Blog Post

References

https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-neural-style-transfer-ef88e46697ee
https://www.youtube.com/watch?v=imX4kSKDY7s&t=2s
https://pytorch.org/tutorials/advanced/neural_style_tutorial.html
https://blog.techcraft.org/vgg-19-convolutional-neural-network/
https://medium.com/geekculture/neural-image-style-transfer-515fe09f1c0c
https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5
https://www.geeksforgeeks.org/python-foreground-extraction-in-an-image-using-grabcut-algorithm/
https://www.fritz.ai/resources/tutorials.html

Also put like an image here

Introduction

From being inspired to change the way we advertise movie posters, our group decided to look into Neural Style Transfer for our Final Project. Throughout this project, we tested Style Transfer with multiple different models, parameters, and art images in order to find out what would make the output movie posters as appealing as possible. Additionally, we tested Style Transfer with foreground and background detection and used multiple style images in relation to one content image to get a more thorough understanding of the topic.

Background

To put this into perspective, Style Transfer uses the content of one image, and the style of another image in order to create an entirely new output image. In our case, the movie posters were the content images and various paintings that we chose were the style images. For the image that we are trying to generate, common practice is to just make a copy of your content image and then alter that through the Style Transfer process (known as input image). To get this desired output image, loss functions were needed in order to create a successful Style Transfer.

The total loss function is shown below.

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

In terms of the loss function for the Content Image, we need to compare the content of both the content image and the input image. To do this, we need to choose a set of convolutional layers where we want to place our content loss functions. As we tend to get into the higher convolutional layers as we iterate through a particular CNN (Convolutional Neural Network), this is where the network tends to capture higher-level features such as objects, people, faces, and other similar aspects of an image. Consequently, by placing our content loss function after these middle to higher-level convolutional layers, this will allow for a more accurate representation of the content in the new generated image.

The content loss function is shown below.

$$L_{content} = \frac{1}{2} \sum_{i,j} (A_{ij}^{l}(g) - A_{ij}^{l}(c))^2$$

The content loss

In a nutshell, this loss function is simply just comparing the feature maps of the content image with the input image after a particular convolutional layer in the network, and the output is the mean square error of these feature maps.

Additionally, in regards to the loss function for the Style Image, this process is slightly more complicated. Similarly, just like how we did it for the content loss function, we place the style loss function directly after a set of convolutional layers in the network. However, to compare the styles of the style image and the input image, we need to compute gram matrices for each of the images. To calculate these gram matrices, all you need to do is compute the dot product between the feature maps and their respective transpose. These gram matrices are responsible for computing the correlation between the features within the images and hence this correlation happens to represent the style of the images.

The style loss function is shown below.

$$G_{ij}^{l} = \sum_{k} g_{ik}^{l} g_{jk}^{l}$$

$$S_{ij}^{l} = \sum_{k} s_{ik}^{l} s_{jk}^{l}$$

Gram Matrix for Style and Generated Image

$$\mathcal{L}_{\text{style}}(g, s) = \sum_{i,j} \left(G_{ij}^{l} - S_{ij}^{l} \right)^2$$

In similar regards to the content loss function, the style loss function is comparing the correlations of the feature maps (gram matrices) between the style image and the input image, and the output of this function is the mean square error of these gram matrices.

As a result, for this project, these loss functions were used and minimized using an optimizer (L-BFGS) in order to try and get an appealing output image through this process of Style Transfer. In addition, within the total loss function, specific weights are multiplied by each of the individual loss functions. For example, alpha is multiplied by the content loss function and beta is multiplied by the style loss function and the values assigned to these weights can drastically affect the outcome of the newly generated image.

Data

At the beginning of our project, one of the challenges that we faced was choosing a substantial dataset that would get us the results that we wanted. We ended up finding a dataset of around a thousand different movie posters for our content images. In terms of our style images, we wanted to have a theme for our images so we went ahead and chose Pop Art, Pointillism, and Art Nouveau as our three different art themes. For each of these art themes, we chose five images that represented each theme. At first, we thought that we would go through all of these images, however, we only used a handful of content images and style images to demonstrate the different processes that we performed. This is because we really wanted to see the changes in the outputs of the images whenever we tried new techniques so as a result, we would keep the images consistent in order to see these differences.

Maybe provide a picture for each of these different eras and some movie posters too to add some more visuals to this blog

Data Preprocessing

In order to process our images, we needed to resize our content and style images so that they would have the same dimensions. This was done so that the loss functions would be computed in

a correct manner. Additionally, in regards to the input image that we use to create our output image in the end, we just made a copy of our content image and then just minimized our losses with that particular image. We followed this PyTorch tutorial in order to understand the general process of resizing the images and converting them into tensors. The pixel size of 200x200 is just an arbitrary number that we chose so that we would not have to wait an extensive amount of time to run the Style Transfer.

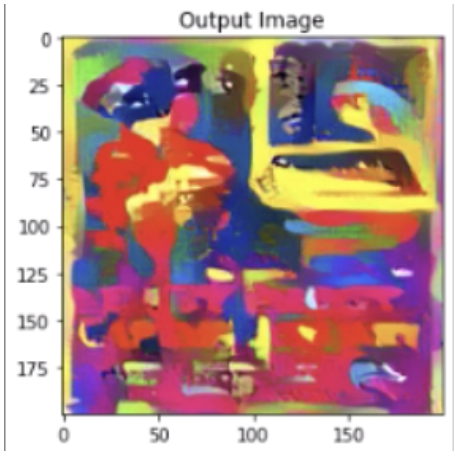
```
imsize = 512 if torch.cuda.is_available() else 200  # use small size if no gpu

loader = transforms.Compose([
    transforms.Resize(imsize),  # scale imported image
    transforms.ToTensor()])  # transform it into a torch tensor

def image_loader(image_name):
    image = Image.open(image_name)
    resized = image.resize((200, 200))

    # fake batch dimension required to fit network's input dimensions
    resized = loader(resized).unsqueeze(0)
    return resized.to(device, torch.float)
```

Our Implementation

At the very beginning of this process, we followed this PyTorch tutorial just to get an idea of how to code and test Style Transfer. The very first model that we used and the one that ended up working the best in the end was the VGG19 model which consists of 16 convolutional layers, 3 Fully connected layers, 5 MaxPool layers and 1 SoftMax layer. We will discuss this model in greater detail below, especially as this is the model that we tried a multitude of different techniques on. Throughout this implementation process, we went from tinkering with various parameters in the Style Transfer algorithm to building upon the scope of the process and introducing Foreground and Background detection. While not without its errors, an abundance of information was learned in the process and we will share our findings in the upcoming sections.

