Painting the Starry Night: Implementing Image Style Transfer Using Convolutional Neural Networks

Melis Gokalp
Brown University
Providence, RI, USA
melis_gokalp@brown.edu

Joshua Kim
Brown University
Providence, RI, USA
joshua_kim@brown.edu

Abstract

We took on the task of implementing image style transfer, that is applying the artistic style of an image to another image. We aimed to implement the original Gatys et al (2016) paper titled "Image Style Transfer Using Convolutional Neural Networks"[1]. The main idea was to use the image features derived from the layers of a Convolutional Neural Network (CNN) and recombine them to create stylized image. Our loss-minimizing algorithm used with CNN produces quick and proper results under 20 iterations. Our results provide a proper understanding of how semantic features of an image can be transferred to another image.

1. Introduction

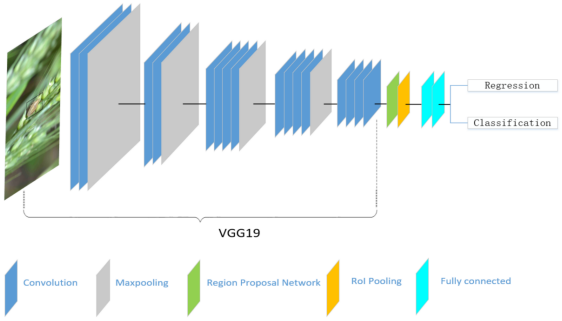
After carefully studying the Gatys et al (2016) paper, we figured out that their approach is to think of style transfer problem not only as a texture transfer problem but rather a semantic feature transfer one. The image style transfer papers written before theirs all focus on the low-level image features, whereas Gatys et al suggest focusing on high-level features. By extracting high-level features, they were able to create semantically meaningful results that reflect the overall style of the image.

1.1. Convolutional Neural Network

To tackle the problem of separating the content from style in images is difficult since looking at per-pixel differences in the image does not yield good results. Instead we can use the Convolutional Neural Networks trained with labeled weights, which are mainly used for categorization problems, and use their high-level feature extraction. Therefore instead of looking at per-pixel differences in images to extract the style, we looked at semantic features of the image.

The standard VGG19 network is a 19-layer network with 16 convolutional layers, and 3 fully connected layers. Its original application is to extract different levels of features

Figure 1. VGG19 Architecture



in each of its layers for image classifiation. It is a network pretrained on the Imagenet database. Gatys et. al found that these feature extractor layers would enable the extraction of content and style features from images. Hence, they proposed the use of a VGG19 network.

```
model = VGG19(input_tensor=combined_tensor,
weights='imagenet', include_top=False)
```

By specifying the $include\_top = False$ parameter, we chose not to use the 3 fully connected layers on top and also used pretrained weights from the $imagenet$ dataset. Our pretrained data with proper labels allowed our VGG19 model to learn from it.

Then we calculated the cross-entropy loss of our input images compared to the training images. Finally, we used a gradient optimizer to optimize the parameters and minimize the loss in each iteration.

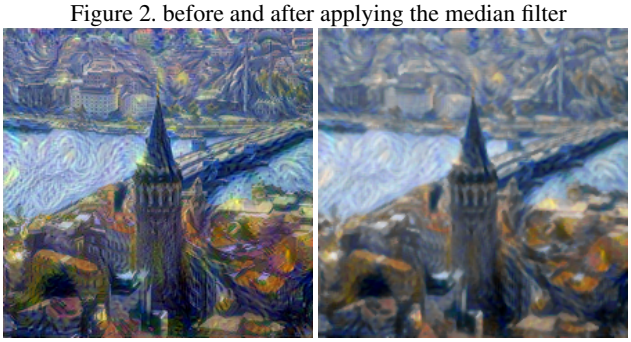
1.2. Design Choices

While we aimed to implement the Gatys et al paper, we made several design choices to generate the best results. We have used a maxpooling layer contrary to the average pooling layer suggested in the paper as we were already happy with our results.

We have tried using both the convolutional layer 4 and 5 to extracy our content features from and we decided to continue with layer 4 as it worked better with our parameters.

