Artistic Style Transfer using Convolutional Neural Networks

E4040.2018Fall.UPSR.report
K Sheetal Reddy kr2793, Ujjwal Peshin up2138
Columbia University

Abstract

Works of artists like Van-Gogh and Pablo Picasso can easily be recognized by their style. This paper focuses on the use of convolutional neural networks to be able to recreate these classic styles and many other onto an image of your choice. Pre-trained networks are used to be able to extract the style and the content from the images. The objective is to be able to do justice to both the content and style image. The challenges being the time required to produce the image. We were successfully able to re-create the images

1. Introduction

Neural networks have been leveraged for image analysis, since 1968 when it was discovered that there are two basic type of visual cells that human beings use for perception.
Convolutional neural networks are the go to for extraction of features from the images. The property of these networks that lends so well for their use in image processing is the fact that they require very little pre-processing of the data and that they are space invariant. Hence a change in the location of a feature would not affect the network's ability to recognize it.
The filters to pick out features which until then were engineered by experts, now need not be explicitly defined and are learnt as a part of the training of the network.

Since then there have been a lot of attempts since3, to be able to design architectures for work best for specific purposes. One such architecture is the VGG-19[1]. This architecture published in 2014 showed hoe pushing the depth of the layers to ~19 while keeping the size of the filters small provided significant improvements over other architectures. At that time their networks generalized well to other datasets and were the state of the art models. The best performing networks were made available for use.

Convolutional Neural Networks were used for texture identification in the 1997[2] when a paper proposed a theory. This theory combines filtering theory and Markov random field modeling through the maximum entropy principle, and interprets and clarifies many previous concepts and methods for texture analysis and synthesis from a unified point of view. This has now moved onto using neural networks and in specific CNN's to be able to recognize and extract features out of images.

CNN's when trained on an image, recognize different level of features at different depths. It has been noticed that as we move deeper along the layers of the neural network the output maps are more focused on the "content" of the image than the actual values of the pixels that create it [3].

This information leads one to believe that the style and content of the images can hence be extracted as two separate features of the image and are distinguishable. The image that is generated as a combination, is able to retain the objects and their positions as present in the content image, but picks up the colors and textures of the style image.

To be able to pick out the "style" of an image, a set of correlation of features are used to make the style invariant of the position in the image. It is also important to consider the emphasis that we would be basing on both style and the content of the image. A higher emphasis to the content of the image would show us a great reproduction of the image and the objects in it but wouldn't have much of the style encapsulated. Similarly higher emphasis on the style would lead to an image having the style of the artwork but the objects that are actually part of the image are hazy and ambiguous.

This style extraction from images also has applications in recognizing the style of various images and in style classification, which can be used as a tool to identify the period that an art piece was from.

The main challenge in this project is that, it is not an exact science to calculate the parameters and hence requires a lot of experimentation to figure out the right configuration.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

The paper that we are working on is "A Neural Algorithm of Artistic style" by Leon A. Gatys, Alexander S. Ecker, Matthias Bethge [4]. This paper focuses on the idea of optimizing two types of losses to obtain the result.

The paper begins by using an image of white noise, that is used to calculate the associated losses: Content loss : This calculates the distance between the feature map at layer l, of the content image that we provide as target and the white noise image that has been generated.

Style loss: To calculate this, it was required that this entity be spatially invariant and hence a correlation of the feature maps at a layer was used instead of using the feature maps themselves. This correlation matrix is referred to as the gram matrix. Hence the loss is the distance between the gram matrix of the style image and the white noise image

The overall objective is to minimize the total loss given by :

Total loss = Style_weight *Style loss +
Content_weight *Content loss

This loss gets evaluated at each iteration and the values of the pixel in the white image are changed accordingly. These changes in pixels finally lead to an image that is the output and encapsulates both the style and content in some manner.

