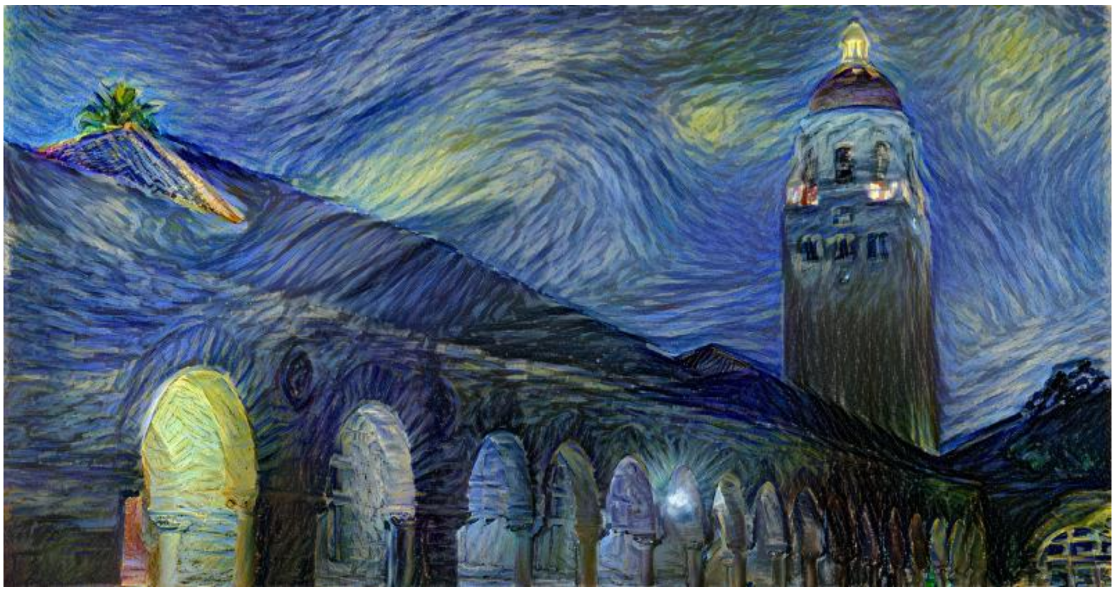
Software Engineering Department
Ort Braude College
Course 61771: Final Project in Software Engineering

Image Style Transfer Using Convolutional Neural Network



Karmiel – February 2019

Authors

Shahar Assenheim	301766341
Mohammad Siah	305308371

Supervisor
Dr. Renata Avros

Contents

1. Introduction 3
2. Background and Related Work 4
   2.1 Image Representation 4
   2.2 Convolutional Neural Network 4
   2.3 VGG Network 5
      2.3.1 Convolution 5
      2.3.2 Pooling 6
      2.3.3 Non-Linearity (ReLU) 6
      2.3.4 Zero Padding 6
      2.3.5 Stride 7
   2.4 Tensonflow 7
3. The Proposed Method Details 7
   3.1 Content Representation 8
   3.2 Style Representation 8
   3.3 Style Transfer 9
   3.4 Summary of the Process 10
   3.5 Theoretical Results 10
   3.6 Effect of Different Layers of the CNN 11
4. Preliminary Software Engineering Documents 12
   4.1 Requirements (UseCase) 12
   4.2 Design (Class Diagram) 13
   4.3 Sequence Diagram 14
   4.4 GUI Designing 15
   4.5 Testing Plan 20
5. Results and Conclusions 21
   5.1 Results 21
   5.2 Conclusions 26
6. References 26

Abstract. *Rendering the semantic content of an image in different styles is a difficult image processing task. Here we propose a new approach to extract content and style from an image and combined them together in order to get an artistic image.*
As opposed to previous approaches our system based on a Convolutional Neural Network that optimized for object recognition, which make high level image information explicit. In this approach we use the image representations derived from the neural network to separate and recombine the content of a photograph and the style of well-known artworks.
Moreover, the proposed work offers a path forward into the deep image representations learned by Convolutional Neural Networks and demonstrates their potential for high level image synthesis and manipulation.

Keywords: *Style Transfer, Convolutional Neural Network, VGG Network, Feature maps, Style loss, Content loss, Gram Matrix.*

1. INTRODUCTION

Painting is a popular form of art, for thousands of years people have been attracted by the art of painting with the appearance of many fantastic artworks. In the past, re-drawing a picture in different styles required a very skilled and trained artist and lots of time. In the last 20 years, the art theories behind the artworks have begun to get the attention not only of the artists themselves but also of many researchers in the field of computer science. There are many studies that exploring how to automatically transform a picture into an artwork and give it a special style and texture, in other words to take content from one picture and give it a style of another picture or famous artwork. This field of style transferring is in constant progress and today is a well-established field in the community of computer graphics, but this problem of separating content and style of an image is still considered one of the most difficult problems in the field.

Many algorithms that published for this issue suffer from the fact that they are appropriate for specific styles and cannot easily be extended to other styles, and most of their results are not particularly impressive. Recently inspired by the power of the Convolutional Neural Network (CNN) published many researches that show how to use CNN to reproduce the content of an image and the style of famous artworks, these researches showed that by using a Neural Network that trained to identify objects in images, it is possible to do this reproducing. The results of these researches showed that the content and style in an image can be separated, which gave the possibility to change the style of the image while preserving the desired semantic content. One of the breakthrough researches was that of Gatys et al. [1], who suggested using the CNN to separate the content of one picture and combine them with the style of another image or artwork and creating a new impressive art image. The idea behind Gatys et al. [1] algorithm is to optimize a random image so that it minimizes the difference between the random image to the content from one image and the style from another image.

Since Gatys et al. [1] algorithm has been published, it breaks the constraints of the old approaches and has opened a new field called "Neural Style Transfer" which is the process that uses the Neural Network to make content images in different styles. In this paper we provide an overview of our algorithm (which based on Gatys algorithm) and it’s amazing results, in addition we provide an overview on every step in the CNN process and how it can be used for manipulating the content and the style of natural images, an example of a generated image from the algorithm is presented in Figure 1 that shows transferring the style of Van Gogh’s “The Starry Night” onto a photo of night time in Stanford campus.

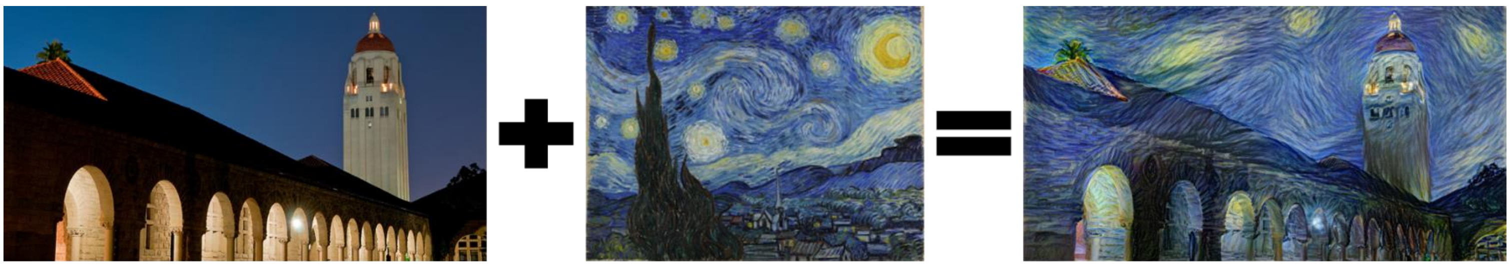

Content: Night Time of the Stanford Campus
Style: Van Gogh's "The Starry Night
Style Transfer Result

Fig. 1 Example of a generated image from the algorithm

2. BACKGROUND AND RELATED WORK

Most previous style transfer algorithms based on non-parametric methods for texture synthesis. For example, Efors and Freeman [2] which computes an output texture image by merging together patches taken from the input image. Hertzman et al [3] use image analogies to transfer the texture from an already styled image onto a target image. Ashikhmin [4] starts from a sample image and generates a new image of arbitrary size the appearance of which is similar to that of the original image.

Despite these algorithms achieve good results, they all suffer from the same basis limitation, They use only low-level image features of the target image to imform the texture tranfer, and so unlike the algorithem we offer, they give less impresive results.

2.1 Image Representation

In computer graphics every image represent as a matrix of pixel values. Most commonly, 1 Byte sized pixel is used, then the possible range of values a single pixel can represent is [0, 255]. With colored images, particularly RGB (Red, Green, Blue) images, there are 3 color channels that make the input 3-dimensional. Therefore, for a given RGB image of size, say N x M (Width x Height) pixels, we'll have 3 matrices associated with each image, one for each of the color channels, Thus the image is represented as a N x M x 3 3D matrix. An example of an RGB image is presented in Figure 2.

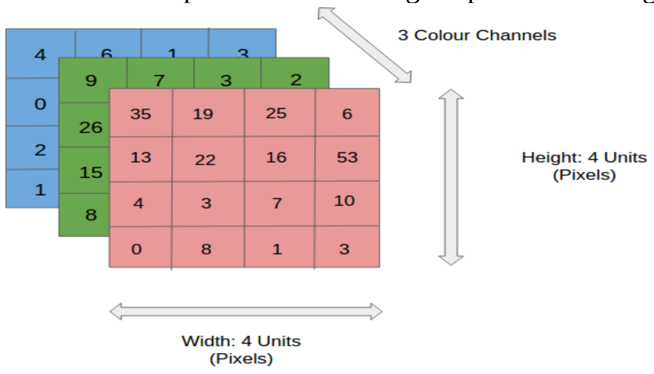


Fig 2.An example of RGB image representation – 3Dmatrix

2.2 Convolutional Neural Network

The class of Deep Neural Networks that are most powerful in image processing tasks are called Convolutional Neural Networks (CNN). CNN have been successful in computer vision related problems like identifying faces, objects and traffic signs apart from powering vision in robots and self-driving cars.

If we take a CNN that has already been trained to recognize object within images, then that network can develop some representations of the content and style contained in a given image. One of the famous Convolutional Neural Networks that been published called VGG network that we expand about it in section 2.3.

2.3 VGG Network

In 2014 the winner of the ImageNet challenge was the Visual Geometry Group (VGG) Network, the ImageNet Large Scale Visual Recognition Challenge has been running since 2010 and has become the standard benchmark for large scale object recognition. This network has two versions VGG16 and VGG19, in our algorithm we use the VGG19 that consists 16 layers of convolution and ReLU non-linearity, separated by 5 pooling layers and ending with 3 fully connected layers.

Since we need only convolutional layers of the VGG model, we use the model without the 3 fully connected layers as a basis for trying to extract content and style representations from images. VGG19 Chart presented in Figure 3.

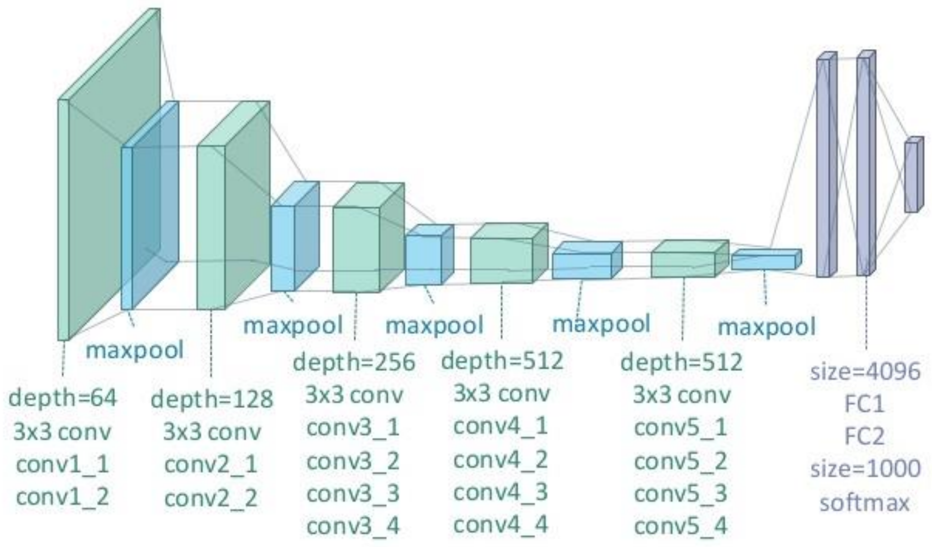


Fig. 3 VGG19 Chart, each convolution block contains different amount of filters. First block has 64 filters with a pooling step, 2nd block contains 128 filters with a pooling step, 3rd block contains 256 filters with pooling step, 4th block contains 512 filters with a pooling step, 5th block contains 512 filters with a pooling step and the 6th block contains the fully connected layers that we are not using in the proposed algorithm.