This might be a result of deeper layers extracting more general features.
Our initial results generated very pixelated and noisy images since we resize the images before loading them to our model. We overcome this problem by using a median filter of size 3x3, to reduce the noise in the image and make it look more smooth. As it was included in the paper, we

Figure 2. before and after applying the median filter



experimented with feeding a white-noise image to our algorithm, which generated interesting results that demonstrated style extraction. Gatys et al paper suggests changing the relative weights of the style and content images, which we did while experimenting with different images and found the most optimal $\frac{\alpha}{\beta}$ ratio to be $\frac{0.025}{1.0}$.

2. Image Representations

Image representations are gathered from the contents of VGG19 layers. Layers act as perception fields that perform feature and pattern extraction. Deeper layers focus on more general patterns. As it was suggested in the paper, we used convolutional layers `conv1_1`, `conv2_1`, `conv3_1`, `conv4_1`, `conv5_1` for generating the style features, while only the `conv4_2` was used for content extraction.

2.1. Content Images

As we are using the VGG19 trained on object recognition, higher levels of the network capture the high level content but miss out the exact pixel values, thus provide a categorical representation of the image. Whereas the lower layers just reproduce the existing pixels and do not provide a meaningful representation for content. This is why we are using the feature responses in higher layers of VGG to create our content features.

2.2. Style Images

For style images we aim to gather the textural information from our image. We looked at the feature correlations across five convolutional layers and minimized the distances between the gram matrices from the style and contact image. This process is explained in our loss functions.

3. The Algorithm

Our algorithm can be split up into three stages:
1. Extraction of Style and Content features
2. Computation of loss function
3. Evaluation and Optimization

First, we set up the pre-trained VGG19 network using Keras's model package. Using this pre-trained model, we passed in a concatenated tensor of the Content, Style image and an unitialized placeholder for the result image which would then get combined with style and content. With the model built, we then stored a dictionary with the layer name as key and all computed layer outputs as values.

For each layer in the content layers and style layers, we computed the losses of the generated image against both the content image for content features and style image for style features. This was done by computing loss for each of the respective relevant layers. Using a linear combination of style loss and content loss, the total loss was computed.

We next calculated the gradients using Keras's gradients function by passing in the generated image and the loss. Next initialized a Keras function to calculate the loss and gradients while iterating throughout the optimization. Optimization was done using a BFGS from the scipy.optimize package and the gradient and loss values were tracked using a seperate "evaluator" class.

3.1. Loss Functions

The loss functions we used were the ones defined by Gatys et. al (2016).

Content Loss
Content loss was computed by computing the least square difference (Euclidean distance) between the original content image and the mixed output image.

Figure 3. Content loss as in Gatys et. al (2016)

$$\mathcal{L}_{\text{content}}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^{l} - P_{ij}^{l}\right)^2 .$$

Style Loss
Style loss was computed by calculating the least square difference (Euclidean distance) between the gram matrix of the style image and mixed output image. The Gram matrix of these layers acted as a non-normalized correlation matrix of the filters, which computes the change in feature extent of style. For each style layer, the loss was computed as a function of the number of layers (Figure 5.) since the loss

was computed across multiple layers.

Figure 4. Gram function

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l.$$

Figure 5. Style loss for each layer

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - A_{ij}^l\right)^2$$

Figure 6. Total style loss

$$\mathcal{L}_{\text{style}}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l,$$

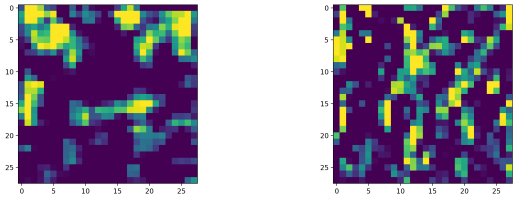
4. Results

After following the implementation in the paper we were able to get step by step results. Visualizing our images was a great way of debugging as it pointed out what might be wrong in our algorithm.

4.1. Process and Problems

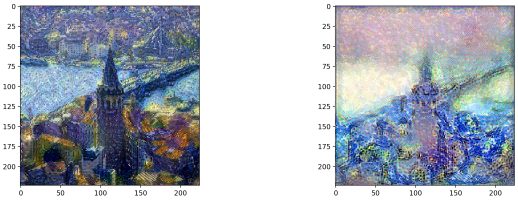
Our initial progress was rather poor as our loss functions were incorrect and we could not produce a good image. The first visualization of the VGG layers was like below:

Figure 7. First attempts at visualizing feature layers



After implementing more functionality and completing the loss minimization algorithm, we were able to get initial outcomes.