Our first run through:


Style Image
Content Image
BABY PEGGY
"CAPTAIN JANUARY"
HOBART BOSWORTH, IRENE RICH,
LINCOLN STEDMAN, HARRY T. MOREY
and BARBARA TENNANT


Output Image

Tinkering with Parameters

All of these test cases below are using the VGG19 model.

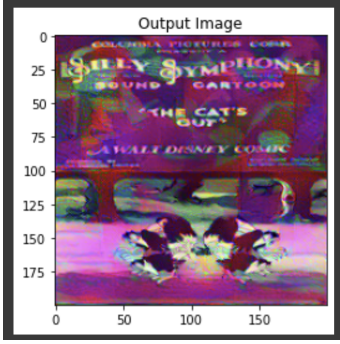
Style Weight (Beta) vs Content Weight (Alpha)

These are the Style and Content images that we will use for the next couple of sections.

As said before, the weights attached to both the style loss function and the content loss function can drastically affect the result of the generated image. Shown below in two different test cases, it is evident that the Style transfer tends to work when the style weight is much larger than the content weight. When the content weight is set to 1000 and the style weight is set to 1, the output image is identical to the original movie poster that we started with.

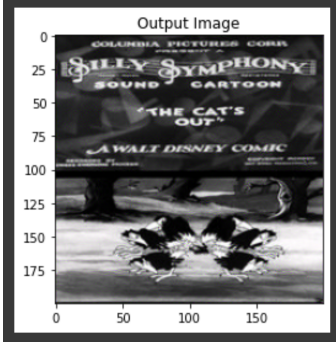
Style Weight - 1000 and Content Weight - 1

```
Optimizing..
run [50]:
Style Loss : 41.629936 Content Loss: 33.641907

run [100]:
Style Loss : 27.302164 Content Loss: 35.975368

run [150]:
Style Loss : 22.557524 Content Loss: 32.864368

run [200]:
Style Loss : 21.794353 Content Loss: 31.081718

run [250]:
Style Loss : 20.549959 Content Loss: 32.338600
```



Style Weight - 1 and Content Weight - 1000

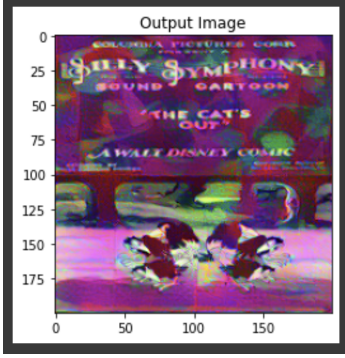
```
Optimizing..
run [50]:
Style Loss : 0.287057 Content Loss: 729.304016

run [100]:
Style Loss : 0.286821 Content Loss: 166.560471

run [150]:
Style Loss : 0.286046 Content Loss: 78.041763

run [200]:
Style Loss : 0.285444 Content Loss: 40.763897

run [250]:
Style Loss : 0.285118 Content Loss: 26.792656
```



Placement of Losses in relation to Convolutional Layers

The specific placement of style and content losses can make the difference in creating an appealing output image. In the case below, we placed the style losses after the first five convolutional layers that we saw and only one content loss after the fourth convolutional layer that we saw. In terms of the output image, the content of the generated image looks similar in regards to the font and the text of the original movie poster. Additionally, some of the colors from the style image can be seen in this output poster, yet, the styles are not too overpowering in the overall scheme of things. This is due to the fact that we are using the earlier layers of the VGG19 model. As the VGG19 model has 16 convolutional layers in total, we are only looking into the first five convolutional layers which only represent the simpler styles of the style image.

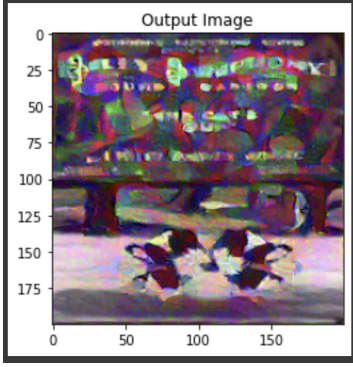
```
Optimizing..
run [50]:
Style Loss : 44.458622 Content Loss: 35.308987

run [100]:
Style Loss : 29.122795 Content Loss: 35.834297

run [150]:
Style Loss : 22.203642 Content Loss: 34.122150

run [200]:
Style Loss : 21.483519 Content Loss: 31.847603

run [250]:
Style Loss : 20.621500 Content Loss: 31.236357
```



In comparison to the previous output image up above, for the next case down below, we placed the style losses after the 1st, 3rd, 5th, 9th, and 13th convolutional layer and placed one content loss after the 9th convolutional layer in the network. With these placements, the text and the objects have drastically different textures than before. This is due to the fact that we are also incorporating styles and content from the higher-level layers which focus more on objects. There is not as much color as the previous output image had, however, this is because we are not

focusing as much on the lower-level layers. Overall this output image feels a lot more complex and it provides a whole new mood and connotation to the movie poster.

```
Optimizing..
run [50]:
Style Loss : 64.947113 Content Loss: 95.887939

run [100]:
Style Loss : 50.397869 Content Loss: 91.507729

run [150]:
Style Loss : 46.418030 Content Loss: 89.910347

run [200]:
Style Loss : 44.383923 Content Loss: 89.128502

run [250]:
Style Loss : 43.181736 Content Loss: 88.633453
```



Testing Different Models

The same style and content images will be used to keep things consistent.

VGG19

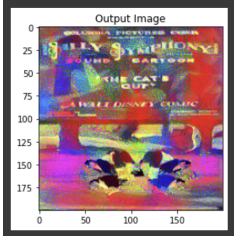
As mentioned before, the VGG19 model consists of 16 convolutional layers, 3 Fully connected layers, 5 MaxPool layers and 1 SoftMax layer. In terms of the project, the style loss and the content loss were mostly dependent on the outputs of the 16 convolutional layers.