2.3.1 Convolution

Convolutional Neural Networks derive their name from the “Convolution” step. The main purpose of Convolution in case of a CNN is to extract features from the input image. Convolution preserves the relationship between pixels by identifying image features by using small filters of input data.

Every image can be considered as a matrix of pixel values, the VGG network uses different amount of filters in every convolution block, each filter is a 3 x 3 window of weights. By scanning the input image with a filter we get a feature map which describes the presence of a given feature throughout the input image, example of the convolution shown in Figure 4. The result we get from each convolution layer is a 3D matrix N_{width} X N_{height} X $N_{convolution}$ while N_{width} is the image width, N_{height} is the image height and the $N_{convolution}$ is the number of the filters in the layer.

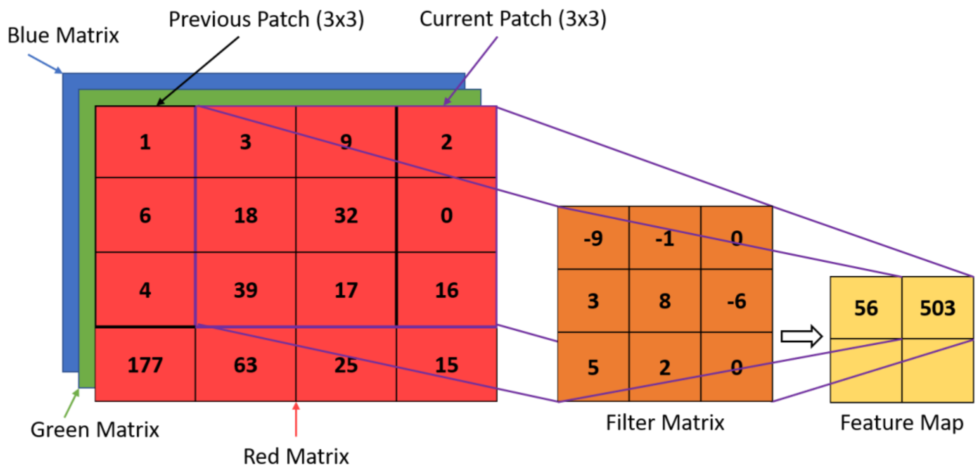


Fig. 4 Explaination of the convolution process, we slide the filter matrix over the original image (on each of the RGB matrices) by one pixel which called stride and for every position we compute multiplication between the 2 matrices (input image and filter matrix) and add the multipication result to get the final integer which forms the single element of the feature map.

2.3.2 Pooling

The pooling step reduces the dimensionality of each feature map, but retains the most important information, this step can be of different types: Max, Average, Sum, etc. In case of the Max pooling we define a 2 x 2 window and take the largest element from the rectified feature map within that window and then slide the 2 x 2 window by 2 cells which called stride, because we slide 2 cells at a time (stride = 2) the feature maps in each convolution block will be reduced in half. Instead of taking the largest element we can also take the average (Average Pooling) or the sum of all elements in that window (Sum Pooling). For image synthesis we found that replacing the maximum pooling operation by average pooling produces more appealing results.

2.3.3 Non-Linearity (ReLU)

An additional process called ReLU used after every convolution operation. ReLU is an operation that applied per pixel and replaces all negative pixel values in the feature map by zero. Its output is given by: $Output = Max(0, Input)$.

The purpose of ReLU is to introduce the Non-Linearity to the Convolution Neural Network, Convolution is a linear operation (matrix multipication and addition) and since most of the real world data we would want the CNN to learn is Non-Linear (Image Processing), that’s why we introduce a Non-Linear function like ReLU to our CNN.

2.3.4 Zero Padding

The zero padding process is applied to control the size of the feature maps, it is convenient to pad the input matrix with zeros around the border so that we can apply the filter to bordering elements of the input image matrix. We need the convolution step to keep the size of the input image. Since we use 3 x 3 filter, the generated feature map is being 2 pixels less in the width and in the height after the convolution step, by using the zero padding once we can keep the original size of our input image in the generated feature map. Figure 5 shows an example of 1 zero padding.

2.3.5 Stride

Stride is the number of pixels by which we slide the filter over the input matrix. When the stride is 1 we move the filter 1 pixel at a time. In the convolution step we define stride to 1 pixel. When the stride is 2 than the filter jump 2 pixels at a time, in the pooling step we define the stride to 2 pixels. Having a larger stride will produce smaller feature maps. Figure 5 shows an example of 1 pixel stride.

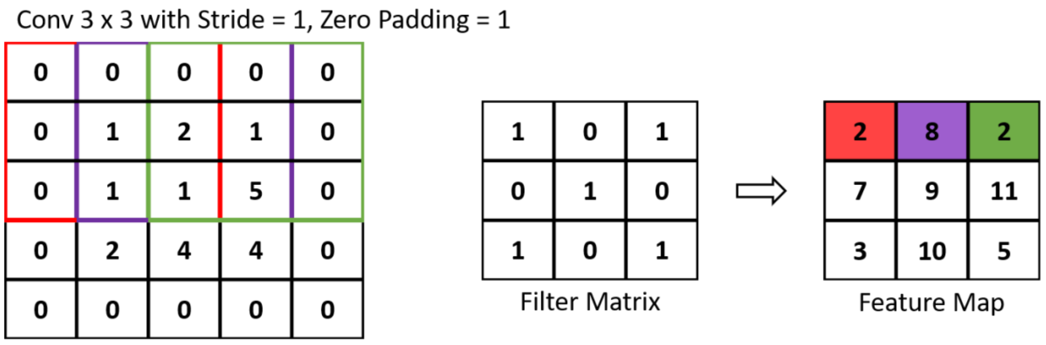


Fig. 5 An example of taking a 3 x 3 image and by adding 1 zero padding and using a stride of 1 pixel we get a feature map with original image size.

2.4 Tensorflow

In our project we used tensorflow in order to work with the VGG network. Tensorflow is a python open source library for numerical computation that makes machine learning faster and easier.

It created by google Brain team, it bundles together many machine learning and deep learning (neural netowrking) models and algorithems makes them useful. Tensorflow can train and run deep neural networks for handwrittin digit classification, image recognition and more. In our research we found that the most recommended API to look for when using neural networks with tensorflow called Keras.

Keras is a high level interface for neural networks that runs on top of multiple backends. Its functional API is very user-friendly and flexible enough to build all kinds of applications. Keras got very popular after its introduction and in 2017 the Keras API was integrated into core tensorflow as tf.keras.

3. THE PROPOSED METHOD DETAILS

The main idea of this paper is that the representations of content and style of an image in the Convolutional Neural Network are separable, for that reason we can manipulate both representations independently to produce new stunning artworks images. To demonstrate this finding, we generate images that mix the content and style representation from two different source images. Using the content and style representations of images, we can measure the content and style similarity (loss) between images separately. Therefore, this problem can be solved as an optimization problem to minimize content and style loss, the algorithm performed gradient descent algorithm on a white noise image to find another image that matches to the content and style representations of content and style input images respectively. Let C and S be the input content and style images, and X be the image that is generated, then this optimization problem is represented by the following equation:

$$X = \arg\min_{X}(\alpha J_{content}(C, X) + \beta J_{style}(S, X))$$

3.1 Content Representation

In general, when Convolutional Neural Networks are trained on object recognition, higher layers in the network capture the high-level content of images, for that reason our algorithm refer to the feature responses in higher layers of the network as the content representation. The purpose of the content loss in our optimization problem is to make sure that the generated image X retains some of the global contours of the content image C, we expect the two images to contain the same content—but not necessarily the same texture and style. each Layer *l* in the network has N_l feature maps each of size $N_h X N_w$ (the height times the width of the feature map), from each layer we get a $N_h X N_w$ grid of N_l dimensional vectors, it is easier to reshape this 3D $N_l X N_h X N_w$ matrix to a 2D matrix $N_l X N_h * N_w$ (Figure 6 shows this process), so the responses in layer l can be stored in a matrix $F^l_{N_l X N_h * N_w}$where F^l_{ij} is the activation of the i^{th} filter at position j in layer l. Let C be the original content image and X the image that is generated, then P^l and F^l are their respective feature representation matrices in layer *l* after passing through the CNN.

We then defined the content loss as the Mean Squared Error between the two feature representations matrices:

$$J_{content}(C, X, l) = \frac{1}{2}\sum_{ij}(F^l_{ij} - P^l_{ij})^2$$

When this content-loss is minimized, it means that each pixel in the feature representation of the generated image in a given layer is very similar to the feature representation of the content image in the same layer. Depending on which layer we select, this should transfer the contours from the content-image to the generated image. In this case, we choose “Conv5_2” as the CNN layer for the content image feature maps, based on Gatys et al. [1] algorithm this layer best represents the semantic content.

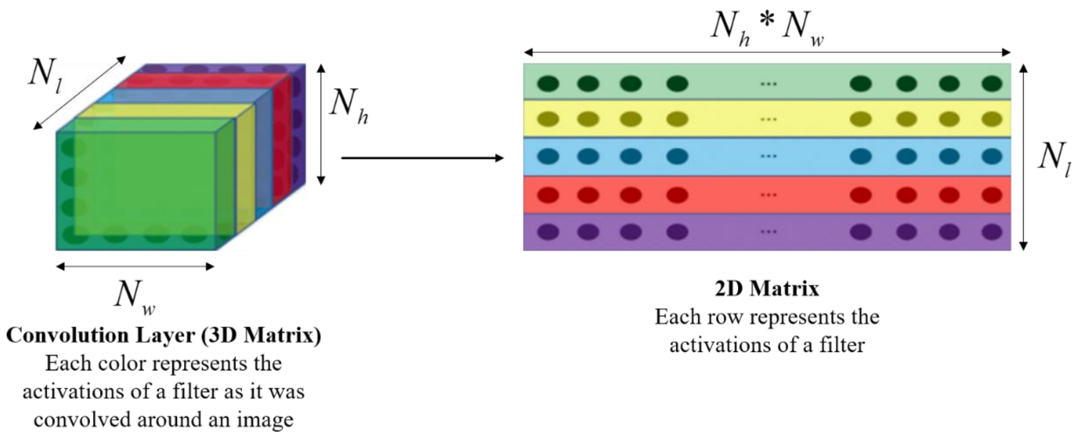

N_l
N_h
N_w
Convolution Layer (3D Matrix)
Each color represents the activations of a filter as it was convolved around an image
$N_h * N_w$
...
N_l
2D Matrix
Each row represents the activations of a filter

Fig. 6 Converting the feature represntation 3D Matrix to 2D Matrix

3.2 Style Representation

The style of the image is represented by the correlations between the different filter responses in any layer of the network. In the described algorithm we compute these feature correlations by the Gram matrix $G^l_{N_l X N_l}$, in other words the Gram matrix comperes how similar one feature map to another, if they highly similar we would expect them to have a large value, and thus G_{ij} to be large. We compute the Gram matrix of an image by passing the image through the VGG network in a chosen layer *l*, we compute the feature representation matrix of the image the same way we did it for the content, as a result we get a 3D matrix $F^l_{N_l X N_h X N_w}$, we reshape it to a 2D matrix $F^l_{N_l X N_h * N_w}$, then we compute that times its own transpose:

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

In Figure 7 we show how the Gram matrix computes in a visual way, it easy to see the correlations between the different filters.

