Image Style Transfer using Text Descriptions

Ionuț Gabriel Boboc, Mihai Dascalu, Ștefan Trăușan-Matu
University Politehnica of Bucharest
313 Splaiul Independenței, Bucharest, Romania
boboc.ionut.gabriel@gmail.com, {mihai.dascalu, stefan.trausan}@cs.pub.ro

ABSTRACT
Artistic images are becoming trendier and recent researches in computer vision have showcased specific use cases of style transfer in which two images are combined together. The aim of this paper is to introduce a new method for transferring style from one image to another based on deep neural networks. The main idea is to combine a specific image with user's requirements expressed as input text. Our method relies on the ReaderBench framework to extract keywords from the given text, which are afterwards used to search for a representative image and to change accordingly the input image. We developed a desktop application which starts from an image and a text description as input data, and generates a new image with the same landscape and objects, but taking into account the requirements from the input text, for example shift to a night landscape or an artistic change.

Author Keywords
style transfer, deep neural network, ReaderBench framework, image processing.

ACM Classification Keywords
H.5.2. Information interfaces and presentation (e.g., HCI): User Interfaces.

I.2.7 Natural Language Processing: Discourse, Language parsing and understanding, Text analysis

General Terms
Natural Language; Text analysis; Image processing.

INTRODUCTION

Images are an integrated part of our lives as all observed information is visual. Moreover, images provide contextualization and a means for people to express themselves. A specific showcase is centered on social networks in which pictures are frequently used alongside with words which provide corresponding descriptions. In addition, artistic images are becoming trendier and deep neural networks have been used to transfer style between two images in order to combine them [1].

Our aim is to transfer style from one image to another using text descriptions. Our input consists of an image and a text description with a desired change, while the model tries to accommodate the modifications in a coherent manner. We use a deep neural network that relies on models designed to transfer image styles from one picture to another. Various changes were performed in order to reduce runtime, while still obtaining qualitative results. We applied a keywords extraction mechanism on the input description using the ReaderBench framework [2, 3].

The paper continues by describing state-of-the-art approaches, our architecture, followed by tests and experiments performed in order to improve resource usage, and the quality of the end results, together with running time optimizations. The paper ends with conclusions and ideas for future work.

STATE OF THE ART

The idea of automated style transfer techniques was recently implemented in a functional manner [1] due to the many encountered difficulties in designing an algorithm that can actually understand the image as one concept, and not a collection of values. Another required process consists of keywords extraction, which is a Natural Language Processing (NLP) technique supported by the ReaderBench framework [3].

Style transfer

There are many technical issues which are difficult to resolve using classic approaches of style transfer that do not rely on deep neural networks. The most frequently encountered issues are:

- Generating automatic strokes with a certain size using pixel analysis;
- Selectively applying strokes only to certain areas;
- Understanding what elements need to be kept and what to replace.

One of the solutions [1] considers style transfer as an optimization problem which refers to the existence of a function *F* that takes as input two images and has as output a new image for which the style is as similar as possible to the first one, while its content is as similar as possible to the second one (see Figure 1).

Figure 1. Generic optimization function [1].

Thus, we can derive the following optimization problem:

$$R = \min_{r}[L_c(c,r) + L_s(s,r)]$$

where R is the image result, r is an image candidate, c is the content image, s is the style image, L_c is the a function that takes two images and computes the content differences between them (it will tend to zero when those two are very similar), L_s is the a function that takes two images and computes the style differences between them (it will tend to zero when those two images are very similar).

Furthermore, we need to argue why classic style transformations at pixel level are not applicable, even if these types of image transformations have been iteratively improved:

- There is no clear way to define what is more important from an image (*style or content*), and how much information needs to be kept from each element.
- *Object variations* – there are multiple visual representation for the same object category (ex., we cannot define how a car should look like).
- *Light variation* can cause an object to be only partially visible, whereas shadows can change entirely how an object looks like.
- Aside from light or object variations, the *angle* from which the object(s) is (are) observed can change entirely its representation.
- Homogeneity – presuming we can separate style from content, the next challenge is to make the image look homogenous.

