Style Transfer from Artworks to Photographs

Custom project by

Marin Karamihalev
Siyana Ivanova
Jakub Blaszkowski
Jae Sun Lee
Rafal Ziomek

MLW 2019

Background

Our project consists of a program which performs style transfer from one image to another: in simple terms, it takes two images as input - one from which the style (style image) will be extracted and one on which it will be applied (content image) - and outputs a filtered version of the content image, changed to emulate the chosen style. The algorithm is largely based on a 2015 paper called “A Neural Algorithm of Artistic Style” by Gatys, Ecker, and Bethge. In it, a CNN-based system for creating artistically stylised images is introduced. The neural network is employed to discern and recombine the style and the content (i.e. objects and how they are arranged) of digital images.

Convolutional neural networks are composed of layers, each layer processing the given information feed-forward and hierarchically. In our case, every layer of computational units could be viewed as a set of image filters: every filter extracts a feature from the image, which ends up in several differently filtered versions. High layers are responsible for capturing content like objects present on the image and their location, hence they are referred to as the content representation by Gatys et al. Conversely, low layers work on exact pixel values.

Representing an image’s style is similar to extracting texture information from it: Gatys et al. use a feature space designed for this purpose, which is built on the filters applied by each layer of the CNN. It captures the style of the image by defining it as the correlation between these filter responses, i.e. a representation is obtained by the feature correlations of several layers, capturing texture but no content information.

In our project, we have used these methods of obtaining content and style in order to then recombine them to achieve style transfer from one input image to another. This is made possible by the fact that both style and content representations are individually modifiable since they are separable in a CNN - Gatys et al. point this out as their paper’s main finding. The creation of the output image requires finding a good match between the content of the first input image (in our case a photograph) and the style of the second image (in our case a piece of art). In the result, the arrangement of objects in the photograph remains the same, but colours and local detail are determined by the style image; visually, the stylistic appearance of an input artwork is applied to any input photograph.

As can be seen in our case studies, the style transfer is not perfect; this is due to the fact that while separable to a degree, the content and the style of an image cannot be fully isolated from each other. It is not always possible to find an image which would be the perfect balance between the content and style representations obtained from the input images. Nevertheless, in both Gatys’ implementation and in ours, the loss function (which must be minimised) contains separate terms for content and style, which have different effects on the result when changed. This means that by modulating the hyperparameters (style weight and content weight), it is easy to put an emphasis on either the content or the style to achieve the best results. Too strong an emphasis on the style would distort the content of the photograph too much, and on content it would not match the artwork’s “feel” particularly well; there is a balance to be struck.

Algorithm

The algorithm closely follows the implementation provided by Gatis et al. while making it more suited to our purposes. Its steps are described below.

Step 1 *Loading the pre-trained classifier*

We use the vgg19 classifier, which is pre-trained on the ImageNet dataset. The classifier has been modified so that the last (fully connected) layer which outputs classification is removed - we have no need for it.

Step 2 *Choosing layers for content and style*

```
# Relevant layers, so far these seem to perform best
self.style_layers = ['r11', 'r21', 'r31', 'r41', 'r51']
self.content_layers = ['r42']
```

These layers were found by experimentation and chosen because the best results were achieved with them. The r42 layer deals with the location of objects in the image, while the others hold important information about the image's style. After testing, it made the most sense to use these layers for content and style loss.

Step 3 *Selecting style and content weights*

```
# Init weights
style_weights = [style_weight / n**2 for n in [64, 128, 256, 512, 512]]
content_weights = [content_weight]
self.weights = style_weights + content_weights
```

Different weights are applied to different layers: the higher the number of the style layer, the lower its style weight. As previously mentioned, style and content weight are hyperparameters which can be changed to achieve different results. The ones we have chosen seem to yield quite pleasing results more often than not, based on our testing with several values.

Step 4 *Loading content and style images*

```
self.prepare = transforms.Compose([transforms.Resize(self.img_size),
      transforms.ToTensor(),
      transforms.Lambda(
            lambda x: x[torch.LongTensor([2, 1, 0])]),
            transforms.Normalize(mean=[0.40760392, 0.45795686, 0.48501961],
            std=[1, 1, 1]),
            transforms.Lambda(lambda x: x.mul_(255))
      ])
```

The image is preprocessed at this step. It is resized, and then its format is changed to match that of the images in the ImageNet database. This is done so that it will work better with the classifier, which was trained on that database.

Step 5 *Initialising resulting image*

The resulting image is a copy of the content image; we also tried to make it a copy of the style image or random noise, but the content image gave the best result.

Step 6 *Selecting content and style targets for calculation*

```
# Init loss functions
loss_fns_1 = [networks.GramMSELoss().to(self.device)] * \
      len(self.style_layers)
loss_fns_2 = [nn.MSELoss().to(self.device)] * len(self.content_layers)
self.loss_fns = loss_fns_1 + loss_fns_2
```

Each style layer is run through a convolutional neural network, and then the Gram matrix for the result is calculated. The content layers are only run through the CNN.