Figure 8. Visualizing the Starry Night



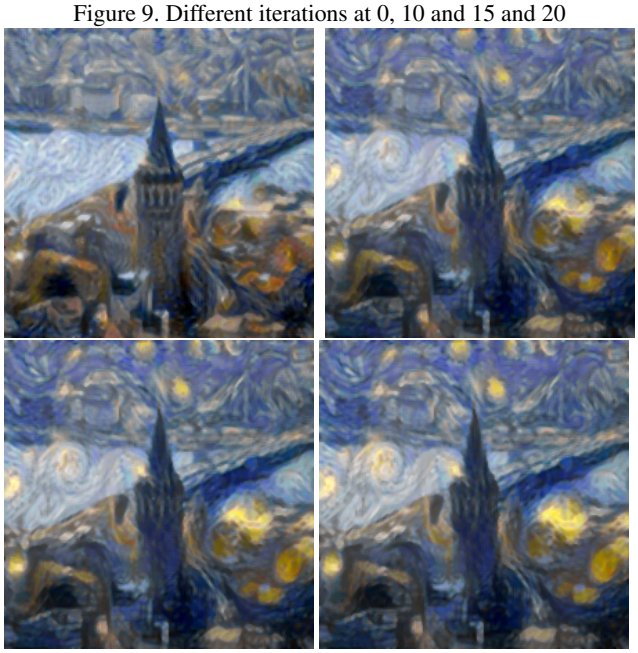
Our first iteration was somehow meaningful and stylized, yet the next iterations just became very bright. We struggled in reducing brightness for a while and reexamined all of our algorithm.