To overcome many of these problems, Gatys, Ecker & Bethge [1] use semantic differences to compare images based on a pretrained network for image recognition which is capable to recognize features, instead of using pixels.

The above approach has one issue – it can work for any pair of images (style, content), but it can take a long period of time to complete. One solution is to use a neural network which approximates the optimization problem [4]. After this secondary network is trained on a specific style image, it can provide results in seconds, but is limited to that particular image.

ReaderBench

ReaderBench is an automated software framework designed to support students and tutors by making use of advanced NLP and text mining techniques. It has an extraction component that determines the most relevant keywords from a document by using an extended bigram approach based on Cohesion Network Analysis [5]. In order to further improve the results, an enhanced selection of keywords based on their commonness score (a value that is generated from a text corpus that disregards common terms) was performed. This is due to the fact that words with the highest number of occurrences are seldom mapped to be among the most relevant keywords. Therefore, the extraction is based on the Average Logarithmic Distance because it showed the greatest stability, as presented in the study performed by Savický & Hlavácová [6].

METHOD

While different changes were made to the implemented solution for style transfer, we combine the method introduced by Gatys, Ecker & Bethge [1], with an optimization from Berger & Memisevic [7] which is further explained.

Integrated resources

For this work, we opted to allow any pair of images (style & content) and integrate a pretrained network for image recognition on the ImageNet database - the chosen network was VGGNet16, which relies on Keras. We also used a classifier from OpenCV library – Cascade Classifier (https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html) – for detecting persons in an image. The model was already trained for frontal face detection and required additional tuning. We chose Cascade Classifier because it is reliable and simple enough in terms of architecture, which makes it very fast. The model groups features in stages and, if the image doesn't pass a stage, it is dropped; in the end, an image has to pass all layers in order to be correctly classified.

Style transfer

The solution for style transfer transformation is to build the system as an optimization problem in three steps for computing: a) content loss, b) style loss and c) the total variation loss:

- G^l is a $N^l x N^l$ matrix corresponding to the Gram matrix for the original image;
- T^l is a $N^l x N^l$ matrix corresponding to the Gram matrix for the generated image;
- $F^l_{j:}$ is the i-th vectorized feature map of the layer l (from the generated image);
- $P^l_{j:}$ is the i-th vectorized feature map of the layer l (from the original image);
- M^l is the number of elements in each map of the layer l;
- w_c is the weight applied for normalizing content;
- w_s is the weight applied for normalizing style;
- $<>$ is the inner product.

The standard VGGNet [8] uses a 19-layer model for object classification and any convolution layer from the first ones can be used to define content loss. The content can be defined as Euclidean distance between the chosen layer result and combination images ant the following formula can be applied for content loss:

$$L_{content} = w_c \frac{1}{2} \sum_{i,j} (F^l_{ij} - P^l_{ij})^2$$

However, style loss must be computed independent of context. Thus, a Gram matrix is used. The Gramm matrix is a Hermitian matrix of inner products whose entries are given by $G_{ij} =< v_i, v_j >$, where v_i is one of the set of vectors in an inner product space. Figure 2 introduces the style loss used within our deep neural network.

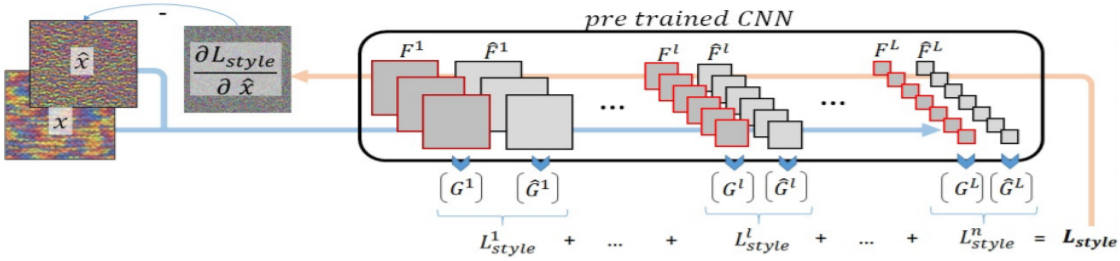

Figure 2. Style loss [9].