Step 7 *Initialising optimiser*

The optimiser is initialised using the result image. We use an optimiser called LBFGS (Limited memory BFGS) - it seemed to work best, based on trial and error.

Step 8 *Running optimisation steps*

```
def closure():
      optimizer.zero_grad()

      out = self.cnn(opt_img, self.loss_layers)
      layer_losses = [self.weights[a] * self.loss_fns[a](A, targets[a])
      for a, A in enumerate(out)]

      loss = sum(layer_losses)
      loss.backward()
      iteration_count[0] += 1

      if iteration_count[0] % log_every == (log_every - 1):
            print(f'Iteration: {iteration_count[0] + 1},
            loss: {loss.data}')

      return loss
```

The above code shows one step of the optimiser. Using a CNN, the next iteration of the output image is calculated. For each layer in the output image, the layer's losses are computed. Then, the total loss is summed. Finally, we back-propagate for the next step of the optimiser.

Step 9 Saving output image

```
self.post_process_first = transforms.Compose([transforms.Lambda(lambda x:
x.mul_(1./255)),
transforms.Normalize(mean=[-0.40760392, -0.45795686, -0.48501961],
std=[1, 1, 1]),
transforms.Lambda(
      lambda x: x[torch.LongTensor([2, 1, 0])]),
])

self.post_process_second = transforms.Compose([transforms.ToPILImage()])
```

The output image is returned to its original dimensions and postprocessed, i.e. the reverse steps to the preprocessing are done. Then, the image is saved - this image is the result of the algorithm.

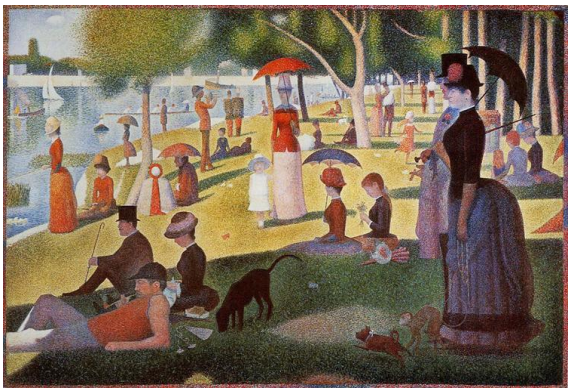
Case studies

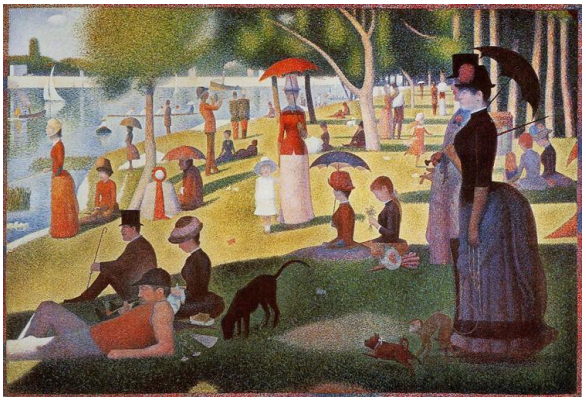
The application was tested extensively with many different inputs. Here we present some successful attempts alongside some typical ways in which the style did not transfer very well.

Successful transfer

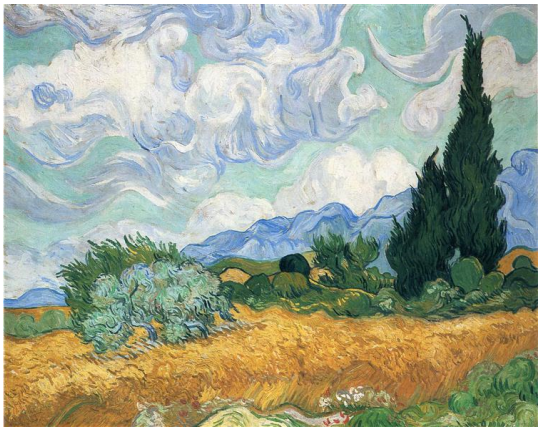
Paintings with mostly evenly distributed, smooth textures seem to produce the best results, of which we provide several examples below.