These were the parameters that were used:

Content weight: 1
Style weight: 100
Style Losses: [1, 2, 3, 4, 5]
Content Losses: [4]

The output shown below is expected as we are only looking at the lower-level convolutional layers, so the objects within the image should not be as affected by the styles. However, overall the colors from the style image mostly translated over creating a more interesting movie poster.

```
Optimizing..
run [50]:
Style Loss : 15.077452 Content Loss: 45.974949

run [100]:
Style Loss : 15.204498 Content Loss: 43.498608

run [150]:
Style Loss : 26.833170 Content Loss: 45.264893

run [200]:
Style Loss : 13.302997 Content Loss: 43.112194

run [250]:
Style Loss : 13.476197 Content Loss: 45.385742
```



Smaller Custom-made Model

The next model that we used was this smaller model that we created, which we have shared down below.

```
[ ]  # This way is also equivalent
     class SmallConv(nn.Module):
         def __init__(self):
             super().__init__()
             self.relu = nn.ReLU()
             self.conv1 = nn.Conv2d(3, 6, (3, 3), stride=(1, 1), padding=(1, 1))
             self.pool = nn.MaxPool2d(2, 2)
             self.conv2 = nn.Conv2d(6, 16, (3, 3), stride=(1, 1), padding=(1, 1))
             self.conv3 = nn.Conv2d(16, 32, (3, 3), stride=(1, 1), padding=(1, 1))


         def forward(self, x):
             x = self.pool(self.relu(self.conv1(x)))
             x = self.pool(self.relu(self.conv2(x)))
             x = self.pool(self.relu(self.conv3(x)))
             # x = torch.flatten(x, 1) # flatten all dimensions except batch

             return x
```

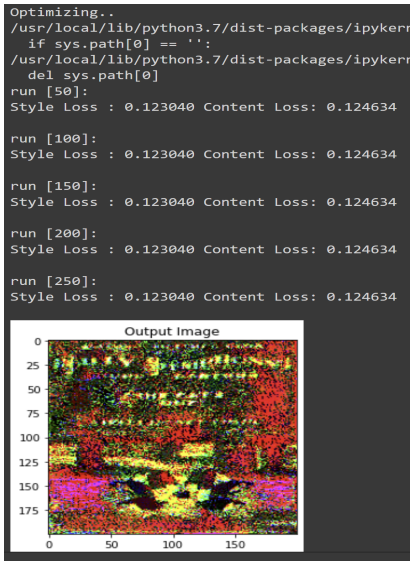
Within this model, we are only using three different convolutional layers and 3 MaxPool layers just to compare how a smaller model differs from a much deeper network like VGG19.

These were the parameters that were used:

Content weight: 1
Style weight: 100
Style Losses: [1, 2, 3]

Content Losses: [2]

In terms of the output image, the network did pick up a lot of the different colors and textures from the style image, however the integration of the style into the content did not perform as well as the VGG19 model did. Because this network did not have many convolutional layers to begin with, it did not do a good job at styling the independent objects (higher-level features) within the image.



ALEX NET

Finally, the last model that we tried in relation to Style Transfer is known as AlexNet. Within the AlexNet framework, we have 5 convolutional layers, 3 MaxPool layers, 2 normalization layers, 2 Fully connected layers, and 1 SoftMax layer. In regards to our objective, we only focused on the 5 convolutional layers and the 3 MaxPool layers.

These were the parameters that were used:

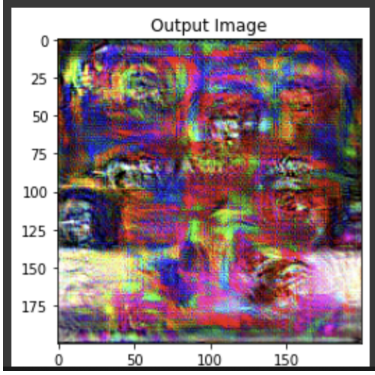
Content weight: 1
Style weight: 100
Style Losses: [1, 2, 3, 4, 5]
Content Losses: [4]

The parameters that we used were the same as the VGG19 model as we were trying to create an accurate comparison between the two different models. Within the output image, the result appeared to be very different from what we were expecting. It looks nothing like a combination

of the style image and the content image, but rather this appears to be a whole new painting. We realized that the reason that this is happening is most likely due to the fact that the first few layers of the AlexNet model have a very large kernel size. These larger kernel sizes have a tendency to capture more features at a time, which means that it does not slowly learn and build upon lower-level features as the VGG19 model. As a result of this, the AlexNet model did not capture a detailed perspective of the style image and instead just did a general overview of the styles within the style image, which therefore created a convoluted mess of colors within the movie poster.

-

```
Optimizing..
/usr/local/lib/python3.7/dist-packages/ipykernel
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel
  del sys.path[0]
run [50]:
Style Loss : 87.251282 Content Loss: 94.798012

run [100]:
Style Loss : 56.386326 Content Loss: 91.204399

run [150]:
Style Loss : 44.779701 Content Loss: 88.208626

run [200]:
Style Loss : 38.092583 Content Loss: 86.475838

run [250]:
Style Loss : 32.634441 Content Loss: 87.142265
```



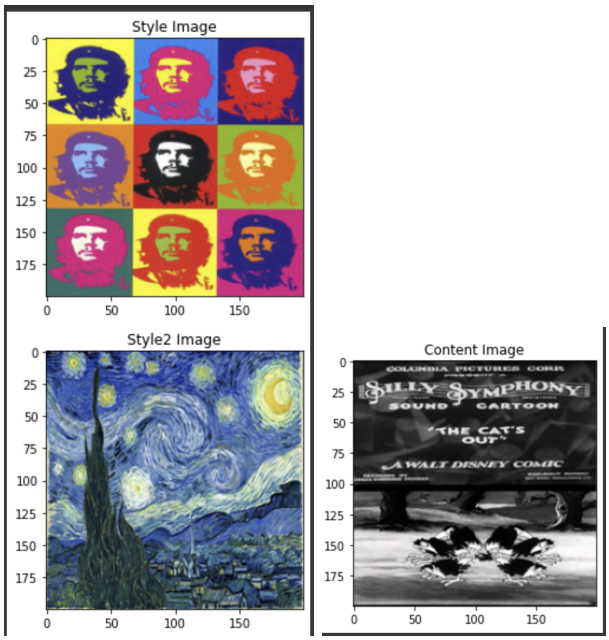
Model Results

After trying all of these different models, it is clear that the VGG19 model produces the best looking output images. Because of the fact that the model has a much smaller kernel size for all of its convolutional layers (3x3), it is able to be more detailed in detecting all of the styles as it iterates through the network. In addition to this smaller kernel size, 16 convolutional layers is helpful in terms of the fact that it lets the network slowly learn, recognize, and build upon the various style features within the images.

Multiple Style Images with just One Content Image

The next step in this process was to see if we could combine the styles from multiple images in order to create an interesting movie poster. To do this, we just added in another Style Loss function for the second style image that we ended up using so that we could compare the styles from each image.