The Gramm matrix values are computed using the following formula [9]:

$$G^l_{ij} = \frac{1}{M^l} \sum_{k=1}^{M^l} F^l_{ik} F^l_{jk} = \frac{1}{M^l} < F^l_{j:}, F^l_{j:} >$$

In order to better retain the geometrical position of objects in the image, co-occurrences can be computed between feature maps F^l and spatially transformed feature maps $T(F^l)$, where T denotes a spatial transformation instead of computing co-occurrences between multiple features within a map. Nevertheless, this solution can be computationally expensive and style loss is computed using the following formula [9]:

$$L_{style} = \sum_{l=1}^{L} w_s ||T^l G^l||^2_F = \sum_{l=1}^{L} w_l L^l_{style}$$

Derived style transfer implementation

The first difference with the 19-layer VGGNET [8] is the use of 16-layer model because our experiments showed no visible improvements when relying on the larger model versus the smaller model (see Figure 3 in which the maxpool layers were removed).

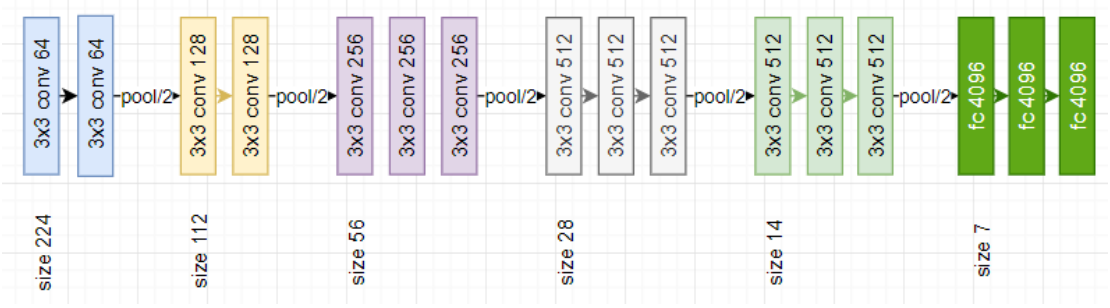

Figure 3. Simplified VGGNET with 16 layers.

We addressed the issue of speed by using GPU acceleration, but another issue that is still present due to resolution limits (i.e., resolution is fixed at 512x512 pixels for this work). Even if for defining the content loss any convolution layer from the first ones can be used, experiments have shown that using the second block with the second convolution layer has better results. When the image is larger than 512x512 pixels, the image is automatically scaled such that either the width or height are scaled to 512, the window of 512x512 is centered, and the original image is cropped to this window. Using the explained methods with corresponding changes, the next step is to improve our results. A gradient descent can be used or an optimized L-BFGS to iteratively improve the solution. In our case, we opted to use L-BFGS because it provides results faster. Furthermore, we chose not to use the feature from any of the layers defined in [1] or [4], and we defined our custom layer for content loss, as follows:

$$content\_layer = \text{3x3 conv 512}$$

The layers for style loss are the same as in [4], where 2 represents the second layer from the block:

$$style\_layers = \begin{matrix} 3x3conv64(2) & 3x3conv128(2) \\ 3x3conv256(2) & 3x3conv512(2size28) \\ 3x3conv512(2size14) \end{matrix}$$

However, the resulting output had a high noise level and, in order to address this, we used an implementation of the total variation loss – a total variation regularization for 2D signals [10] (images in our case) with the following formula. In order to tweak further the results, we choose β (regularization parameter) to be equal to 2.5 as its increase forces the image to have smaller variations:

$$V(y) = \sum_{i,j} (|y_{i+1,j} - y_{i,j}|^2 + y_{i+1,j} - {y_{i,j}}^2)^{\frac{5}{4}}$$

Architecture

The next step consists of combining style transfer with the keywords obtained using ReaderBench. A basic application relying on Windows Presentation Foundation (WPF; https://visualstudio.microsoft.com/vs/features/wpf/) [11] was developed, capable of loading an image, get the user input, work with images (resize, obtain bitmaps, save, load, manage resources), make HTTP requests, parse JSON responses, and manage and synchronize the different processes (face detection, style transfer). Figure 4 depicts the architecture of our solution.