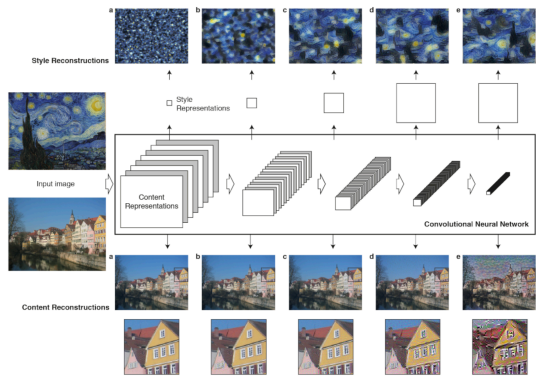


Figure 1: Style and Content Extraction

2.2 Key Results of the Original Paper

The key results and findings of the paper are that the content and the style of an image are actually separable and can be seen as functions of the feature maps that are outputs at different layers of the convolutional network.

Figure 2, represents the findings based on style capture by the different layers of the CNN. The vertical column represents the Content_Weight/Style_Weight ratios varying between 10^{-5} to 10^{-2}. The rows represent the different layers of the VGG-19 architecture that were used for the style extraction.

Figure 2: Different Style extractions

It was found that the 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.

Figure 3, shows the ability of the network to be able to transform the image based on different style inputs and to be able to make coherent images that do justice to both the content aspects and well as the style aspects.

Figure 3: Output of the network on same content image and different style images.

3. Methodology

This section provides an overview of the objectives, challenges and the formulation of the problem that we are intending to solve. We are basing the solution on methodology provided by the original paper. We chose to work with a pretrained VGG-19 model to extract the feature maps.

We chose to work with the available Adam optimizer. We decided to try giving different weights to the style layers associated with the style loss to see how it would impact the overall generation of the image.

3.1. Objectives and Technical Challenges

The objective of paper is to be able to successfully recreate the style and the content. This problem can be boiled down to us trying to minimize the total loss, to keep updating the parameters (pixels in the image) to match the values of both.
The parameters in this case are the size of the image and its values in RGB. Hence with the size of the image the number of parameters to be learnt will keep growing and might become too large.
We see that the results are dependent on the hyper-parameters ratio Content_Weight/ Style_Weight and hence finding a good ratio is a challenge.
Since we would like to consider the impact of having different weights attached to different style layers, finding the corresponding weights for each layer is also a challenge.
With different parameters that can only be determined by trial and error we would need several test benches.

3.2. Problem Formulation and Design

Formulation of the key minimization problem:
As done in the original paper.
The trained weights of the VGG-19 network are available of use and were used for feature map extractions. The fully connected layers at the end of the network were removed.

The responses are stored as feature banks, where if N features are applied and output size is M (width*height) .
Let the p represent the content image and x represent the white noise, Pl and Xl are their corresponding feature representation at layer l. The feature responses are of the size $N * M$.

$$Lcontent\ (p, x, l) = \frac{1}{2}\sum(Plij - Xlij)2$$

The summation is over i and j, that are the length and breadth of the feature bank.
The gradient is computed to make changes to the image.

We need to now find a way to extract the style representations, which were considered to be the correlation of features in those layers known as the Gram Matrix

$$Glij = \sum_{k}^{N} Flik * Fljk$$

Let a be the style image and Al be the gram matrix at layer l.

$$El = \frac{1}{4 * Nl^2 * Ml^2}\sum(Glij - Alij)2$$

$$Lstyle(a, x) = \sum wl * El$$

Where *wl* represents the weightage for each style layer during the process of loss calculation.

The total loss is calculated as a function of the content loss and the style loss. The weightage that we give the content image is represented as *alpha* and the weightage given to the style of the image is represented by *beta*.

The final equation hence looks like the following:

$$Ltotal = alpha * Lcontent + beta * Lstyle$$

Our objective is to minimize this function, and hence we use gradient descent to find the minimal values and try to push the values in the image in a format such that it reduces the loss. This process meanwhile ensures that the white noise image is being constructed in a manner that it encodes the content as well as the style.