"Alfalfa, St. Denis" by Georges Seurat





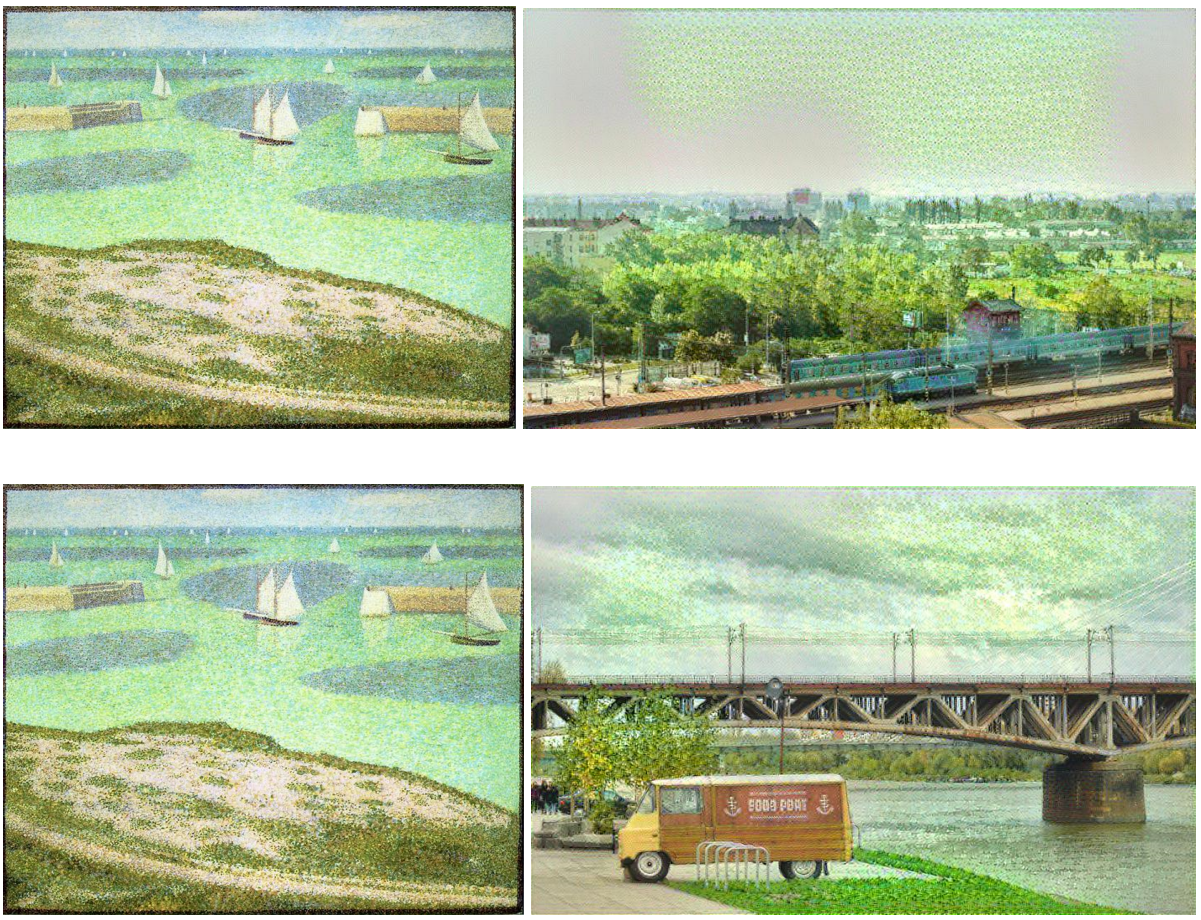

“A Sunday Afternoon on the Island of La Grande Jatte” by Georges Seurat




“Wheat Field With Cypresses” by Vincent van Gogh

Typical issues

The most commonly occurring issues when transferring style seem to be with large “empty” spaces like a piece of clear sky or tiny details like a fence post or a horse leg, as seen in the examples below. These issues occur especially when the style of the artwork being used is very distinctive or “strong,” causing it to overwhelm the content of the resulting image. When the style weight is reduced and the content weight is increased, there may be some improvement, but at the cost of the style becoming much less noticeable in areas where it was working well previously.



“Port-en-Bessin Entrance to the Harbor” by Georges Seurat

In these two images we can clearly see that the style and colour of the painting is being transferred to areas where it shouldn’t be, like the empty patch of sky in the upper image or some of the “flatter” clouds on the lower one. However, for the slightly more detailed parts, it has transferred well. In general, images with several very distinct areas content-wise seem to be more problematic for transfer.

“Melody of the Night” by Leonid Afremov

In this example we see the opposite problem: small details like the horses’ legs and the fence post are blurred and circular patches appear around them. In this case, the painting that was used is also highly distinctive in style.

Conclusion

In general, the easiest paintings to work with appear to be ones with smooth, not overly complicated textures. Transferring their styles is most successful with photographs whose content is similarly evenly distributed. Commonly, some issues crop up when these ideal circumstances are not fulfilled - empty spaces or minute details can often be either overlooked or overcompensated for, resulting in an image that has some problem areas, even if for some parts of it the style transfers well. These issues can sometimes be solved by adjusting the hyperparameters or the number of iterations performed on that particular example, but there does not seem to be a single combination of settings that yields good results regardless of the images used. However, with some individual attention, our project can produce some quite interesting and aesthetically pleasing results.

Sources cited

Gatys, Leon, et al. “A Neural Algorithm of Artistic Style.” *Journal of Vision*, vol. 16, no. 12, 2016, p. 326., doi:10.1167/16.12.326.

Gatys, Leon. “Leongatys/PytorchNeuralStyleTransfer.” *GitHub*, github.com/leongatys/PytorchNeuralStyleTransfer/blob/master/NeuralStyleTransfer.ipynb.