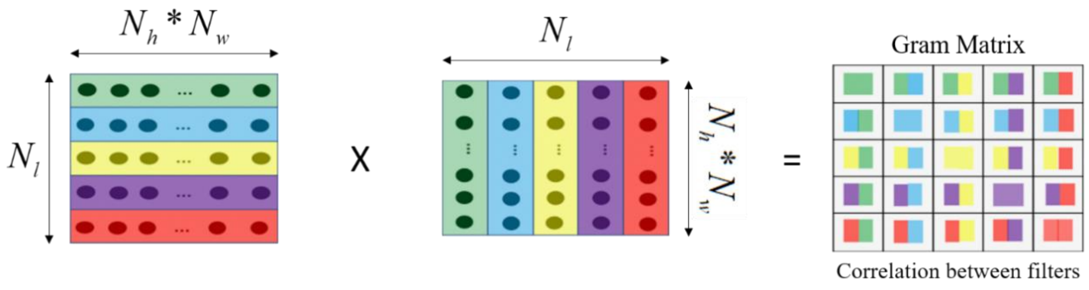


Fig. 7

The loss function for style is quite like our content loss, except that we calculate the Mean Squared Error for the Gram-matrices instead of the feature representations matrices. Let S be the original style image and X the image that is generated, then G^S and G^X their respective Gram matrices in layer *l*.

The contribution of layer *l* to the total loss is then:

$$J^l_{Style}(S, X, l) = \frac{1}{2} \sum_{ij} \left(G^S_{ij} - G^X_{ij}\right)^2$$

Applying this to early layers in the network would capture some of the finer textures contained in the image and applying this to deeper layers would capture more high-level elements of the image style. Based on Gatys et al. [1] algorithm we found that the best results were achieved by taking a combination of shallow and deep layers as the style representation of an image, for that reason we apply the style loss function on different layers to get the total style loss equation:

$$J_{style}(S, X) = \sum_l J^l_{style}(S, X)$$

3.3 Style Transfer

To generate the image that mix the content of a content image (photograph) with the style of a style image (painting) we minimize the distance of a white noise image from the content representation of the photograph in one layer of the network and the style representation of the painting in a number of layers of the CNN. So, let C be the content image, S be the style image and X be the generated image, the total loss function that we minimize is:

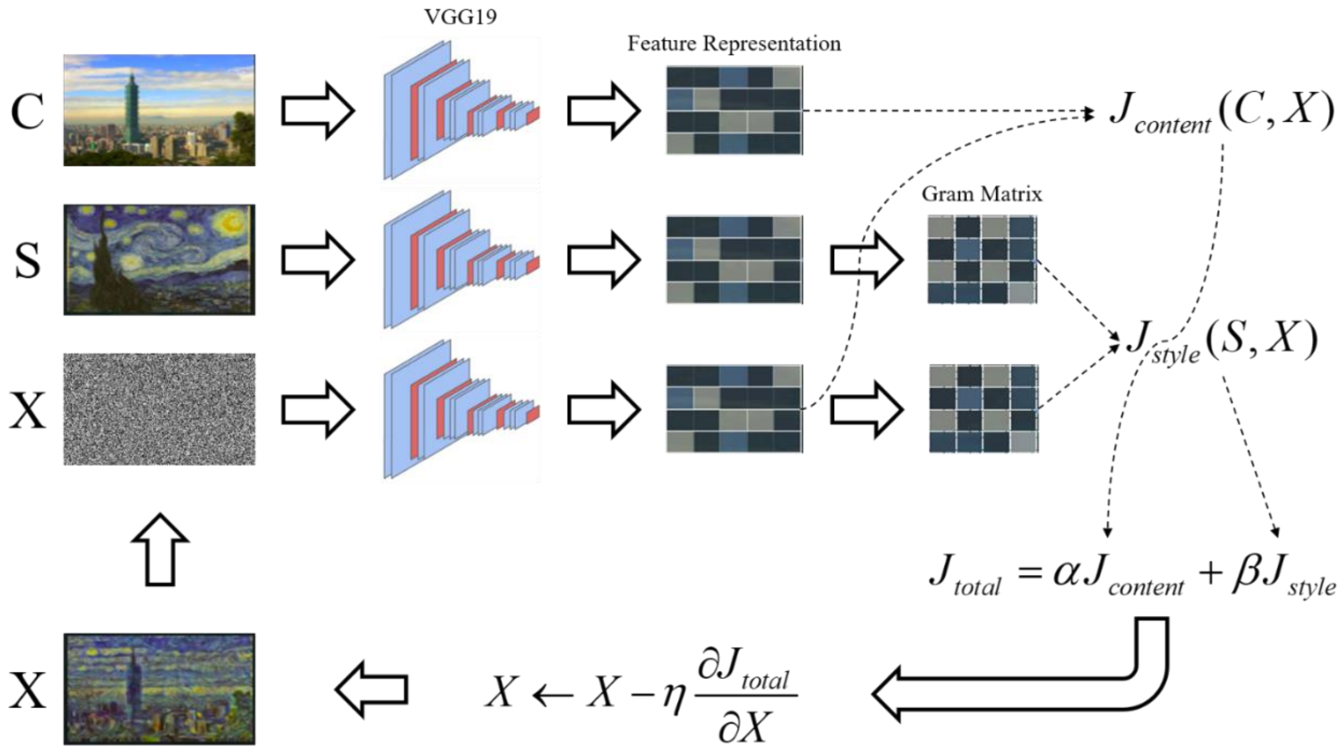
$$J_{total}(C, S, X) = \alpha J_{content}(C, X) + \beta J_{style}(S, X)$$

Where α and β are the weights used to control the overall contributions during the creation of the new image. In the optimization step, the pixel values of X are updated to minimize the total loss function. The gradient with respect to the pixel values $\frac{\partial J_{total}}{\partial X}$ is computed using an optimization algorithm called ADAM. Optimization algorithm It's a way to minimize (or maximize) a function. Since we have a total loss function that is dependent on the generated image, the optimization algorithm will tell us how to change the generated image to make the loss a bit smaller.

3.4 Summary of the Process

Here we provide a pseudo-code and illustration diagram of our algorithm:

1. Pass the Content and Style images into the VGG19 network to get their feature representation matrices.
2. Compute the Style image Gram matrix from its feature representation.
3. Initialize Generated image X from random noise.
4. Pass the Generated image throw the VGG19 network to get its feature representation matrix.
5. Compute the Generated image Gram matrix from its feature representation.
6. Compute content loss: calculate the Mean Squared Error between the Content and Generated feature representation matrices.
7. Compute style loss: calculate the Mean Squared Error between the Style and Generated Gram matrices.
8. Compute total loss: sum the content and the style losses.
9. Compute the gradient by using an optimization algorithm.
10. Update the Generated image pixels according to the gradient.
11. Return to 4th step.


VGG19
Feature Representation
Gram Matrix
C
S
X
X
$J_{content}(C,X)$
$J_{style}(S,X)$
$J_{total} = \alpha J_{content} + \beta J_{style}$
$X \leftarrow X - \eta \frac{\partial J_{total}}{\partial X}$

3.5 Theoretical Results

Based on Gatys paper and our research we present in Figure 8 some of our expected result images, our goal is to achieve similar and even better results, while maintaining good runtime complexity. We estimate that the number of iterations required to obtain a high-level image is around 1000 iterations, while matching the content representation on layer 'conv5_2' and the style representation on layers 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1', 'conv5_1' of the convolutional neural network.

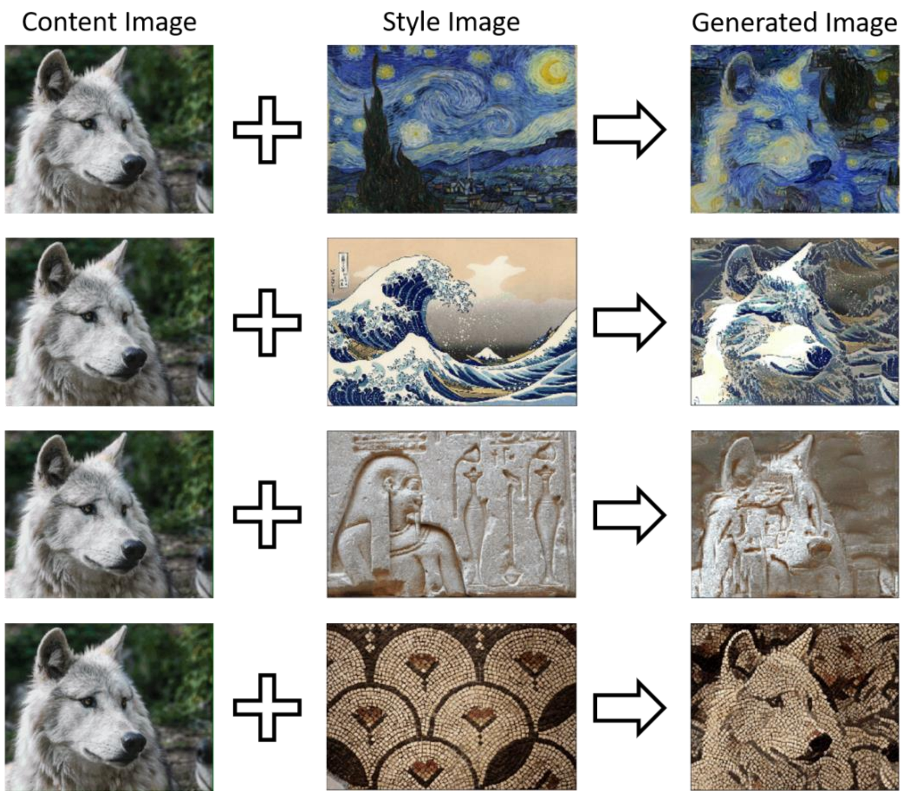


Fig 8. Expected result images.

3.6 Effect of different layers of the CNN

Every layer in the Convolutional neural network produce different feature representation of an image, in our research we found that in order to get the best results for the style representation we must combine the results from several layers together and not using only one, in Figure 9 the upper row shows how texture patterns are getting more global in higher layers. For the content representation we find out that there are no benefits by combining the results from different layers, in Figure 9 the bottom row shows that the content information is preserved even in higher layers, although detailed pixel information is lost. for that reason, in our algorithm we will use the 'conv5_2' layer for the content representation and 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1', 'conv5_1' layers for the style representation.

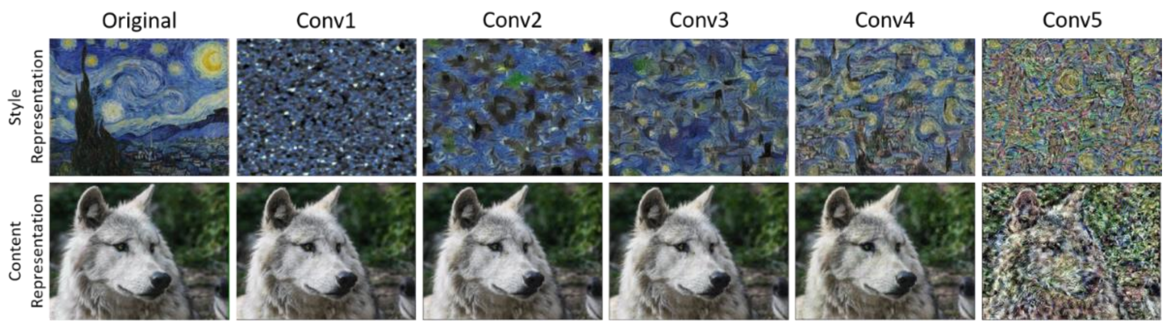


Fig. 9. the differences between the convolution layers.

Figure 10 presents how by combing of many different layers from the network we can capture both, the finer textures and the larger elements of the original style image.