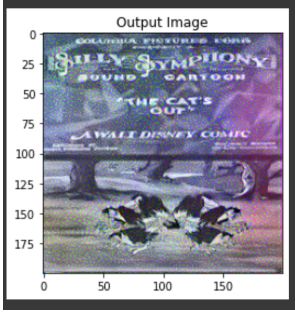
Model Used: VGG19
Content weight: 1
Style weight: 100
Style2 weight: 100
Style/Style2 Losses: [1, 2, 3, 4, 5]
Content Losses: [4]



In terms of the output image, it ended up creating a good mix of the styles from both images. As you can see, the Pointillism style that comes from *The Starry Night* painting is evident throughout the image and you can also see hues of pink and purple that is coming from the other style image. It seems like the output image took more from *The Starry Night* painting than the

other style image, most likely because the input image (copy of the content image) was already similar in tone/colors from the beginning.

```
Optimizing..
run [50]:
Style Loss : 19.262886 Style2 Loss : 3.630908 Content Loss: 7.784529

run [100]:
Style Loss : 18.880173 Style2 Loss : 3.720140 Content Loss: 7.871753

run [150]:
Style Loss : 18.882263 Style2 Loss : 3.654611 Content Loss: 8.059183

run [200]:
Style Loss : 18.662394 Style2 Loss : 3.705199 Content Loss: 8.656576

run [250]:
Style Loss : 29.410664 Style2 Loss : 1.920884 Content Loss: 36.948425
```



Testing for text retention with style transfer

Reverting style transfer

To investigate the permanence of style transfer and the lasting effects of a style on an image, we attempted to recover the original input image after a transfer.

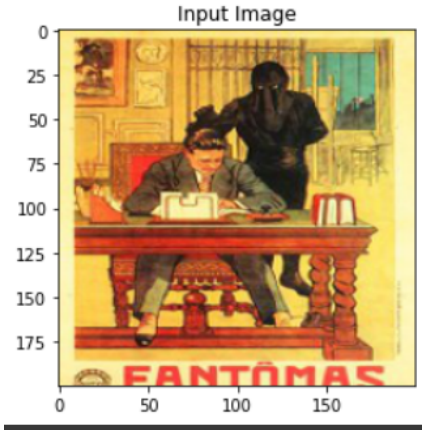
The approach to do so was as follows:

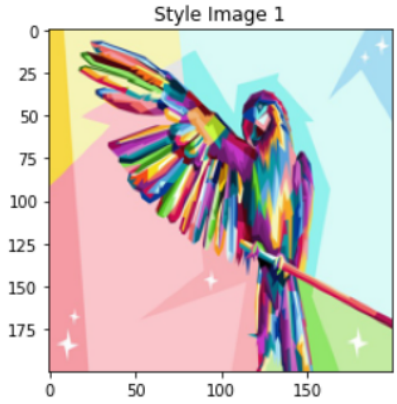
1. Generate style transferred image using original image and a style
2. Utilize output in new style transfer where the content image is the result of step 1 and the style is the original image
3. Repeat step 2 adjusting the content image to the result of each loop
4. Observe results of each transfer

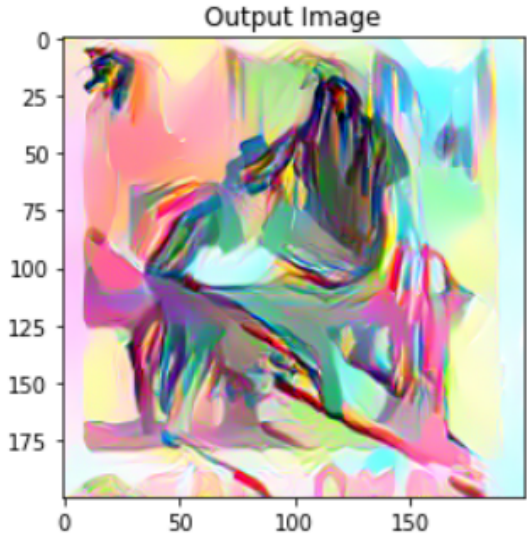
Interestingly, the first pass and second pass generate almost identical images. The second pass is only marginally better in color matching the original piece in comparison to the first pass. One pass seems sufficient to make an image recognizable and two passes make it truer to the original.

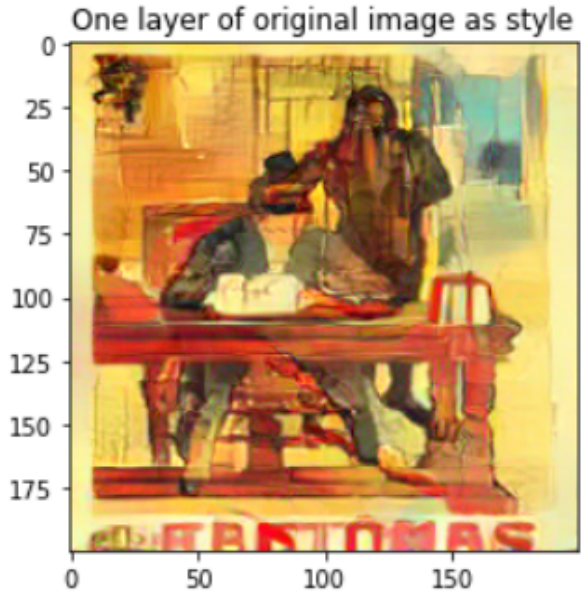
However, when reaching the 3rd pass, both style and content loss inflate greatly and completely lose the original image save for some approximate matches for color locations.