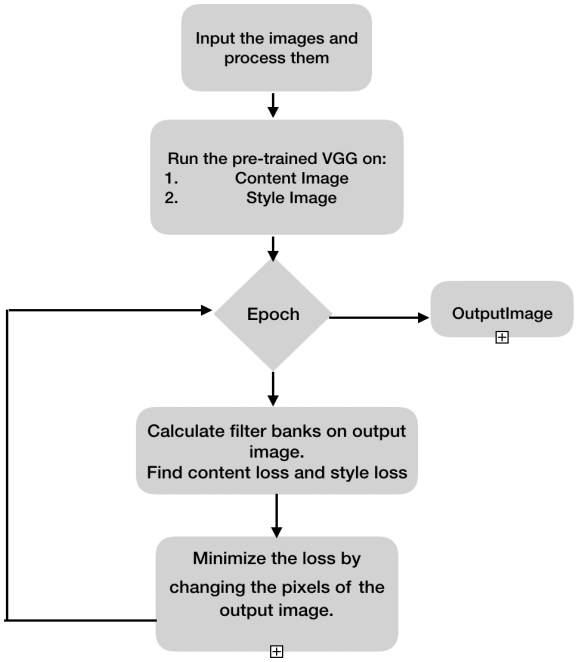


Figure 4: Flowchart for the problem

4. Implementation

The following section describes our implementation of our solution to the problem, in section 4.1 the network is explained and in detail. Section 4.2 provides pseudo code for all the functions used in our program. We also outline the various approaches used and challenges faced.

4.1. Deep Learning Network

Architectural block diagram:
The algorithm leverages VGG's ability to identify the features in an accurate manner. The model we chose is VGG19. The model looks as below.

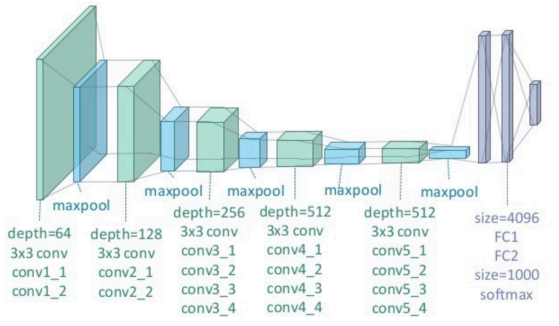


Figure obtained online

We did not utilize the fully connected layers towards the deeper end of the network. The trained weights were taken from the source:
http://www.vlfeat.org/matconvnet/pretrained/
The input dimensions of these are 224*224*3*10. Hence, we need to scale our images to 224*224 and replicate it 10 times as an input to the VGG network.

VGG accepts images which have been normalized by subtracting their mean, hence, the input images need to be normalized before the input.

Data considerations:
This network is not a neural network in the traditional sense. There is no explicit training or testing data present. The weights that we are trying to learn using the optimization is the output.
In the VGG model, since we use a pretrained model's weights we do not need data for the training of the VGG model either. The only data that we would require would be the content image (of choice) and the style image (of choice).

Training Algorithm Details:
The training algorithm can be broken down into these following steps.

- Obtaining a VGG model.
- Normalizing the white noise image.
- Running the VGG model on the normalized input of the noise input to extract values at each layer.
- Calculate the content loss.
- Calculate the style loss.
- Calculate Total loss.
- Initialize the optimizer to minimize the total loss
- Keep changing the parameters (the white noise image pixels) to keep reducing the loss.