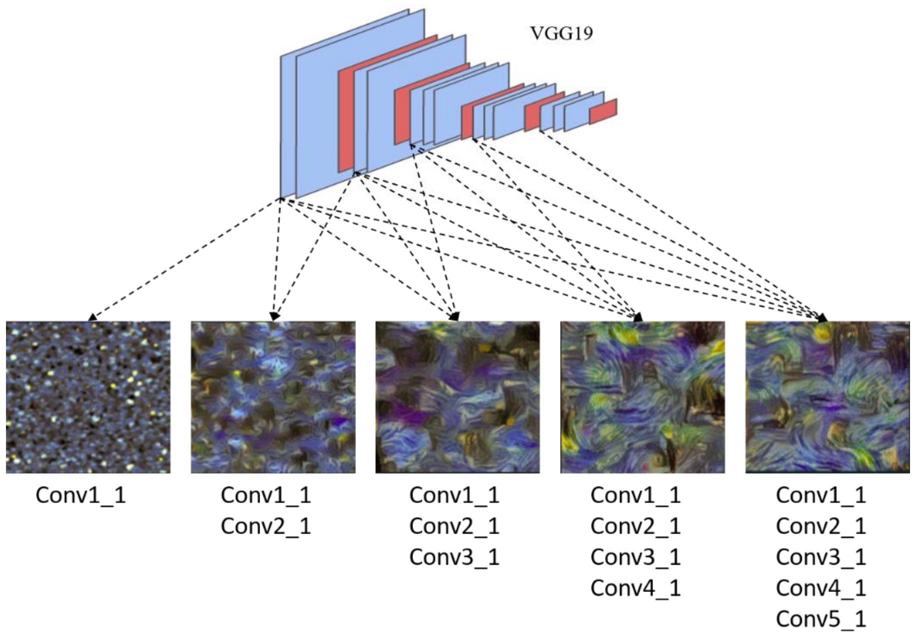
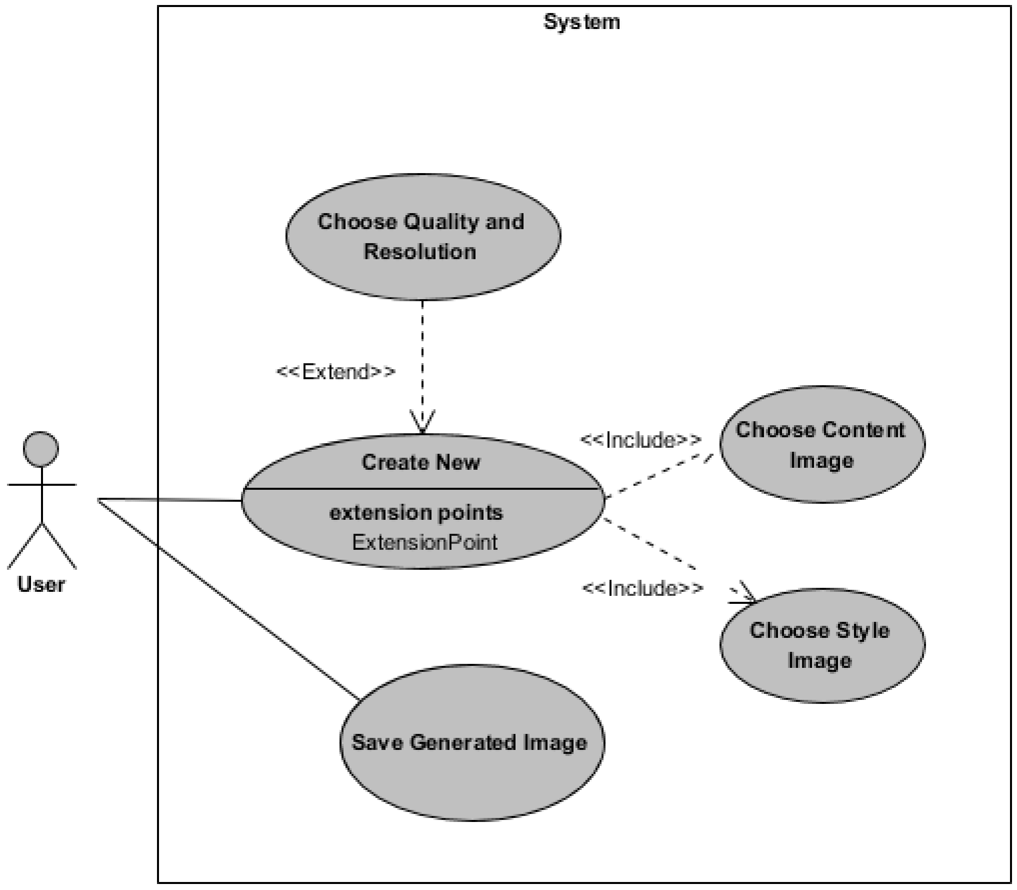


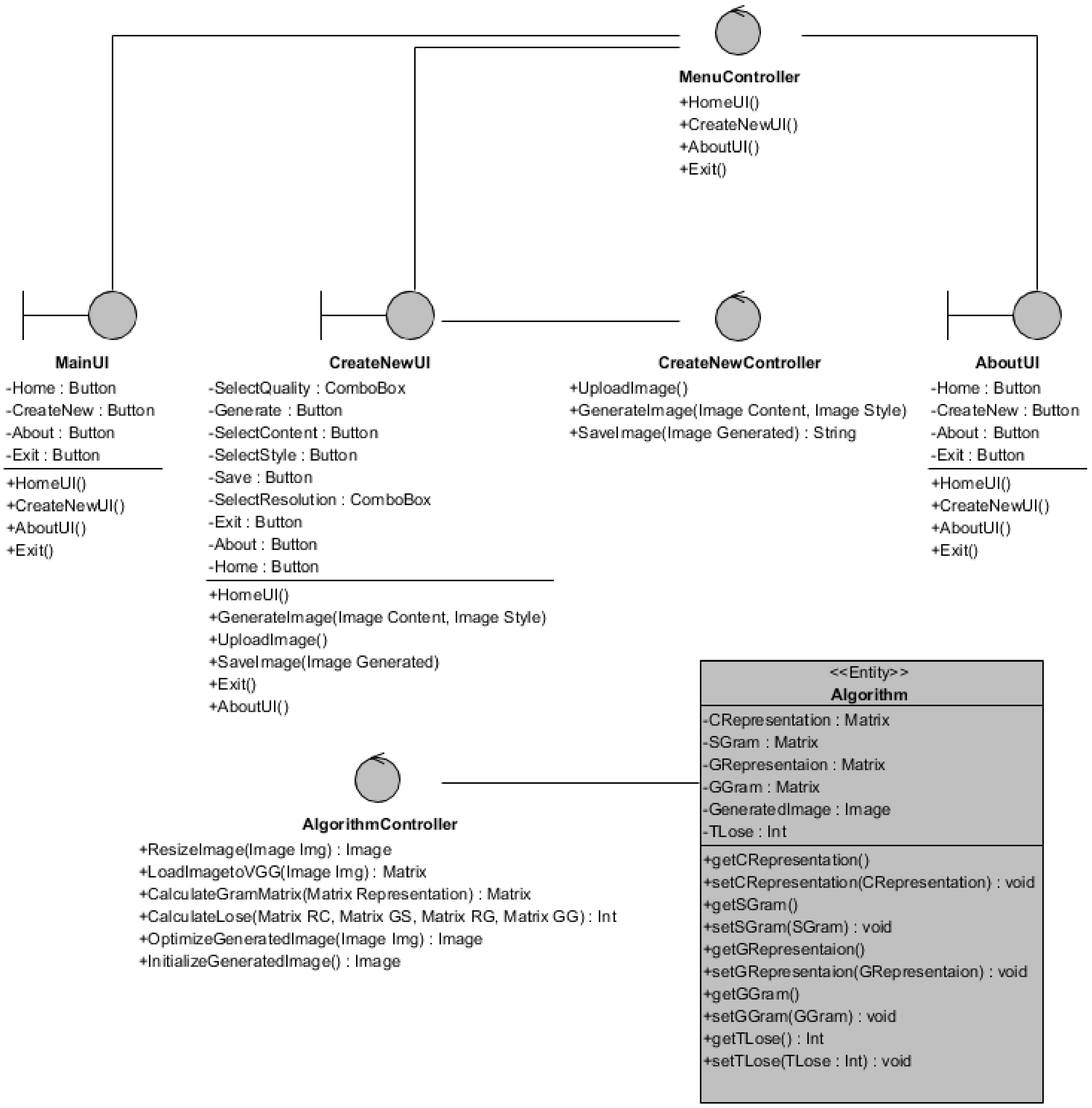
Fig. 10. The result of combining different layers from the network.

4. PRELIMINARY SOFTWARE ENGINEERING DOCUMENTS

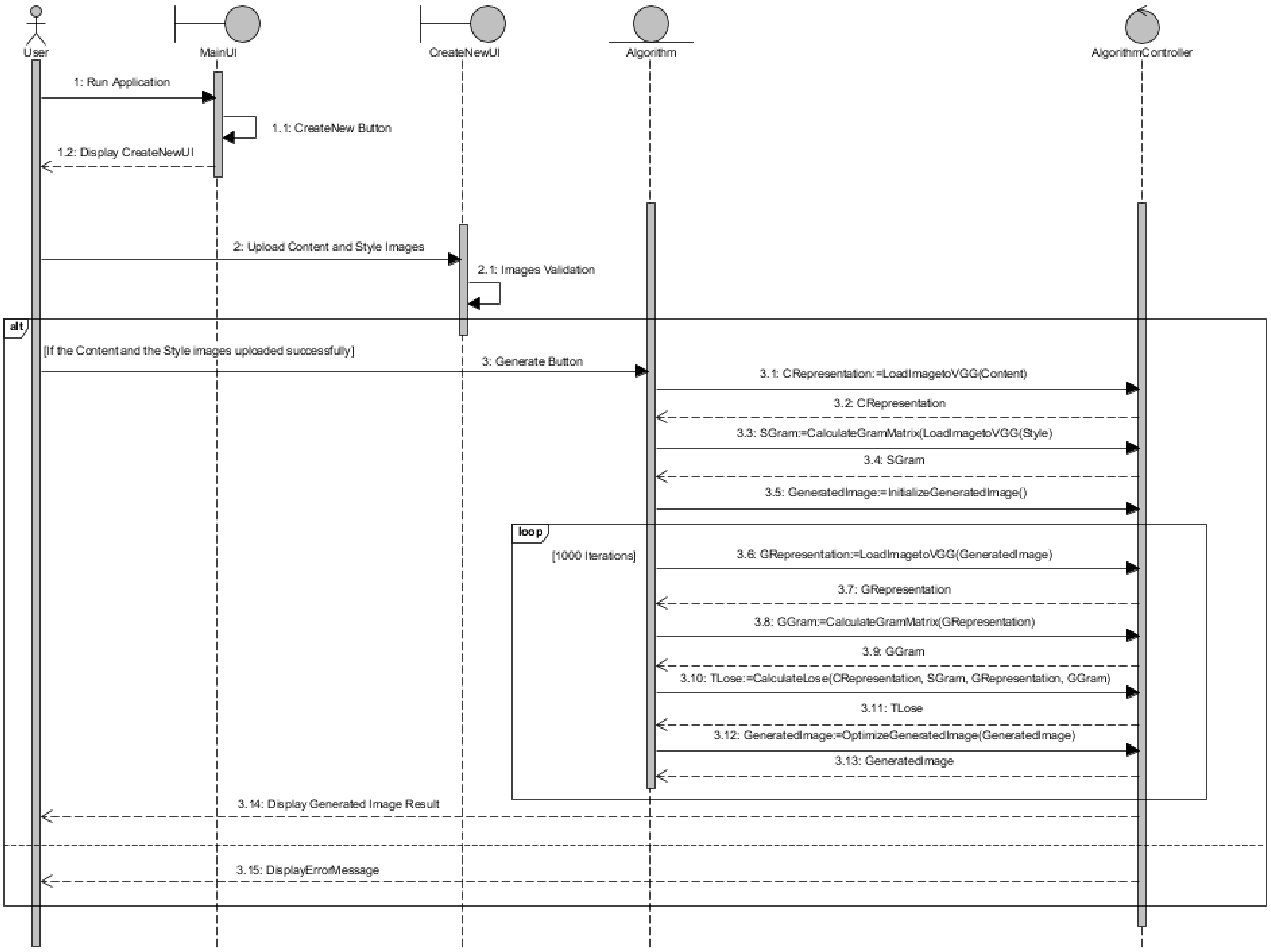
4.1 Requirements (Use Case)