The application uses interactional and independent windows, while its flow is controlled in the central class that manages and displays images, as well as makes HTTP requests to ReaderBench. Style transfer is based on TensorFlow, in a Python environment, on top of which a pretrained VGG network from Keras is used for object detection.

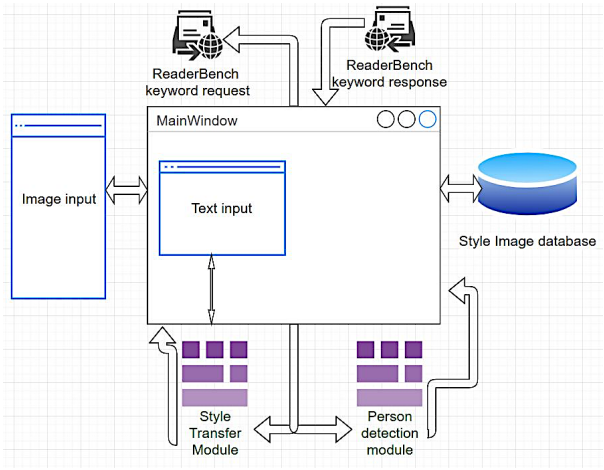

Figure 4. Architecture.

At the core of the application is the main window component which is responsible for loading and displaying the target image, as well as combining all the components of the system. The image displayed inside the main window is selected using another window which displays a preview of images. All images, both within the main window or within the layers window, are represented as an abstract object with required functionalities. The main window also makes calls to the utility component which triggers an input window for the user to write text which is sent to ReaderBench for further processing and used as input for selecting a representative image from the local database.

After the image is obtained, it is used as input for the Python module responsible for style transfer. The neural style transfer component is further tuned by considering the detection of human faces inside the content image.

Workflow

Figure 5 shows a simple use-case scenario. The user provides the content image which is loaded using the image handler from the main window. In the next step, the user opens the style transfer input window and starts the transformation process. Next, users are prompted to provide the textual change that they want to apply. Inside the main window, a request for the text keywords is made for ReaderBench. Using the keywords, the main module selects one of the image styles (this is chosen as the best match between keywords and labels). The content and style image, alongside with other chosen parameters based on the content image information, are sent to the Python module which generates the output image.

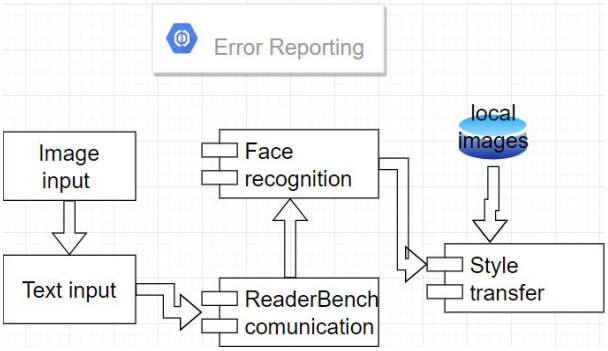

Figure 5. Application workflow.

RESULTS

Image results

Starting with the image from Figure 6.a and the text *"at night it will look even more impressive*", the model generates the output from Figure 6.b. In contrast, the same input image combined with the text "*it must be a painting*" generates Figure 6.c. Similarly, the portrait in Figure 7.a was subject to night (Figure 7.b) and sketch (Figure 7.c) transformations.

After multiples runs, we discovered that in certain scenarios the style transfer process must be adapted to the content from the image, especially when the content is a face. The default settings that apply artistic changes can make an image lose its content and more fine details, when containing at least one face (see Figure 8 in which the test was made without face detection). In Figure 9 we present sample results when the option of face detection is activated.

(a) (b) (c)

Figure 6. a) Initial normal image. b) Night transformation. c) Artistic transformation.

(a) (b) (c)

Figure 7. a) Initial normal image. b) Night transformation. c) Sketch transformation.

Figure 8. Output that lost content details.

Figure 9. Output which retained more content details.

Figure 10. Partially transformed image.