4.2. Software Design

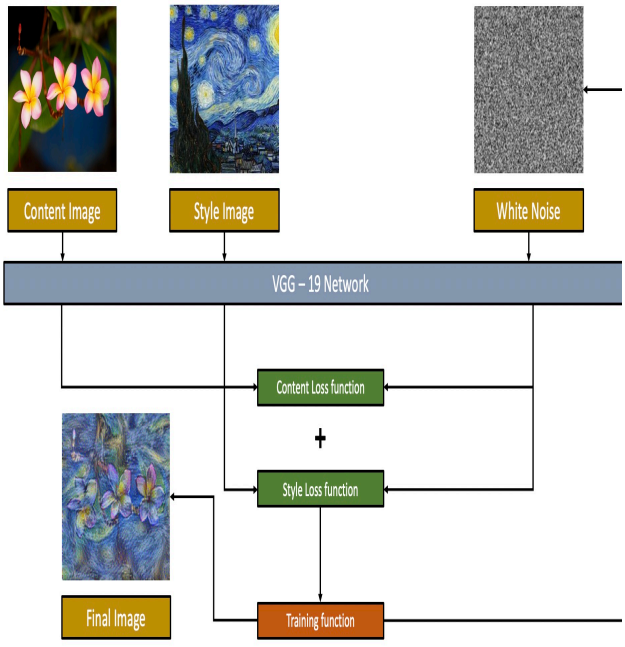


Figure: Flow chart for the overall algorithm

We have defined helper functions for the algorithm.
The algorithms for the functions as follows.

READ IMAGE:

- Load the Image
- Resize the Image

WHITE NOISE:

- Generate Random numbers.
- Add with a probability to content image.

CONTENT LOSS :

- Import the VGG model.
- Normalize the content image.
- Run the VGG for content image and the noise image.
- Find the difference between both the outputs at the content layer.
- Sum them up for each feature value
- Multiply this by 0.5

STYLE LOSS :

- Import the VGG model.
- Normalize the style image.
- Run the VGG for style image and the noise image.
- Evaluate the gram matrix.
- Reshaping the feature into num_filters * (height*width).
- Multiply the transpose of this reshaped vector with itself to find correlations.
- Find the difference between both the gram matrix at the style layers.
- Sum them up for each feature value
- Give each style, a weightage depending on the kind of features that are required.

TRAIN:

Same as the algorithm given in 4.1

5. Results
5.1. Project Results

We decided to conduct various experiments on the hyperparameters to observe the different effect that they would have on the final image. These experiments would also help us decide the parameters for the most suitable output for the images.
The experiments can be divided into the following sections.

All experiments were run on GCP, with 4 x NVIDIA Tesla K80 GPUs and n1-standard-4 (4 vCPUs, 15 GB memory).

The content and style image are as follows:


1. **Experiments based on the content weight and style weight:**

The aim of these experiments is to be able to determine a ratio of these values so there is no overpowering effect of the content or the style in the final image. The style weights are all 0.2.

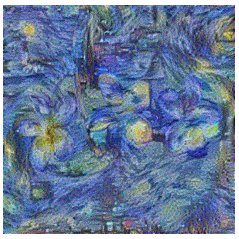
Content weight :10
Style weight :10
Time taken :150.23 sec

Content weight :10
Style weight :100
Time taken :150.35sec

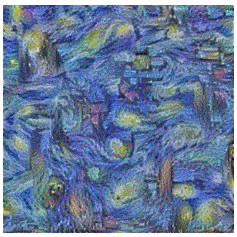
Content weight :10
Style weight :1000
Time taken :150.23 sec

Content weight :10
Style weight :10000
Time taken :150.35sec

Content weight :10
Style weight :100000
Time taken :150.23 sec



Content weight :10
Style weight :10000000
Time taken :150.35sec



2. **Experiments based on the content layer selection:**

The weights for content and style were chosen to be for this set of experiments are 10 and 10000. The time taken for all of the below was ~152 sec.

Content layer : relu1_2

Content layer : relu2_2

Content layer : relu3_2

Content layer : relu4_2

Content layer : relu5_2

3. Experiments based on the style layer selection:

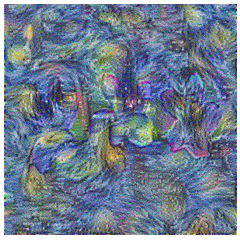
For these set of experiments the content layer relu4_2 was chosen. To pick a certain layer the style weights of the rest of the layers were set to zero. For instance, to pick layer relu1_2 the style weight vector was [1, 0, 0, 0, 0].Time ~118.7 sec.

