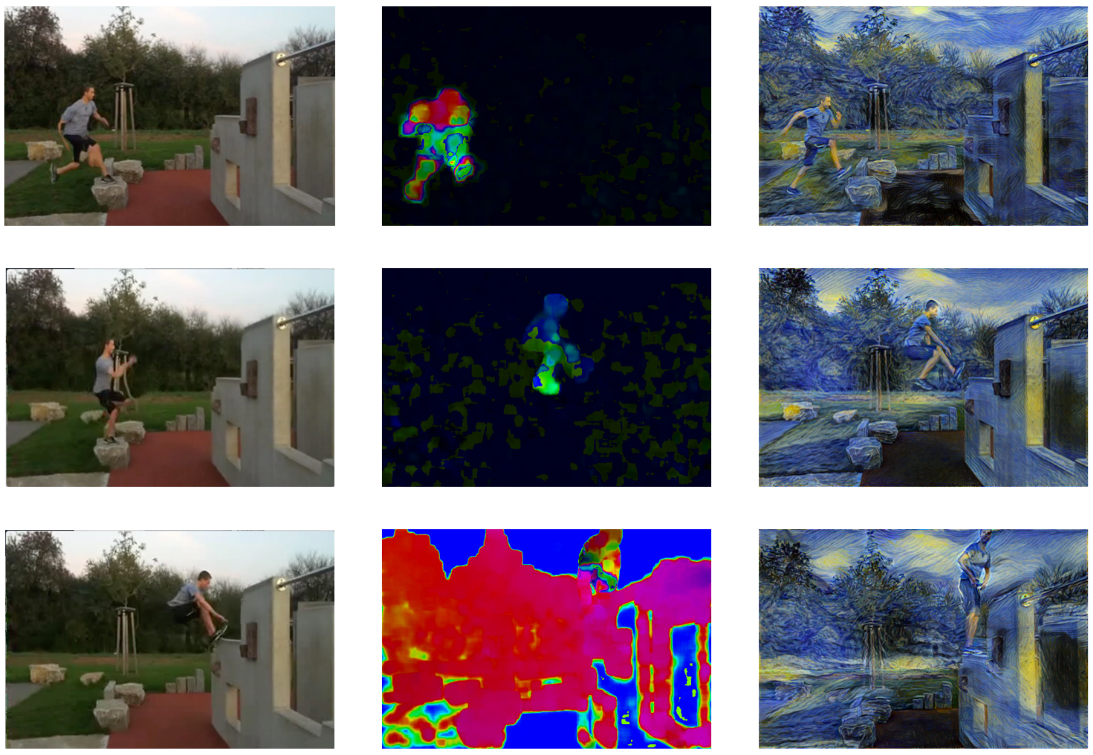

Figure 5: The three images on the right show the original video, warped by the forward flow. I chose frames where some problems with the flow were visible. At the first row the arm is thinner and the leg is thicker than usual, in the second row the tree is warped and in the bottom row the wall is slightly curved. In the second columns there are shown three different optical flow frames. In the first two frames a moving person is visible. In the last frame the camera is moving. In the third row are the corresponding stylized frames to the second row. These frames got stylized with the image The Starry Night by Vincent van Gogh.

6 Future improvements

The main problem with the implementation of videos was the performance. Since the same style has to be applied multiple times on the different frames, it might be a good idea, to use a faster version of a style transfer. Johnson et al. [4] proposed a neural network, that learns the style, that can stylize single content images really fast. This could be utilized for videos.

References

[1] Dense optical flow in opencv. `https://docs.opencv.org/trunk/d7/d8b/tutorial_py_lucas_kanade.html`.

[2] Neural transfer with pytorch tutorial. `https://pytorch.org/tutorials/advanced/neural_style_tutorial.html`.

[3] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

[4] Justin Johnson, Alexandre Alahi, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016.

[5] Manuel Ruder. artistic-videos, github. `https://github.com/manuelruder/artistic-videos`.

[6] Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. Artistic style transfer for videos. *CoRR*, abs/1604.08610, 2016.

[7] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[8] Cameron Smith. neural-style-tf. `https://github.com/cysmith/neural-style-tf`, 2016.