All steps utilize num_steps=500,style_weight=1000000, content_weight=1
To ensure consistency across tests

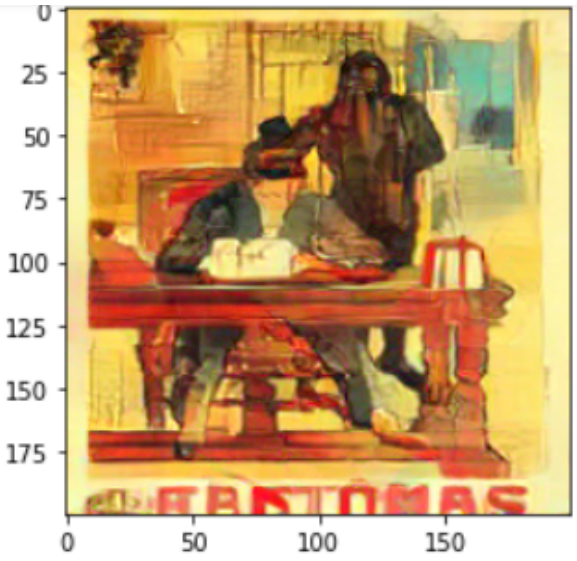




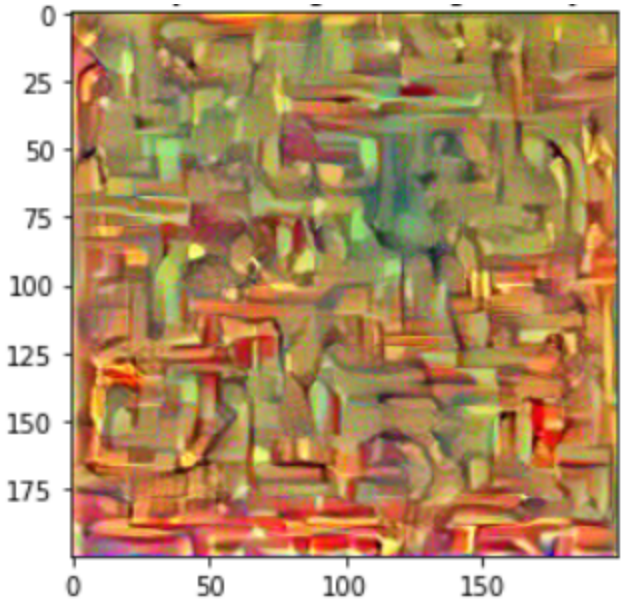





```
run [50]:
Style Loss : 271.824432 Content Loss: 46.814373

run [100]:
Style Loss : 53.327286 Content Loss: 45.031101

run [150]:
Style Loss : 23.129845 Content Loss: 38.722412

run [200]:
Style Loss : 12.847969 Content Loss: 34.363113

run [250]:
Style Loss : 8.302675 Content Loss: 31.722397

run [300]:
Style Loss : 6.086059 Content Loss: 29.758087

run [350]:
Style Loss : 4.915659 Content Loss: 28.383556

run [400]:
Style Loss : 4.289937 Content Loss: 27.395510

run [450]:
Style Loss : 3.800530 Content Loss: 26.697157

run [500]:
Style Loss : 3.508215 Content Loss: 26.197578
```

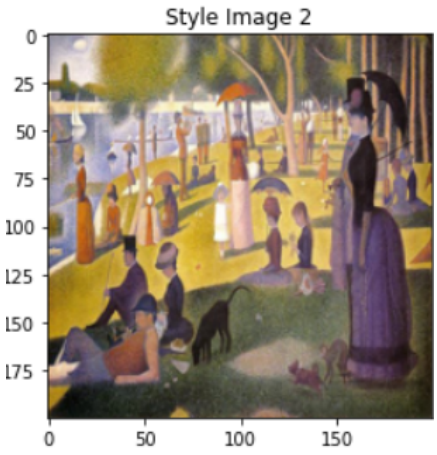


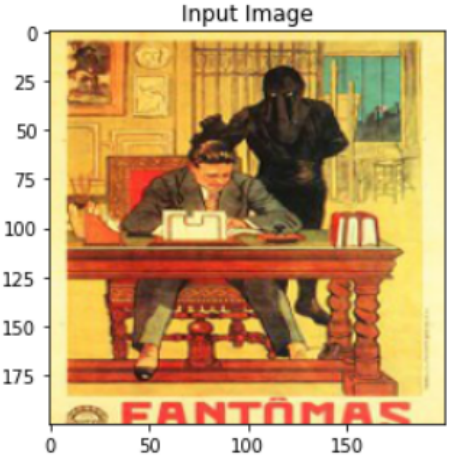

```
run [50]:
Style Loss : 2.120204 Content Loss: 0.264409

run [100]:
Style Loss : 1.701470 Content Loss: 0.392794

run [150]:
Style Loss : 1.506435 Content Loss: 0.447488

run [200]:
Style Loss : 1.386666 Content Loss: 0.476982

run [250]:
Style Loss : 1.297591 Content Loss: 0.504136

run [300]:
Style Loss : 1.234910 Content Loss: 0.517644

run [350]:
Style Loss : 1.187677 Content Loss: 0.529697

run [400]:
Style Loss : 3.199298 Content Loss: 0.627785

run [450]:
Style Loss : 1.112844 Content Loss: 0.606218

run [500]:
Style Loss : 1.083263 Content Loss: 0.564160
```

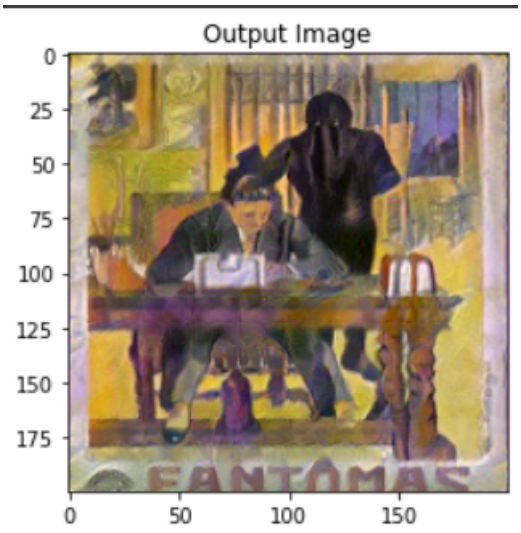


```
run [50]:
Style Loss : 0.898869 Content Loss: 0.058717

run [100]:
Style Loss : 0.811548 Content Loss: 0.095507

run [150]:
Style Loss : 0.776457 Content Loss: 0.110536

run [200]:
Style Loss : 0.753254 Content Loss: 0.122517

run [250]:
Style Loss : 0.737549 Content Loss: 0.137384

run [300]:
Style Loss : 0.718355 Content Loss: 0.155399

run [350]:
Style Loss : 33244248.000000 Content Loss: 545.355713

run [400]:
Style Loss : 32040688.000000 Content Loss: 529.906555

run [450]:
Style Loss : 25555634.000000 Content Loss: 486.749054

run [500]:
Style Loss : 761.554138 Content Loss: 73.320427
```