Style layer : relu1_2

Style layer : relu2_2

Style layer : relu3_2

Style layer : relu4_2



Style layer : relu5_2

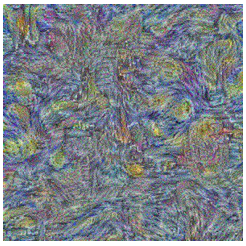
4. Experiments based on combination of style weights

After the experiments on the different layers, we tried to find a combination of those such that all the features get transferred. weights =[0.3,0.1,0.2,0.3,0.1]

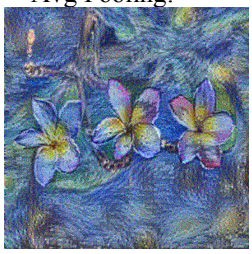
5. Experiments with different pooling:

All the above images were made by changing the vgg,py code (source: anishathalye) to average pooling as recommended in the paper. If the original max pooling is used, the following is the result.

Max Pooling:



Avg Pooling:

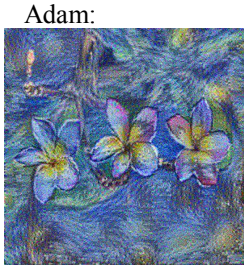


6. Experiments with different optimizers:

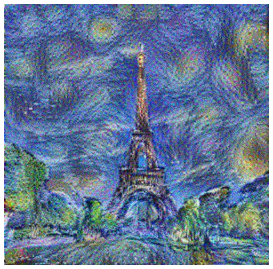
All the above images were made using the tensorflow implementation of Adam optimizer. We tried changing it to:

RMS Prop:

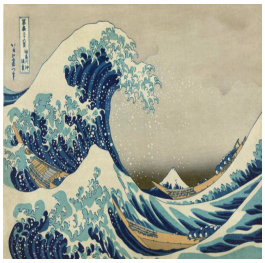
Adam:



7. Experiments with different content image:



8. Experiments with different style image



9. Experiments with white noise image:

We noticed weren't as smooth as excepted, on reason could be the pixels from the white noise that weren't being fully optimized during training, and left the images looking blurry in the middle.

To tackle this problem we decided to let the initial image be a mixture of the content image and a random noise added to it.

5.2. Comparison of Results

Figure : Paper Result

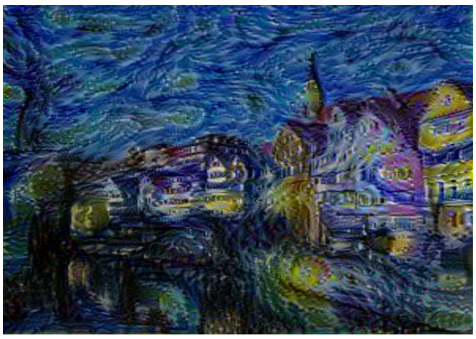
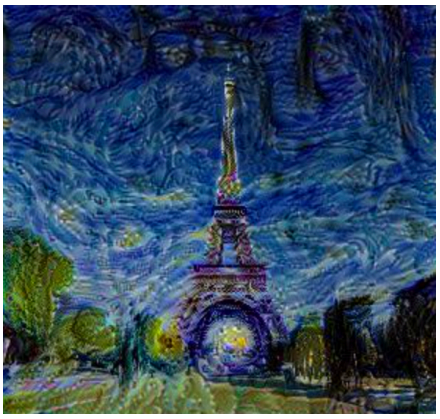


Figure : Our result



To compare our results with the results in the paper, we decided to create the output for a similar input-output scenarios.

Visually we notice that both the results are a successful rendition in the style of the artist but differ in which elements were captured in this rendition.

In the papers result there seems to be more capture of the swirls formed by the stars in the starry night image.