4.2 Design (Class Diagram)



4.3 Sequence Diagram


User
MainUI
CreateNewUI
Algorithm
AlgorithmController
1: Run Application
1.1: CreateNew Button
1.2: Display CreateNewUI
2: Upload Content and Style Images
2.1: Images Validation
alt
[If the Content and the Style images uploaded successfully]
3: Generate Button
3.1: CRepresentation:=LoadImagetoVGG(Content)
3.2: CRepresentation
3.3: SGram:=CalculateGramMatrix(LoadImagetoVGG(Style)
3.4: SGram
3.5: GeneratedImage:= InitializeGeneratedImage()
loop
[1000 Iterations]
3.6: GRepresentation:=LoadImagetoVGG(GeneratedImage)
3.7: GRepresentation
3.8: GGram:=CalculateGramMatrix(GRepresentation)
3.9: GGram
3.10: TLose:=CalculateLose(CRepresentation, SGram, GRepresentation, GGram)
3.11: TLose
3.12: GeneratedImage:=OptimizeGeneratedImage(GeneratedImage)
3.13: GeneratedImage
3.14: Display Generated Image Result
3.15: DisplayErrorMessage

4.4 GUI Designing

4.4.1 Home Screen

The Main Screen of the application with menu and our logo.

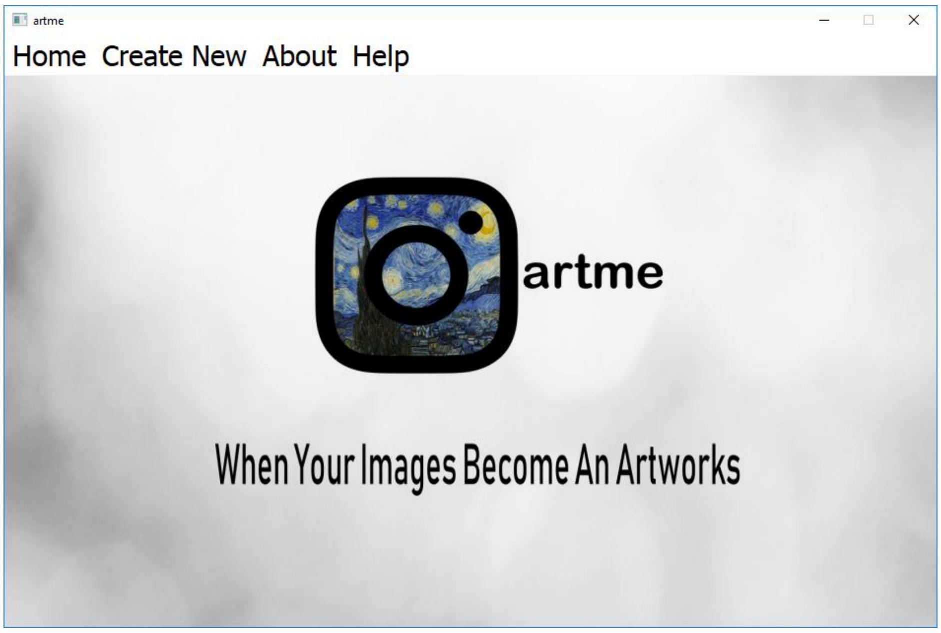

artme
Home Create New About Help
artme
When Your Images Become An Artworks

Fig. 11. Home Screen.

4.4.2 Create New Image Screen

By clicking on the Create New button the user will pass to Create New section.

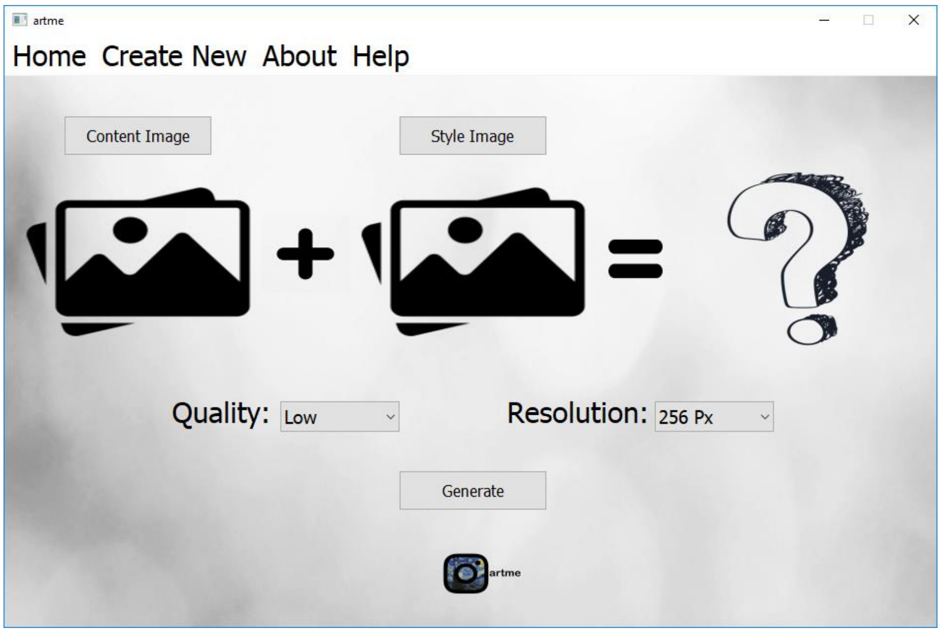


Fig. 12. Create New Screen.

The user should choose content and style images from desktop.

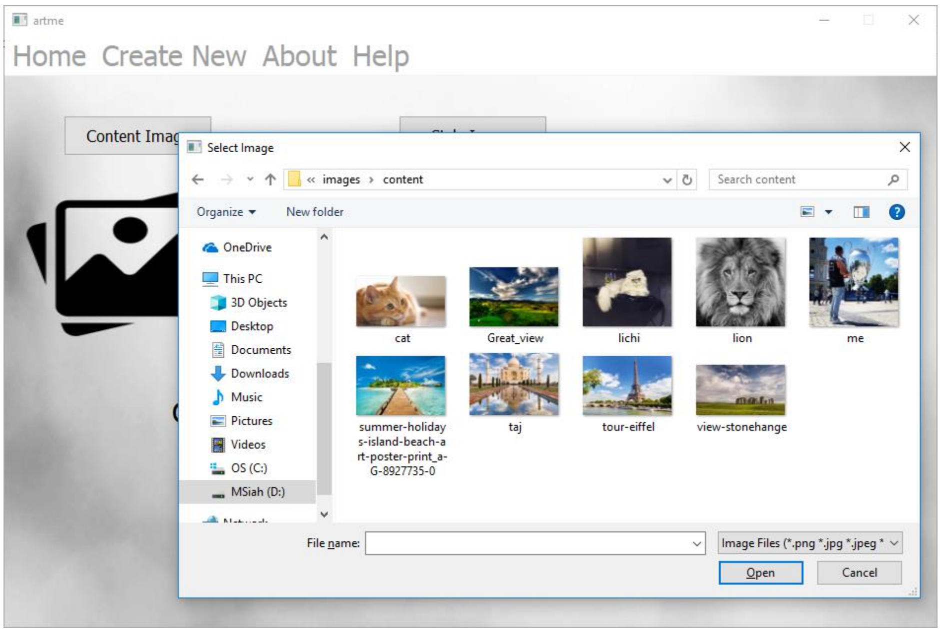


Fig. 13. Choosing Content or Style Image.

If the user pressed Generate button without choosing content and style images, a warning message will appear on the screen.

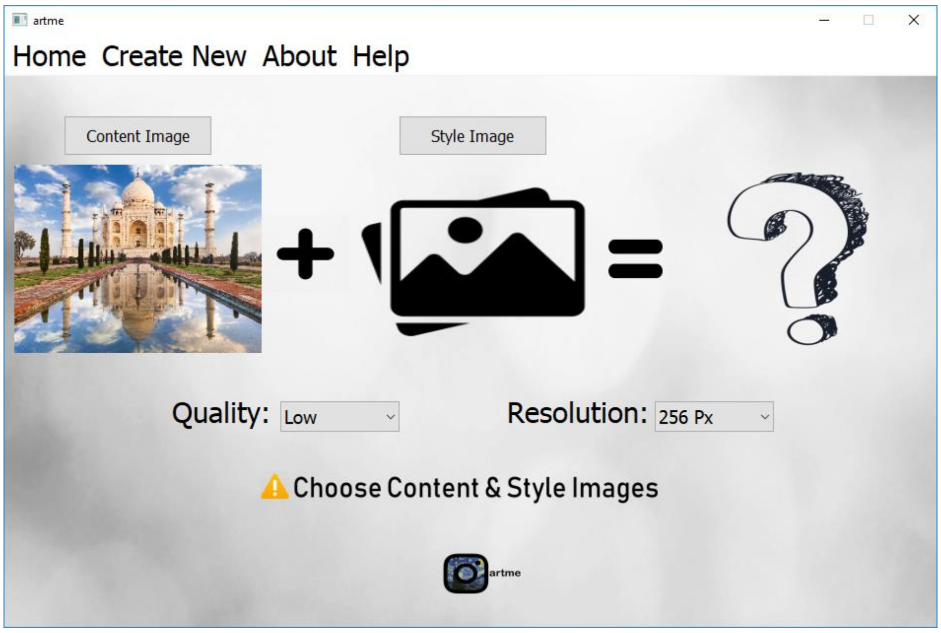
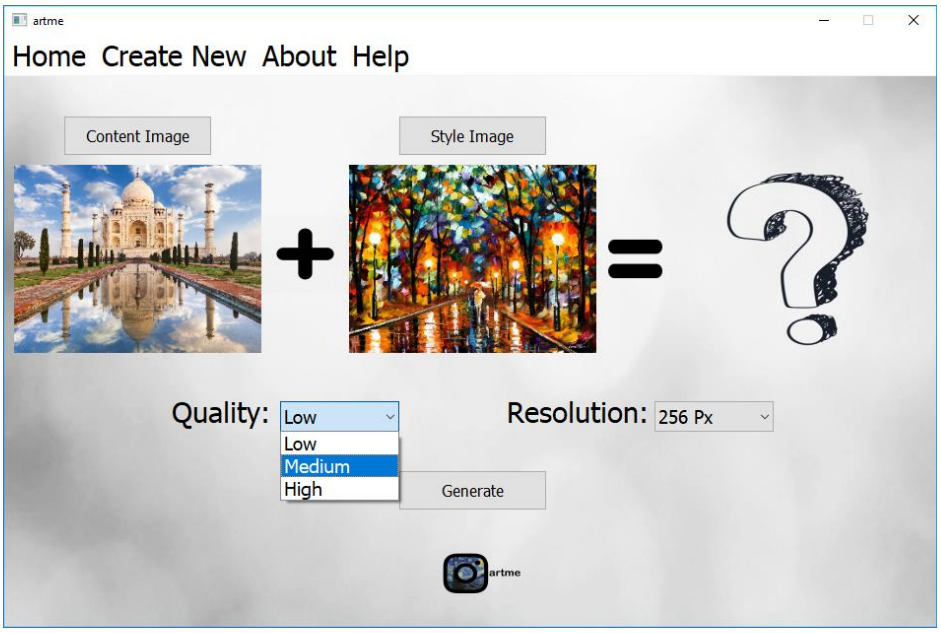


Fig. 14. Pressing Generate without choosing content and style Images.