Then we figured out the simple issue of using the deprocessed image in our reiterations. Since our deprocessing function adjusts the RGB channels of our image to be brighter, we got brighter images each time.

```
def deprocess_image(x, img_h, img_w):
    x = x.copy().reshape((img_h, img_w, 3))
    x[:, :, 0] += 103.939
    x[:, :, 1] += 116.779
    x[:, :, 2] += 123.68
    x = x[:, :, ::-1]
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

We fixed this issue by copying and storing the mixed image x each time before we processed and applied the median filter. Then we reiterated the original one. Our iterations looked smooth and consistent after these adjustments.

Figure 9. Different iterations at 0, 10 and 15 and 20



4.2. Optimization

Optimization of our function involved finding the best number of iterations and optimal weights for content and style in our loss function.

After experimenting with small and large numbers we observed that what matters is the ratio between α and β, rather than their magnitude. We set different values for each pair, as some of them had less dominant style features, such as the Picasso pair. We used a larger style weight to balance it out. The maxfun parameter in our BFGS optimizer, which is the maximum number that is our loss function is evaluated for minimizing the loss. We observed that numbers that are too small, such as 5 and 10 generate results at a lower rate whereas 30 and 60 create highly mixed images

that looked too similar to the style image. We decided that the optimal maximum evaluation was 20.

Figure 10. First iterations of maxfun=10 and maxfun = 50

We decided that the optimum values were $style\_weight = 0.025$, $content\_weight = 1.00$, $max\_iteration = 20$ and $maxfun = 20$.

4.3. Resulting images

Figure 11. Our base image, Galata Tower in Istanbul, Turkey

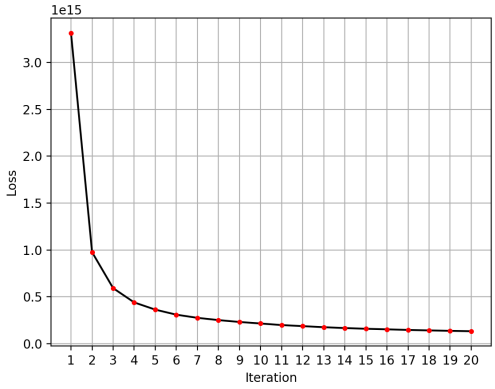
We have generated pleasing results in our trials of image style transfer. We have used a based image and tried different transfers. On average 15-20 iterations were enough to create well-stylized images. We have observed that our loss minimization algorithm reduced the loss exponentially. The graph below shows the loss-reduction for the starry-night image for 20 iterations. Our loss iteration function clearly demonstrated minimization.

We implemented the styles of different artists such as Vincent Van Gogh, Edward Munch, Picasso and Kandinsky. Please see our last page (page 5) for full results and an extra image.

4.4. What we have learned

This project provided us a chance to set up a VGG19 network using Keras with pretrained and extract style and content features from images. This was something neither

Figure 12. Loss versus Iteration Graph



of us had done before.

Using these extracted features we calculated loss functions and minimized the total loss. We studied the equations provided in the paper and wrote functions for each one of them.

We applied our understanding of a categorization problem to a style transfer problem by using the semantic features of an image on a style image.

Our results were pleasing and we were able to run our algorithm on different style and content pairs.

4.5. Notes on running the file

Users can simply run the ourmodel.py file without any parameters.

References

[1] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge *Image Style Transfer Using Convo-lutional Neural Networks*. Gatys et al, 2016.

[2] Denan Xia,1 Peng Chen,1,2,* Bing Wang,3 Jun Zhang,4,* and Chengjun Xie. *Insect Detection and Classification Based on an Improved Convolutional Neural Network* . Addison-Wesley, Reading, Massachusetts, 1993.

Figure 13. Our result images (style image, mixed image)

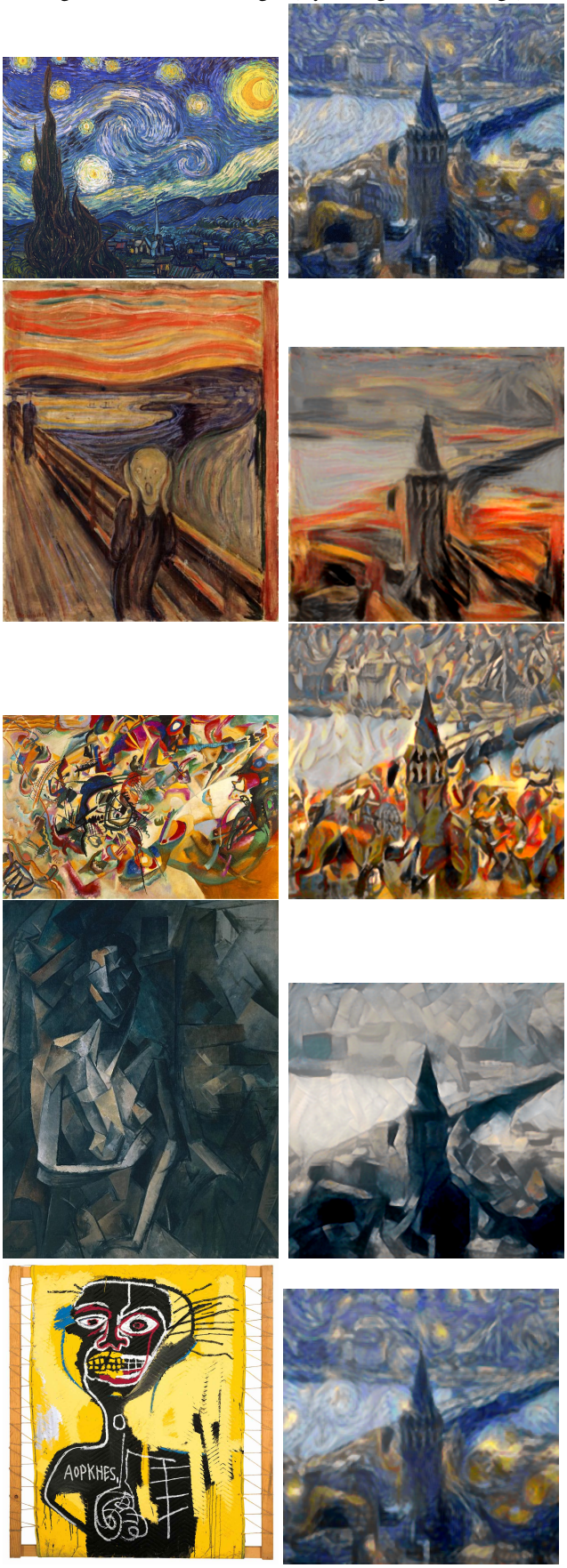


Figure 14. Our favorite professor James Topmkin