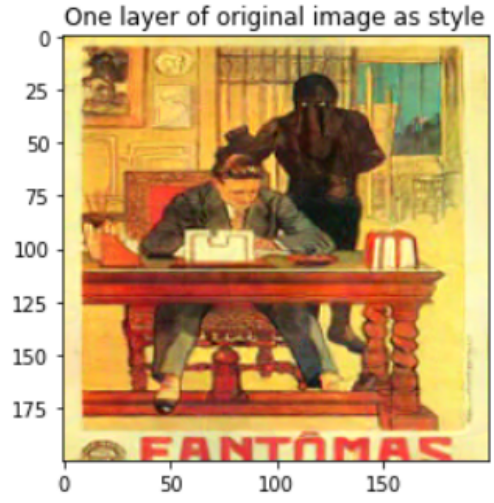
Additional example using pointillism



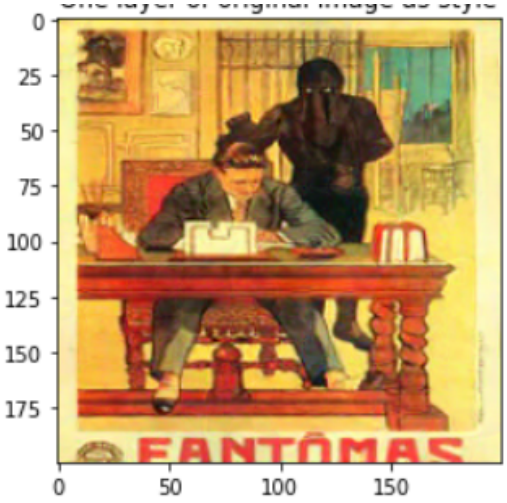




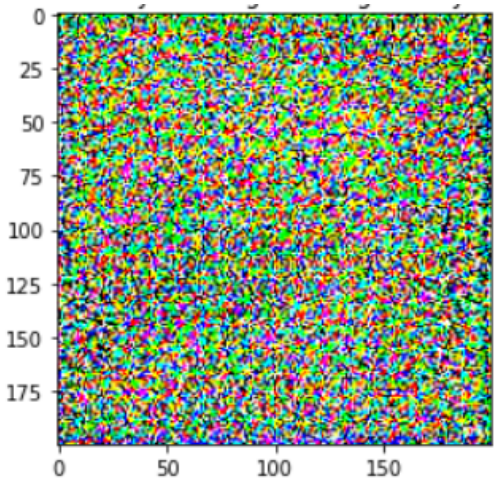

```
run [50]:
Style Loss : 74.728882 Content Loss: 30.920319

run [100]:
Style Loss : 18.532871 Content Loss: 26.306368

run [150]:
Style Loss : 11.308900 Content Loss: 22.335926

run [200]:
Style Loss : 8.404695 Content Loss: 20.254833

run [250]:
Style Loss : 6.682752 Content Loss: 19.019400

run [300]:
Style Loss : 5.562037 Content Loss: 18.239477

run [350]:
Style Loss : 4.836403 Content Loss: 17.745735

run [400]:
Style Loss : 4.371721 Content Loss: 17.368544

run [450]:
Style Loss : 4.044548 Content Loss: 17.091488

run [500]:
Style Loss : 3.804294 Content Loss: 16.880800
```




```
run [50]:
Style Loss : 157.529999 Content Loss: 24.358706

run [100]:
Style Loss : 36.296028 Content Loss: 23.561104

run [150]:
Style Loss : 12.945130 Content Loss: 20.609873

run [200]:
Style Loss : 5.601686 Content Loss: 18.162170

run [250]:
Style Loss : 2.885977 Content Loss: 16.448065

run [300]:
Style Loss : 1.741984 Content Loss: 15.229695

run [350]:
Style Loss : 1.231794 Content Loss: 14.453900

run [400]:
Style Loss : 0.950660 Content Loss: 13.938428

run [450]:
Style Loss : 0.795720 Content Loss: 13.614524

run [500]:
Style Loss : 0.708945 Content Loss: 13.379278
```




```
run [50]:
Style Loss : 0.388786 Content Loss: 0.055576

run [100]:
Style Loss : 0.283925 Content Loss: 0.087093

run [150]:
Style Loss : 0.245369 Content Loss: 0.096060

run [200]:
Style Loss : 0.222995 Content Loss: 0.099662

run [250]:
Style Loss : 0.208317 Content Loss: 0.101429

run [300]:
Style Loss : 0.197823 Content Loss: 0.102281

run [350]:
Style Loss : 0.190533 Content Loss: 0.102615

run [400]:
Style Loss : 0.183821 Content Loss: 0.105127

run [450]:
Style Loss : 0.181436 Content Loss: 0.101732

run [500]:
Style Loss : 0.177500 Content Loss: 0.102172
```

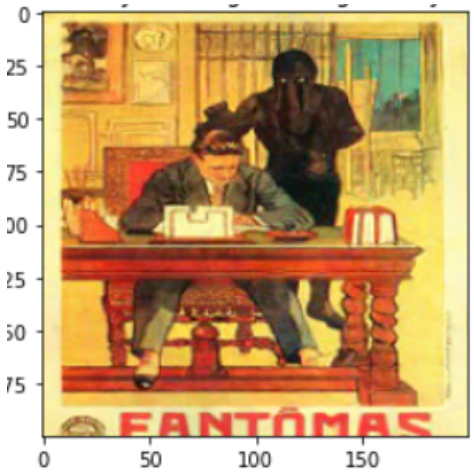



```
run [50]:
Style Loss : 0.143337 Content Loss: 0.011390

run [100]:
Style Loss : 0.125776 Content Loss: 0.018387

run [150]:
Style Loss : 0.118911 Content Loss: 0.021296

run [200]:
Style Loss : 0.114767 Content Loss: 0.022799

run [250]:
Style Loss : 0.111776 Content Loss: 0.024096

run [300]:
Style Loss : 0.109143 Content Loss: 0.025231

run [350]:
Style Loss : 0.107138 Content Loss: 0.025766

run [400]:
Style Loss : 2365.653076 Content Loss: 4.104800

run [450]:
Style Loss : 34899996.000000 Content Loss: 556.598450

run [500]:
Style Loss : 26535660.000000 Content Loss: 503.460846
```

As seen between both runs, when the 3rd pass hits around 300 steps, the style and content loss inflate and distort the original image.