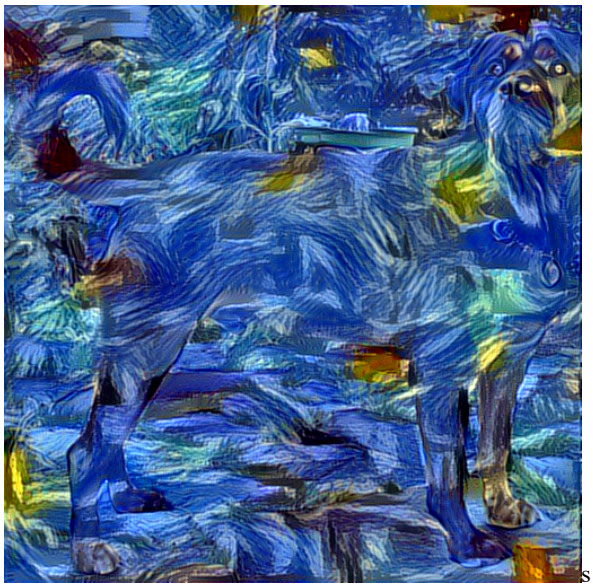

Figure 11. Sample destroyed image.

However, there are situations in which the style transfer parameters are not correctly set for the image, or the extraction technique can't resolve the description (see Figures 10 and 11). In Figure 10, the water and most of the buildings are correctly transformed, whereas the center part of the image is unchanged because it is perceived as a common component. The changes in Figure 11 are more drastic and the content of the input image is almost completely lost.

Performance

In terms of performance, the running time for a change is approximately 2 minutes from transformation submission on the following specifications: a laptop with an i5 Q7300 processor, NVidia gtx1060 max-q and 8GB of RAM. The load of each resource, the CPU/GPU loads reach their peaks when style transfer is performed (see Figure 12). Figure 13 depicts the memory load of the application which normally uses around 100MB of RAM except when the style transfer is done, and it surmounts to almost 2GB of RAM.

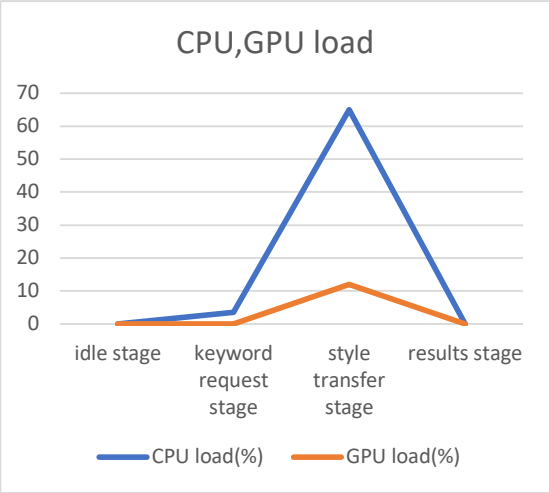

Figure 12. CPU/GPU load across transformations.

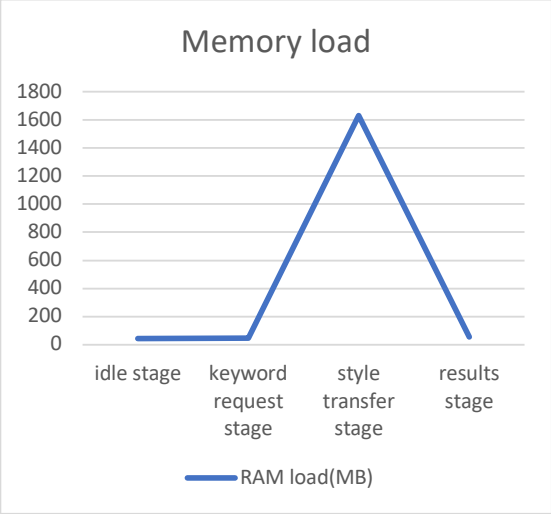

Figure 13. Memory requirements

Survey

To further understand user perception, we performed a small survey with a set of 5 predefined inputs (images + text) and two custom inputs freely selected by the users. Our group consisted of 18 users, 11 females and 7 males, aged between 21 and 26 (mean = 22.72, standard deviation = 1.67) from different fields. Most of the pretested inputs were well received and users were excited about the outputs (see Table 1), all except one which was intentionally introduced as not a great result. Users found the application to be fun, easy to use, suggestive, artistic and quite fast, whereas only one user did not enjoy the generated water effect. Interestingly, targets had a better perception of the custom examples (higher Likert scores) and wanted to change the input text more for custom images in comparison with the predefined use cases. Almost half of the users wanted to change the custom images and the used text instead of the fixed examples, and most of them chose to use a portrait image with the sketch example. An example of changed text is from "I would like a sketch" to "I wonder how this will look as a cartoon".