After choosing content and style images, the user can choose the quality and resolution of the output image.

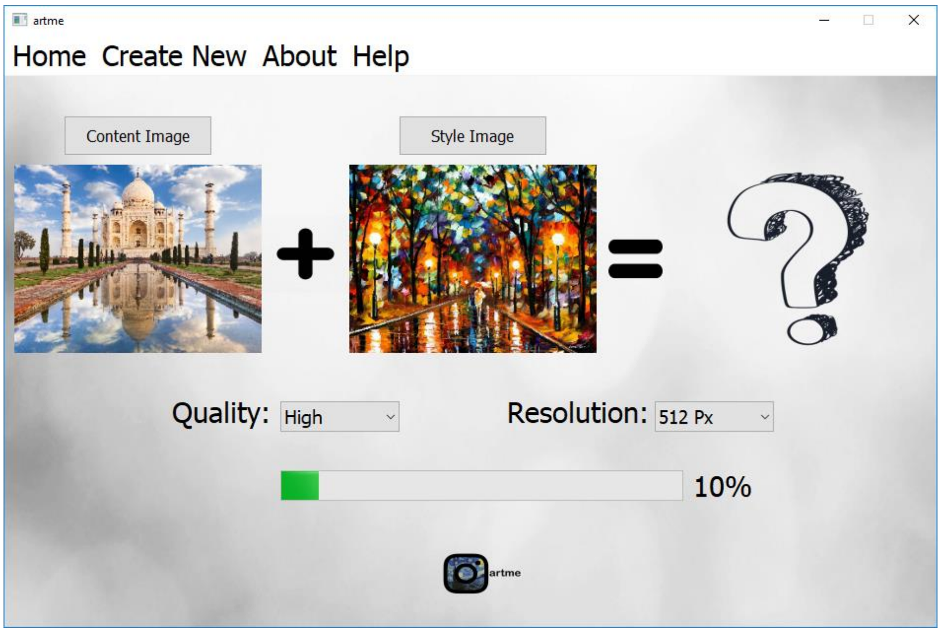


Quality: Low

Resolution: 256 Px

Fig. 15. Choosing Output Image Quality and Resolution.

After choosing all parameters and pressing Generate button the process starts.



Quality: High

Resolution: 512 Px

10%

Fig. 16. Process is on 10% after pressing Generate.

When the process is finished, the output image appears on the screen with a Save button.

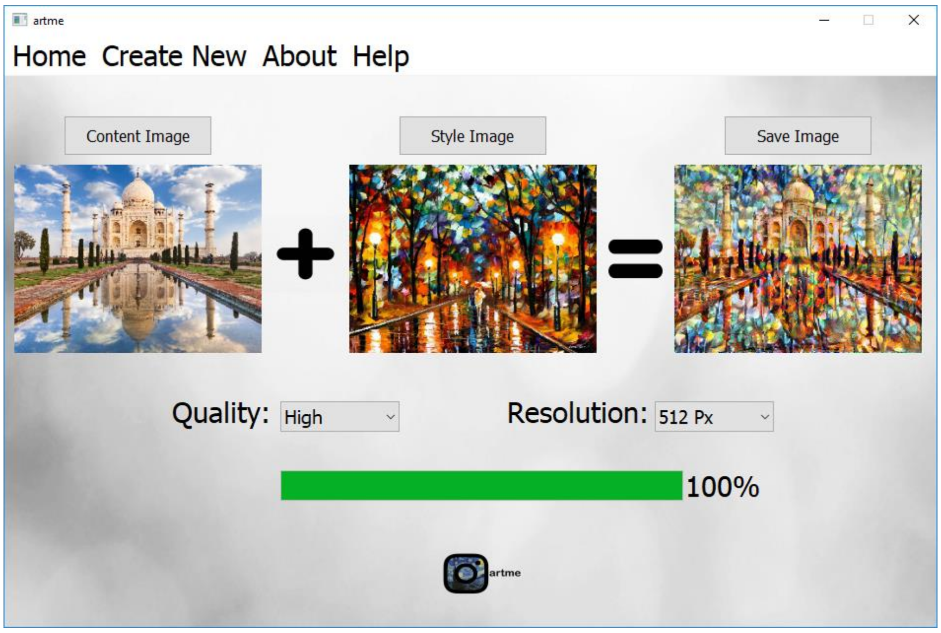


Fig.17. Process is Finished.

The user can press Save button to save the output image on the computer.

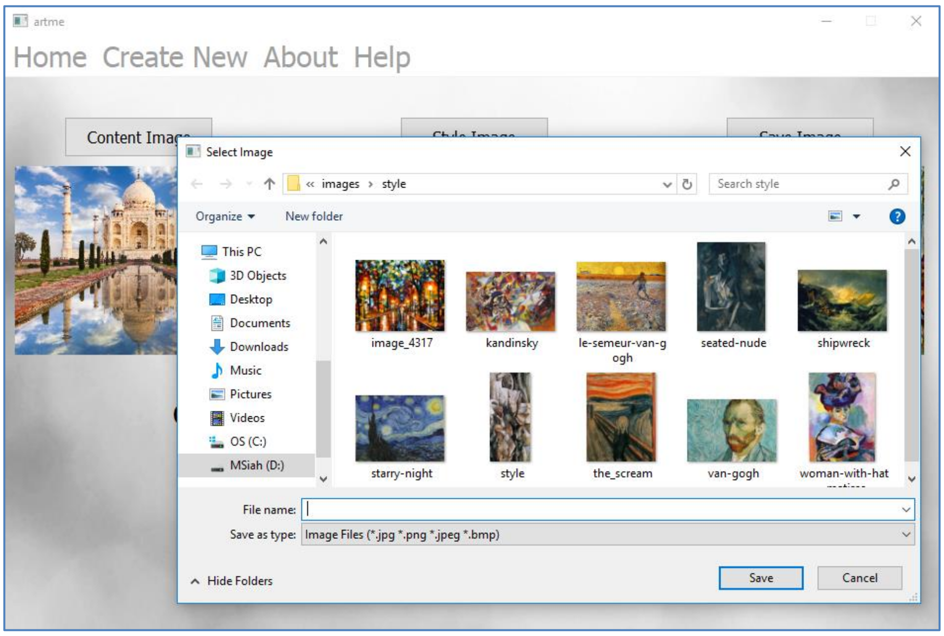
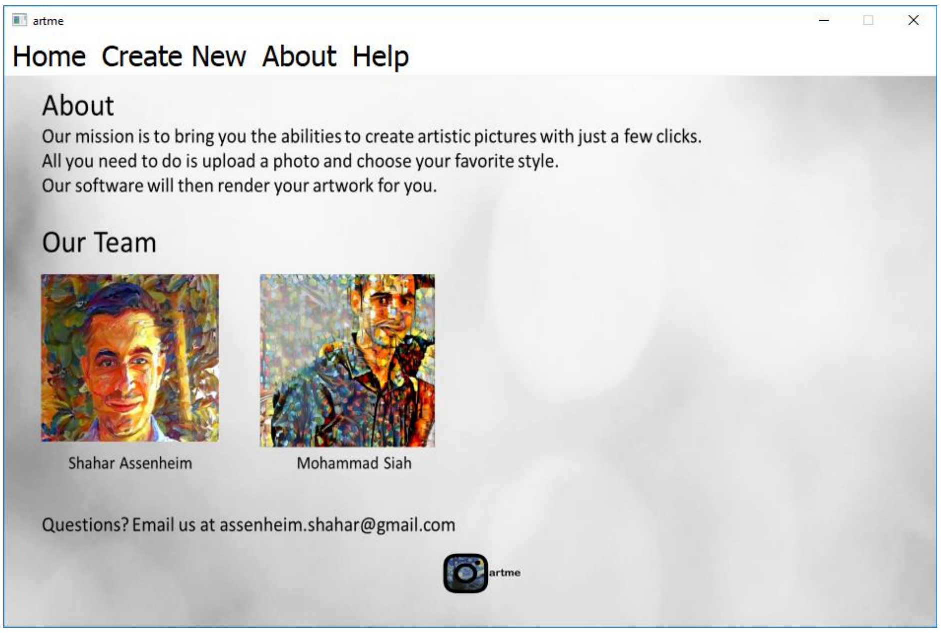


Fig. 18. Saving the Output Image.

4.4.3 About Screen

The about screen included explanation about the application and the developer's visions.



About

Our mission is to bring you the abilities to create artistic pictures with just a few clicks.
All you need to do is upload a photo and choose your favorite style.
Our software will then render your artwork for you.

Our Team

Shahar Assenheim

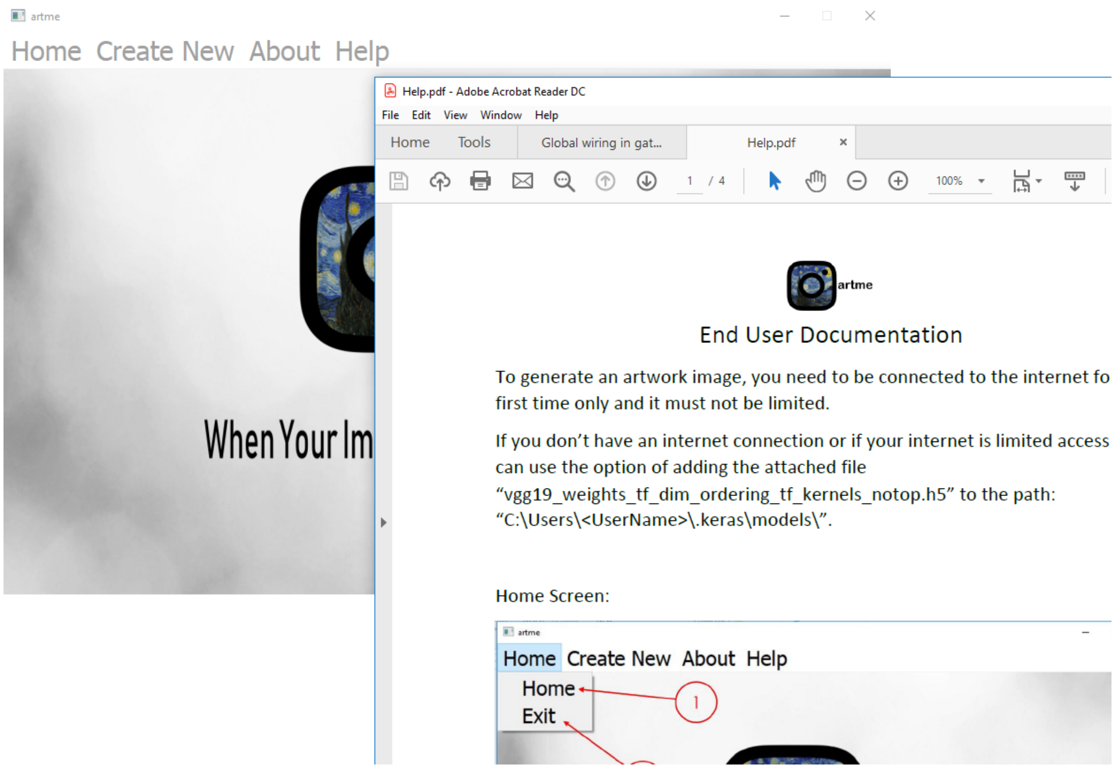
Mohammad Siah

Questions? Email us at assenheim.shahar@gmail.com

Fig. 19. About Screen.

4.4.4 Help File

The help button will open an End User Documentation file to explain how to use the application.


artme
Home Create New About Help
Help.pdf - Adobe Acrobat Reader DC
File Edit View Window Help
Home Tools Global wiring in gat... Help.pdf
1 / 4 100%
When Your Im
artme

End User Documentation

To generate an artwork image, you need to be connected to the internet fo
first time only and it must not be limited.