As such the next step is to push to but not exceed the 300 step limit on the 3rd pass.
For brevity's sake, only the third pass result is shown.

```
run [50]:
Style Loss : 0.143337 Content Loss: 0.011390

run [100]:
Style Loss : 0.125776 Content Loss: 0.018387

run [150]:
Style Loss : 0.118911 Content Loss: 0.021296

run [200]:
Style Loss : 0.114767 Content Loss: 0.022799

run [250]:
Style Loss : 0.111776 Content Loss: 0.024096

run [300]:
Style Loss : 0.109143 Content Loss: 0.025231
```

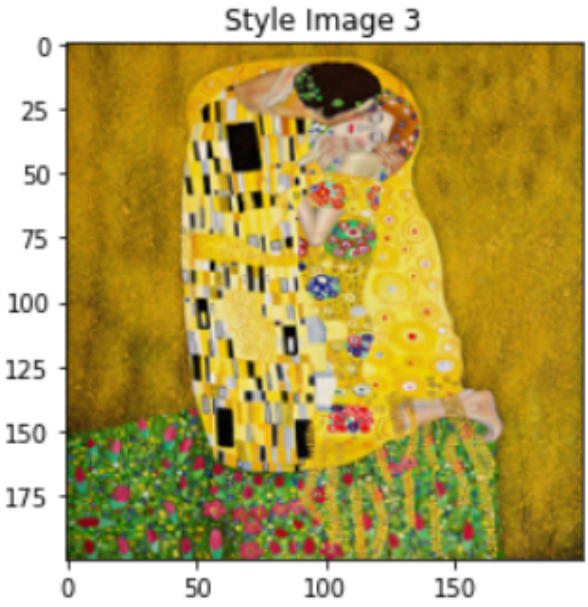


As shown, when stopping the 3rd run at 300 steps, we get a very close approximation of the original with minimal signs of the style image applied to it. However some things such as letters and lines remain misshaped albeit slightly. Thus, to a degree, style transfer is adjustable in removal or decreased presence of applied style.

Conclusive approach for style transfer

Poster Stylization with Text Retention

As seen in previous examples, with styles transferred over posters, text can sometimes get caught up in the transfer and overwritten. Our main goal was to make it possible to generate multiple unique and interesting posters utilizing one source image while applying differing styles. This creates interesting variations while still providing a recognizable format that the user can identify. However, since this method is seen to have issues with the base model we decided to try a variety of approaches to retain text while stylizing the poster art.



We approached this issue with 3 main methods:

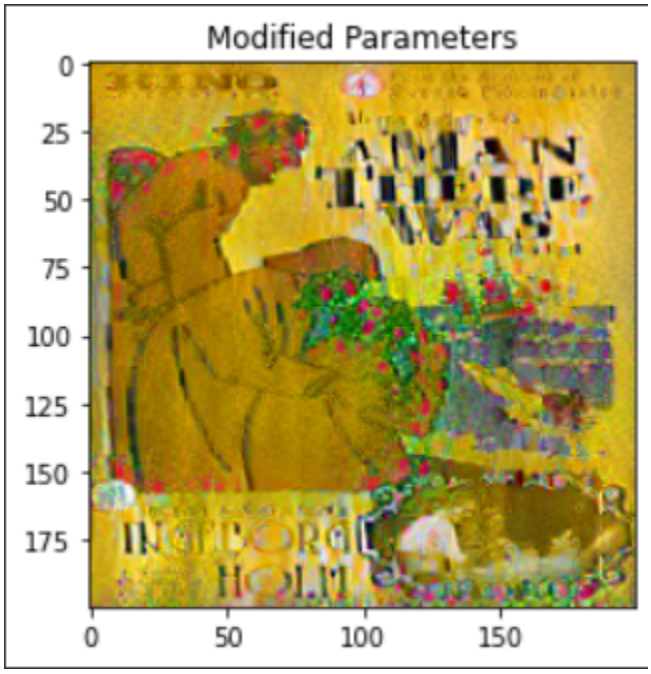
1. Modified content image weights, style image weights, and run steps.
2. Utilizing masks to style transfer portions of images and merge separated pieces back together such that the background (art) is styled and the foreground (text) is left alone.
3. Investigated the approach of segmented style transfer.

Approach 1: Model Modifications

This proved our most successful method in producing recognizable text. However, this also required a stronger content weight which produced a less styled image.

```
run [50]:
Style Loss : 4081.992676 Content Loss: 581.852478

run [100]:
Style Loss : 915.215698 Content Loss: 611.821167

run [150]:
Style Loss : 390.391602 Content Loss: 564.453613

run [200]:
Style Loss : 207.198975 Content Loss: 508.384521

run [250]:
Style Loss : 142.855057 Content Loss: 464.759674

run [300]:
Style Loss : 106.161133 Content Loss: 435.564514

run [350]:
Style Loss : 89.466904 Content Loss: 413.739349

run [400]:
Style Loss : 78.912277 Content Loss: 394.949921

run [450]:
Style Loss : 73.214417 Content Loss: 382.549103

run [500]:
Style Loss : 68.451912 Content Loss: 372.938202
```

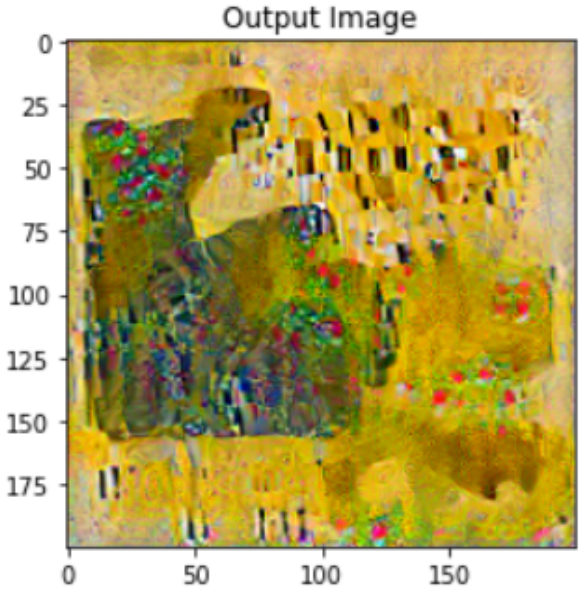
```
run [550]:
Style Loss : 64.667839 Content Loss: 365.795868

run [600]:
Style Loss : 64.606110 Content Loss: 359.358337

run [650]:
Style Loss : 59.274590 Content Loss: 354.517395

run [700]:
Style Loss : 59.157455 Content Loss: 349.709717

run [750]:
Style Loss : 56.884670 Content Loss: 346.022675

run [800]:
Style Loss : 54.386383 Content Loss: 343.006317

run [850]:
Style Loss : 53.158646 Content Loss: 340.079224

run [900]:
Style Loss : 52.214344 Content Loss: 337.519897

run [950]:
Style Loss : 51.192875 Content Loss: 335.189575

run [1000]:
Style Loss : 50.262806 Content Loss: 333.671692
```



Approach 2: Masking

This method was relatively unsuccessful. The most recommended tool for masking, cv2.grabCut, proved difficult to manage. The functions required differing formats for images than our style transfer functions. This produced an issue with multiple patch fixes of conversions and various workarounds to debug incompatibilities. Additionally, after the background image was style

transferred the merging of the masked text with the original image proved difficult as masking did not produce transparencies leading to image content loss. Lastly, grabCut fell short in that when it was able to get a region of an image input, its text detection is not quite what we need for isolating the text within these posters and it either grabs portions of text or produces a completely unreadable foreground mask.