In our result there seems to be more capture of the brush stroke effect as in Van-Gogh's painting and we do notice a general pattern of swirling appear.

This difference could be due to the different weights assigned the styling layers, as observed in the above experiments, conv4_2 picks up a high amount of swirling feature as compared to the others.

Another reason for the difference could be the optimizers used, RMS shows an increase in the losses after some point where it becomes more stable. The different optimizer functions could be looking at different minima's for the loss function.

Another result of ours that we can compare with the paper's was the testing with the content_weight and style weight ratios.
The results in this case as expected were very similar to the paper's (as in experiment1) where increasing the style_weight resulted in a dominating style component and increasing content_weight resulted in increasing content component.

The paper also discusses the effect of using various layers and how they extract different features which has been recreated in experiment 3). This shows which layers contribute to what kind of features in the style image.

5.3. Discussion of Insights Gained

The following were the insights gained from this project:

1. The ability of the layers in VGG to recognize different features
2. The ability of the layers in VGG to be able to differentiate content and style.
3. Use of correlation as a metric to find the likeliness of features occurring together.
4. With the number of parameters being the pixels in the image it was very important to fine tune the parameters, rather than increasing the resolution to get better outputs.
5. The relevance of content_weight and style_weight to balance both the components, as we can see in experiment 1.
6. In Experient 2, we see that the use of the content_layer , using a lower content_layer only captures in colours of the image clearly but is not very significant in terms of edges which are clearly more visibke as we move deeper into the network.
7. In Experiment 3,we see how different style layers affect the features drawn, it shows how higher level features like swirls are picked up the 4th layer, but brush strokes were in layer 2.
8. Experiment 4 shows how adjusting the weightage given to these layers, can improve the overall rendition of the image.
9. Experiment 5, was a trail of the usual max_pooling as the pooling criteria instead of the recommended avg_pooling in the paper. This could probably be because the style is very dominated by blue, and the max would pick out more B-pixel dominant values throughout the layers.
10. Experiment 6 shows that RMS has a slightly better chance of capturing brush strokes when compared to Adam , this could be because RMS doesn't search in osciallting directions.
11. Experiments 7 and 8 were a test on different style and content images
12. One problem we noticed throughout the reproduction of our images what that they were not smooth and seemed to be really blurry due to the random image not optimizing all the pixels, hence we decided to generate the initial image as a mixture of the content image and the white noise which gave us much better results with regards to visual smoothness.
13. The reproductions are very sensitive to parameters, a diff style image might require some fine – tuning to obtain good results.

6. Conclusion

The project was successful in being able to achieve the goal of mixing the content and styles of two different images. The algorithm made use of pre-trained VGG's to extract features.

For further improvement we would recommend being able to generalizes these parameters such that they are not very dependent on the style image. This project can also be extended to styling videos by making a frame by frame stylization of the images in the content video.

7. Acknowledgement

We would like to thank Professor Zoran Kostic and all the TAs of ECBM4040 -Fall 2018 for their guidance

8. References

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

[1] https://bitbucket.org/ecbm4040/2018_assignment2_kr2793/src/master/UPSR.project.kr2793.up2138/

[1] Very Deep Convolutional Networks for Large-Scale Image Recognition, Karen Simonyan Andrew Zisserman

[2] Zhu et. al., "Filters, Random Fields and Maximum Entropy (FRAME): Towards a Unified Theory for Texture Modeling", *International Journal of Computer Vision 27(2), 107–126 (1998)*

[3] A Neural Algorithm of Artistic Style
Leon A. Gatys, Alexander S. Ecker, Matthias Bethge

9. Appendix

9.1 Individual student contributions in fractions - table

	kr2793	up2138
Last Name	Reddy	Peshin
Fraction of (useful) total contribution	1/2	1/2
What I did 1	Code for the loss calculations	Code for the training of the network
What I did 2	Report: Introduction, Background, Results, Insights	Report: Methodology, Formulation, Implementation
What I did 3		