If you don't have an internet connection or if your internet is limited access
can use the option of adding the attached file
"vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5" to the path:
"C:\Users\<UserName>\.keras\models\".

Home Screen:

artme
Home Create New About Help
Home
Exit
1

Fig. 20. Help File.

4.5 Testing Plan

Test ID	Description	Expected results	Comments
NotUploadingContentImage	Clicking the button that generates the artwork image, without uploading a content image.	Display message: "You Must Upload a Content Image".	Make sure the user uploads the content image you want to change to artwork.
NotUploadingStyleImage	Clicking the button that generates the artwork image, without uploading a style image.	Display message: "You Must Upload a Style Image".	Make sure the user uploads the style image to get its style to make the artwork.
NotUploadingAnyImage	Clicking the button that generates the artwork image, without uploading any image.	Display message: "You Must Upload a Content and a Style Images".	Make sure the user uploads two images to make an artwork
SendingTwoGoodPictures	Clicking the button that generates the artwork image, after uploading two good images.	Getting appropriate result: "The paper is REAL, it was written by human".	Make sure the algorithm returns a suitable result.
ChoosingQuality	Choosing a “Low” quality for the output image	Getting a low-quality artwork with only 100 iterations	We define the low quality as 100 iterations, it should be a bad result
ChoosingResolution	Choosing a “512 Px” resolution for the output image	Getting a ~512x512 artwork resolution	The artwork we get will be at the most 512 pixels **not necessarily square image**

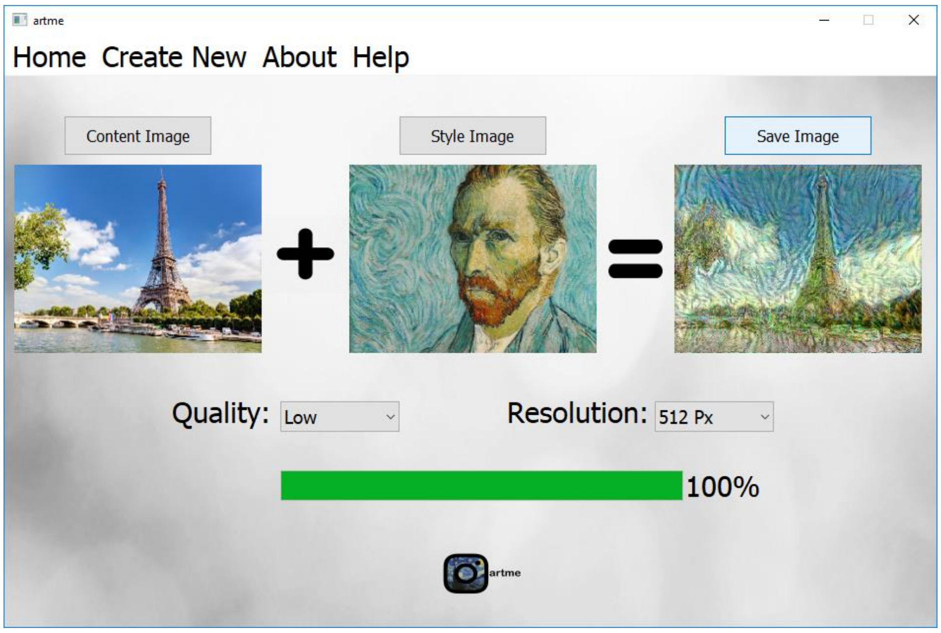
5 Results and Conclusions

5.1 Results

In this section we will present the results of the experiments we have done in order to test our application

Experiment 1

In this experiment we chose the Low-quality level which means the algorithm runs for 100 iterations and choosing resolution as 512 Pixels, we can see that the output image quality is not high, and that the colors don't transferred so good.



Quality: Low

Resolution: 512 Px

100%

Fig. 21.

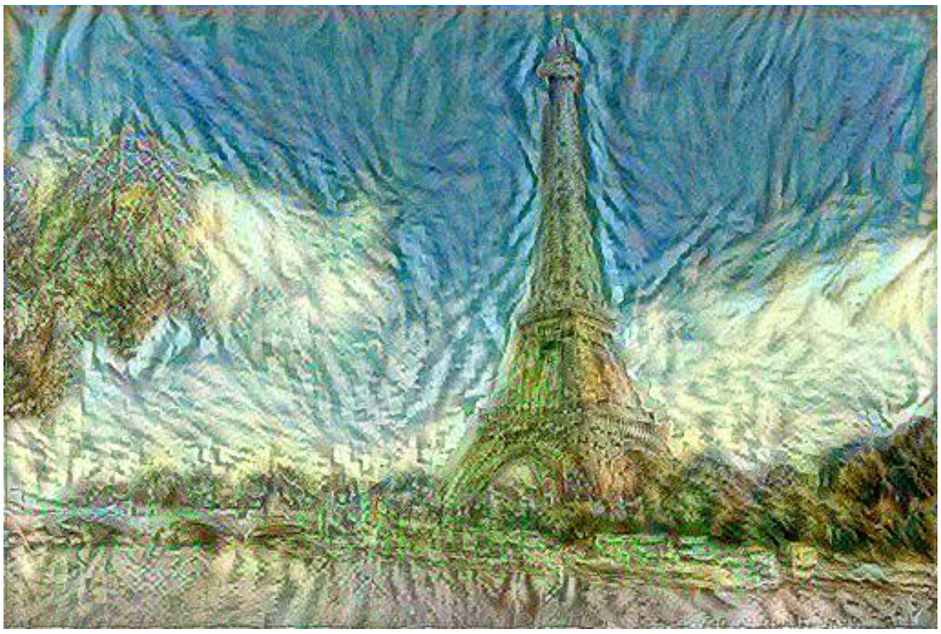
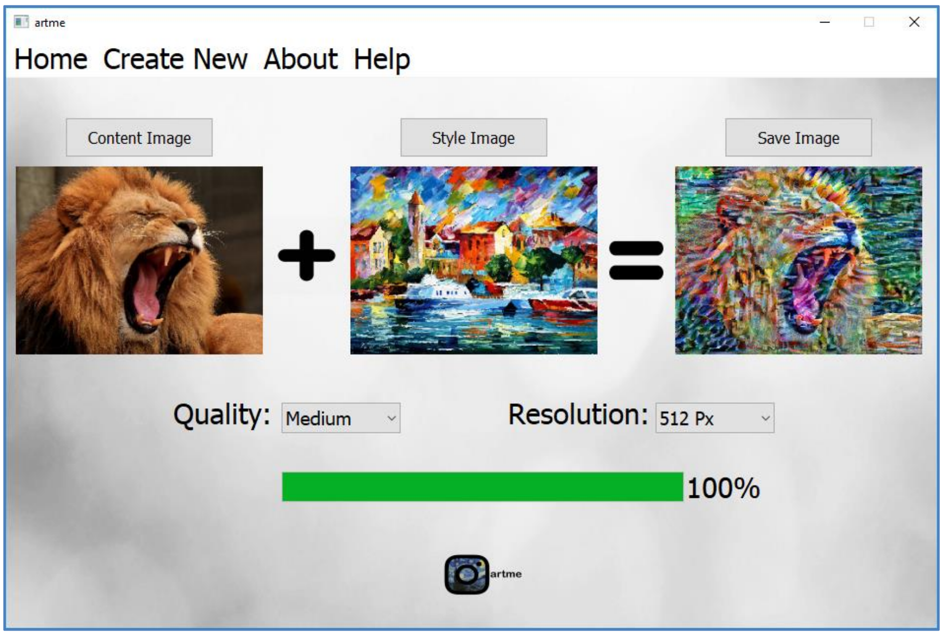


Fig. 22. Output Result (Low-quality and 512 Pixels resolution).

Experiment 2

In this experiment we chose the Medium-quality level which means the algorithm runs for 500 iterations and choosing resolution as 512 Pixels, we can see that the output image quality is better comparing to the last experiment, but still the colors don't transferred satisfactorily.



Quality: Medium

Resolution: 512 Px

100%

Fig. 23.

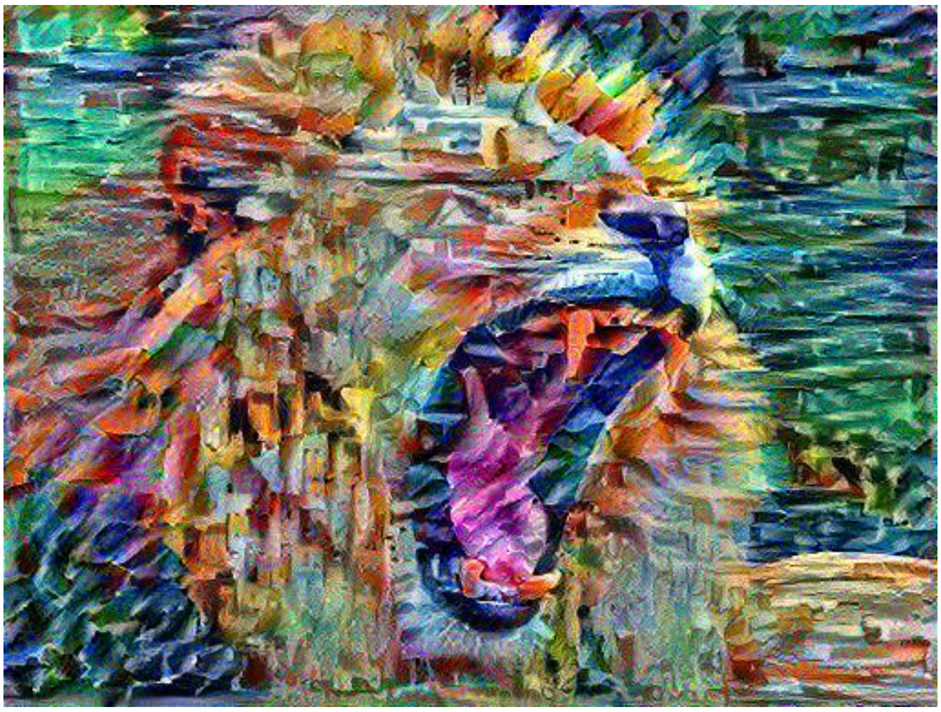
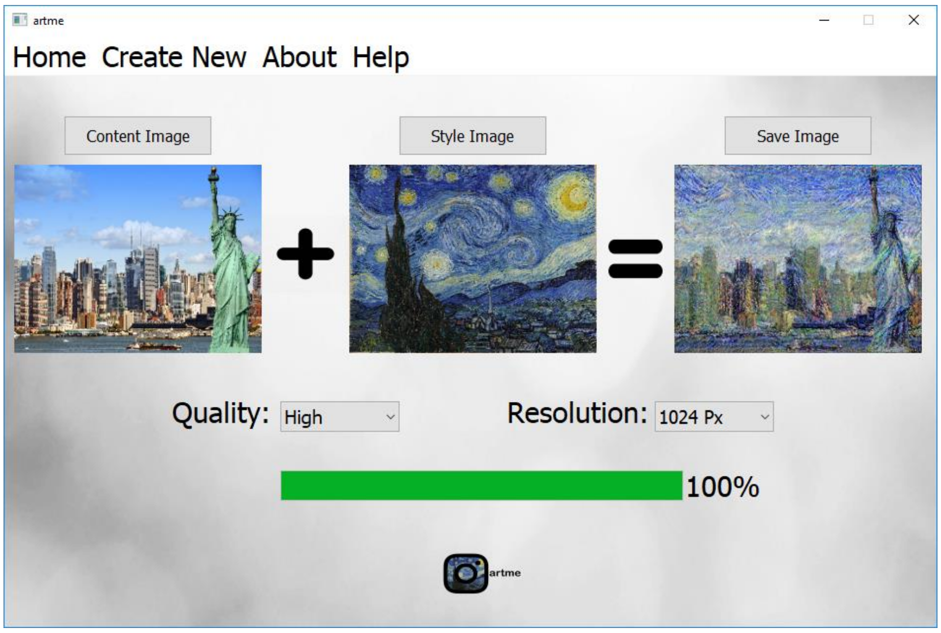


Fig. 24. Output Result (Medium-quality and 512 Pixels resolution).