Closed Question [1-5] Likert scale (1 disagree; 5 – agree)	**Predefined inputs M (SD)**	**Custom examples M (SD)**
Q1: Did the resulted image retain the basic details from your input image?	4.25 (0.95)	4.28 (0.78)
Q2: Is the resulted image a good reflection of the intent expressed in the text input?	3.82 (1.29)	4.08 (1.16)
Q3: On a scale of 1 to 5, how much artistic emotion was induced by the resulted image?	3.81 (1.02)	4.14 (0.91)
Q4: On a scale of 1 to 5, how much do you like the resulted image?	3.87 (1.10)	4.25 (0.81)
Q5: After viewing the results, do you feel the need to change the input text?	2.92 (1.39)	2.89 (1.33)

Table 1. Survey results (M – mean; SD – standard deviation).

CONCLUSIONS AND FUTURE WORK

Results in this work show promising leads towards automating the process of transforming images based on text suggestions. Speed improvements for the style transfer component proved that the system is capable of obtaining the output in a relatively small amount of time. One issue that remains to be addressed is the resolution limitation. Using a secondary network for identifying objects in the image also improved the end results, especially for person pictures in which the parameters have to be adapted accordingly.

ACKNOWLEDGMENT

This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS – UEFISCDI, project number PN-III 72PCCDI / 2018, ROBIN – "Roboții și Societatea: Sisteme Cognitive pentru Roboți Personali și Vehicule Autonome".

REFERENCES

1. Gatys, L.A., Ecker, A.S., and Bethge, M., (2015) A neural algorithm of artistic style. arXiv preprint arXiv:1508.06576.
2. Dascalu, M., Dessus, P., Bianco, M., Trausan-Matu, S., and Nardy, A., 2014. Mining texts, learner productions and strategies with ReaderBench. In Educational Data Mining: Applications and Trends, A. Peña-Ayala Ed. Springer, Cham, Switzerland, 345–377.
3. Gutu, G., Ruseti, S., Dascalu, M., and Trausan-Matu, S., 2017. Keyword Mining and Clustering based on Cohesion Network Analysis. Proceedings of the 3rd Workshop on Social Media and the Web of Linked Data (RUMOUR 2017), in conjunction with the Joint Conference on Digital Libraries (JCLD 2017) (Toronto, Canada), "Alexandru Ioan Cuza" University Publishing House, 23–30.
4. Johnson, J., Alahi, A., and Fei-Fei, L., 2016. Perceptual losses for real-time style transfer and super-resolution. Proceedings of the European Conference on Computer Vision, Springer, 694-711.
5. Dascalu, M., McNamara, D.S., Trausan-Matu, S., and Allen, L.K., (2018) Cohesion Network Analysis of CSCL Participation. Behavior Research Methods 50, (2), 604–619.
6. Savický, P. and Hlavácová, J., (2002) Measures of word commonness. Journal of Quantitative Linguistics 9, (3), 215–231.
7. Berger, G. and Memisevic, R., (2016) Incorporating long-range consistency in cnn-based texture generation. arXiv preprint arXiv:1606.01286.
8. Simonyan, K. and Zisserman, A., (2014) Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
9. Berger, G. and Memisevic, R., (2016) Incorporating Long-range Consistency in CNN-based Texture Generation. arXiv preprint arXiv:1606.01286.
10. Rudin, L.I., Osher, S., and Fatemi, E., (1992) Nonlinear total variation based noise removal algorithms. Physica D: nonlinear phenomena 60, (1-4), 259–268.
11. Prakash, S., 2011. Image Processing is Done using WPF. Retrieved June 18th 2019 from https://www.codeproject.com/Articles/237226/Image-Processing-is-done-using-WPF.