As seen, text is unreadable and the foreground, text, mask result is largely unusable. Thus this method is not usable for our goal.

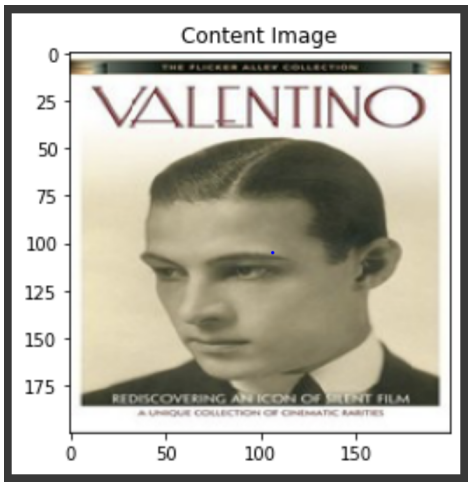
Approach 3: Segmented style transfer

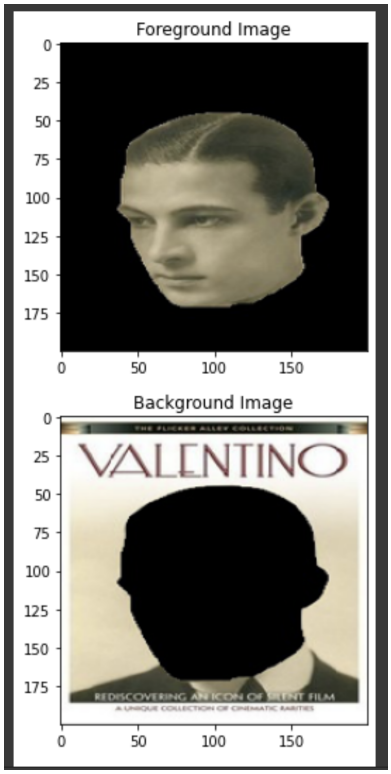
This method took a similar approach as method 2. Thus produced similar issues for the portions we were able to work with. Ultimately, this was unable to produce a usable result as many of the

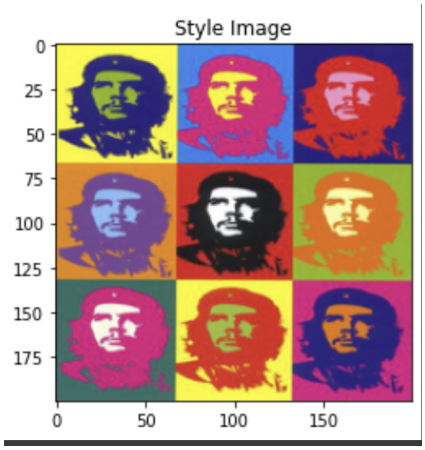
sources regarding it utilized grabCut, running into the same issues as approach 2, or utilized private libraries or difficult to alter functions that did not work with our output/input formats. Thus this method was foregone as we were unable to produce results.
Ideally, This would enable us to utilize an identifier to denote sections we wish to style by marking object types (person, text, scenery, etc.). This would enable us to isolate pieces of the background to stylize each piece, either with one or multiple styles, individually while avoiding text. Sadly, this was not possible due to the constraints introduced by the models and functions we had access to thus no example output exists.

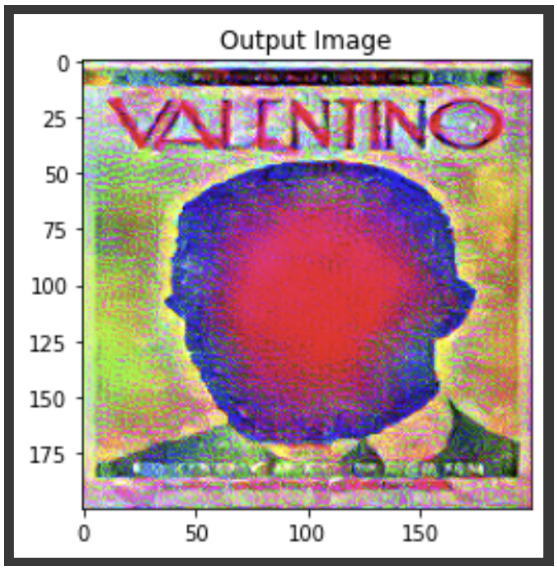
Additional Foreground/Background Style Transfer Images

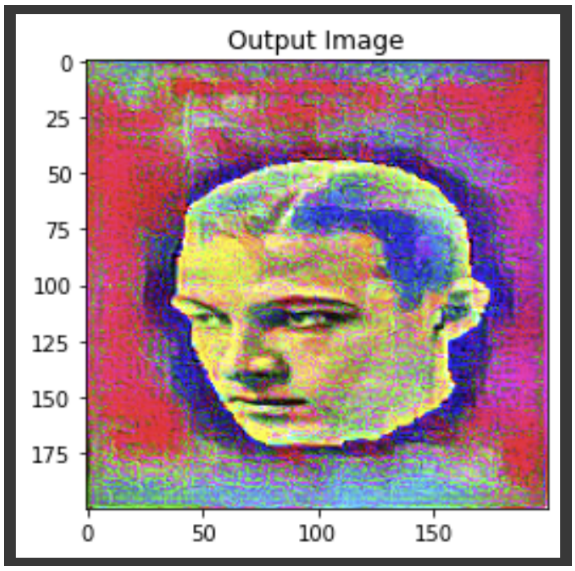
To try even more ways of getting interesting movie posters, we used foreground and background detection which separates the background and the foreground of a certain image. In the case below, we were able to take out the face from the image and then run style transfer on both the background and the foreground in order to create different versions of the movie poster. We used cv2.grabCut to take out the face and we ran a VGG19 model on these two different images.











Conclusion and Takeaways