Experiment 3

In this experiment we chose the High-quality level which means the algorithm runs for 1000 iterations and choosing resolution as 1024 Pixels to get a better quality, we can see that in this experiment the output image is with high color quality, and comparing to the lasts experiments this is the best result.



Quality: High

Resolution: 1024 Px

100%

Fig. 25.

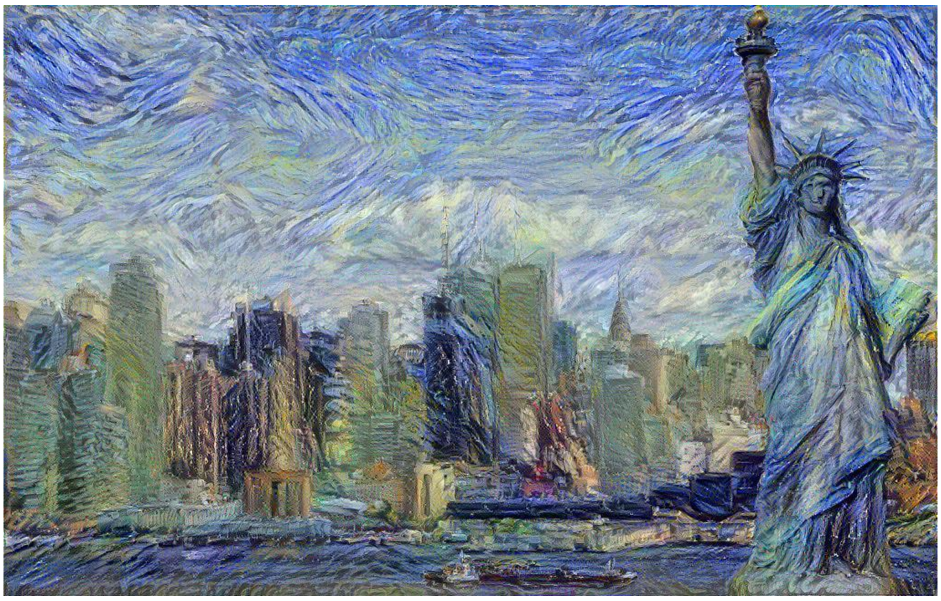
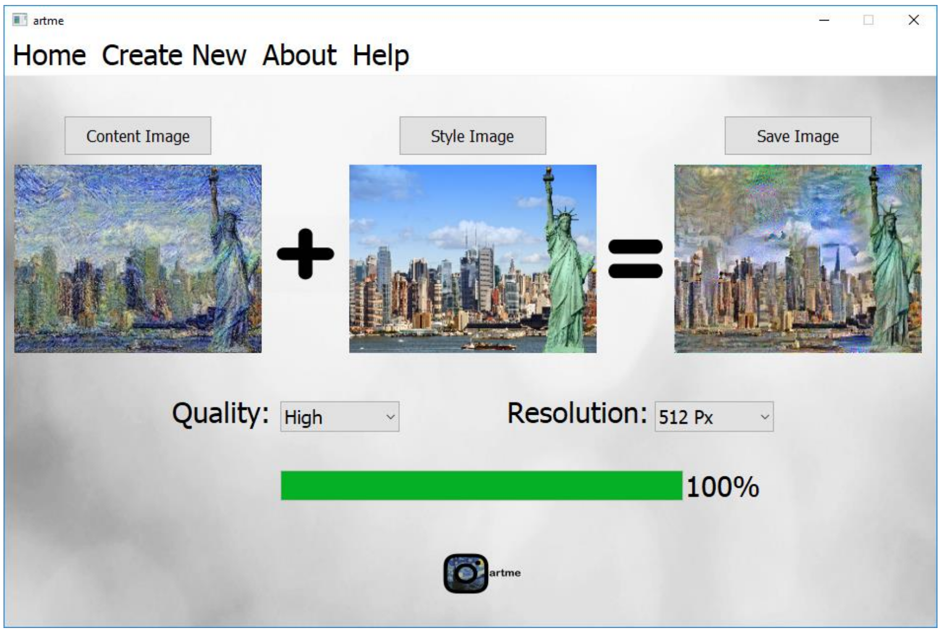


Fig. 26. Output Result (High-quality and 1024 Pixels resolution).

Experiment 4

In this experiment we chose the High-quality level which means the algorithm runs for 1000 iterations and choosing resolution as 512 Pixels to get a better quality, but in this experiment, we chose the content image to be the output of the 3rd experiment and the style image to be the original content image we used in the 3rd experiment.



Quality: High

Resolution: 512 Px

100%

Fig. 27.

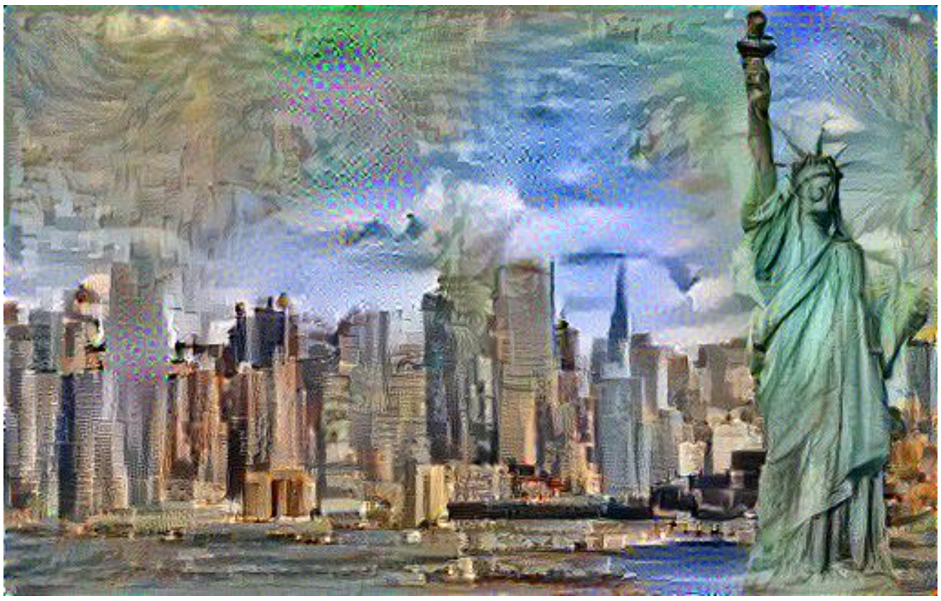
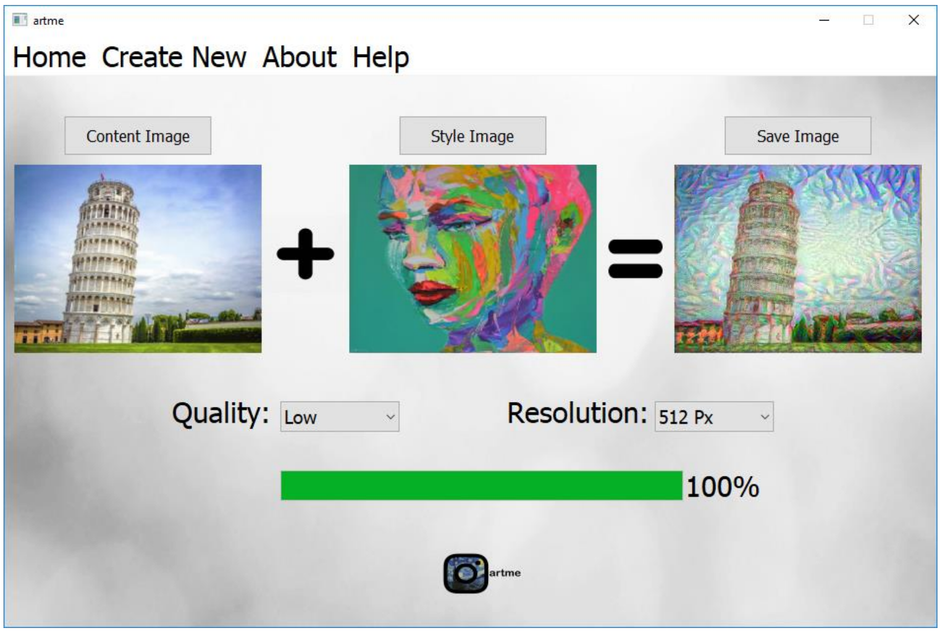


Fig. 28. Output Result (High-quality and 512 Pixels resolution).

<u>Experiment 5</u>

In this experiment we chose a 360x360 content image, 458x458 style image with 512 Px resolution, we can see that the output image is always resized to the resolution we choose, in this case 512 pixels.



Quality: Low

Resolution: 512 Px

100%

Fig. 29.

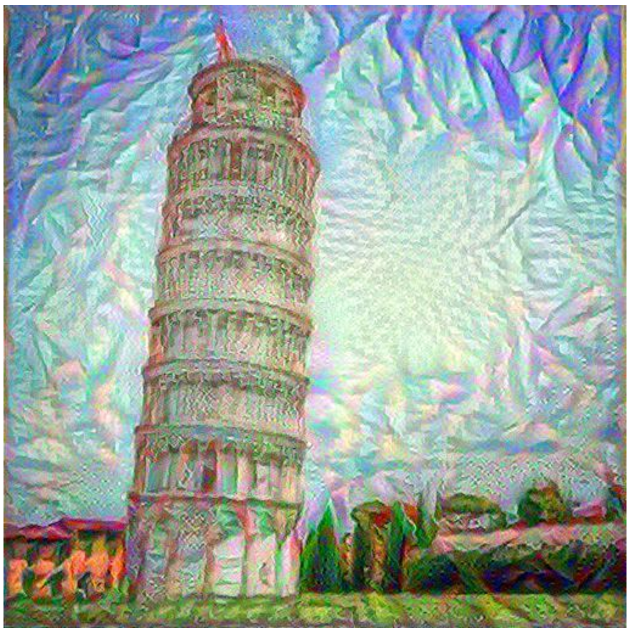


Fig. 30. Output Result (512x512 artwork).

5.2 Conclusion

At the experiment stage we have made executions on different content and style images and different algorithm's parameters, trying to understand better how the algorithm responds to the various parameters' values. We learned that for different style and content images there is no effect on the time for the algorithm to finish the process, however for the resolution and the quality parameters of the output images there is a big effect on the process time.

We found out that for the highest quality of style transfer we must run the algorithm at least 1000 iterations, in addition for a higher resolution the algorithm requires more time to finish.

In experiment 4 we tried to do manipulation and taking for the content image the output result of experiment 3 and for the style image the original content image of experiment 3, we can see in the result that the output image doesn't look so good.

In experiment 5 we took 2 small sized images for the content and the style and as we can see that the application resizes the output image to the resolution we choose.

When we started our experiments, we quickly found out that our laptops are not strong enough to finish the process in a short time, and for this kind of application the processing time should not take more than few minutes, for that reason our main conclusion is that our application must run on a strong server computer.

On the other hand, the output images that our application generated were exactly as we expected to get at the beginning, some of them even look like they were painted by the artist himself, for example Fig. 26.

The name we chose to our application is "artme", it's come from the reason that with this application we can transfer any image of us to an magnificent artwork.

6 REFERENCES

[1] L. A. Gatys, A. S. Ecker and M. Bethge. Image Style Transfer Using Convolutional Neural Networks.

[2] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer.

[3] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies.

[4] N. Ashikhmin. Fast texture transfer.

[5] Michael Elad and Peyman Milanfar. Style Transfer Via Texture Synthesis.

[6] Elias Wang, Nicholas Tan. Artistic Style Transfer.