ImagineNet: Style Transfer from Fine Art to Graphical User Interfaces

Michael Fischer
Stanford University
Stanford, CA 94305
mfischer@stanford.edu

Richard R. Yang
Stanford University
Stanford, CA 94305
richard.yang@cs.stanford.edu

Monica S. Lam
Stanford University
Stanford, CA 94305
lam@cs.stanford.edu

Figure 1: Users can stylize applications using artwork of their choosing. For each style image, a feed-forward image transformation network is trained. The trained network is optimized for the GUIs, which requires the output to be usable and legible. Once the feed-forward network is trained, it can be applied to any GUI.

ABSTRACT
Can we leverage the large corpus of art online to enhance the graphical user interfaces in apps? This paper presents a deep-learning based style transfer algorithm that uses art to restyle a GUI while retaining its functionality. We show that previous work in style transfer does not work because it lacks the structural elements in a style, which can be captured by a correlation between features in different layers of the convolutional neural networks. This work allows for the creation of GUIs that integrate the aesthetic characteristics of art with the functionality of applications.

ACM Classification Keywords
D.2.2. Software Engineering: Design Tools and Techniques

Author Keywords
Style transfer; graphical user interfaces; automated design.

INTRODUCTION
While the majority of websites and apps have “one-size-fits-all” graphical user interfaces (GUIs), many users enjoy variety in their GUIs [40, 31]. Variety can be introduced in a top-down manner. For example, Google’s search home page changes for holidays and other special events. Variety can also be introduced bottom-up. The browser, for example, allows users to use third-party scripts to restyle commonly used websites [38]. Application launchers for Android allow users to choose from a wide collection of third-party themes that change common apps and icons to match the theme of choice. These stylizing approaches require hardcoding specific elements for a programming framework; they don’t generalize and are brittle if the developers change their application.

The above examples suggest that users desire unique stylistic features in the interfaces they commonly use, but no platform exists that can support the automated customization of any interface. Furthermore, the available customizations are limited to finite sets pre-specified by a subset of applications. We therefore sought to develop a system that would enable end users to restyle any interface with the artwork of their choosing, effectively bringing the skill of renowned artists to the end user’s fingertips. In this way, users can change the look and feel of their apps by picking a piece of art they enjoy or that fits their mood of the day. We imagine a future where users will expect to see beautiful designs in every app they use, and to enjoy variety in design like they would with fashion today.

This paper proposes to model stylizing GUIs as an image transformation problem. Instead of changing the code that generates the GUI, this paper explores transforming the display of the app according to the style embedded in a user-supplied image. Our display transform algorithm is based on style transfer, a deep-learning technique shown to be effective in transferring the style of a painting onto images of natural objects. The insight behind the original style transfer algorithm is to leverage the pre-trained representation of images in deep convolutional neural networks such as VGG-19 [36]. Using
information from the activations of the input image and the style image in the neural network, the transfer algorithm can reconstruct an artistic image that shares the same content in the input but with the texture of the style image. This result sheds light on the biological process of how we perceive art, which likewise involves parallel processing of decomposed features that are subsequently bound together at higher levels of the processing hierarchy.

Applying the algorithm to GUIs turns them into artistic renditions, but the result is no longer functional as a GUI. The textual and graphical messages in the original content become ineffective and the input elements are not well defined. While the current style transfer model may suffice in creating remixed works of art, it obscures information critical to a GUI. One approach to this problem would be to try and repair the model’s output using traditional image processing techniques such as edge sharpening and optical character recognition. Instead, we want to understand how a neural model perceives GUIs and use that to improve style transfer directly. We ask what is the information that is important to GUIs missing in the existing style-transfer model? How is this information represented in our neural networks? Is this information available in existing deep convolutional neural networks pretrained with natural images? If so, can we develop a machine-learning algorithm to stylize a GUI properly?

This paper presents ImagineNet, a feed-forward image transform network, that can be trained to transform GUI images according to a style, while retaining their functional characteristics. ImagineNet is based on the real-time style transfer architecture proposed by Johnson et al. [20], which uses a loss network based on VGG to train the image transformer. We identify the layers in the VGG [36] sensitive to text, which is important for GUIs. We introduce a new learning metric, a structure loss function, to enforce the integrity of structures in the reconstruction of the styled image.

The technique presented in this paper can be used by developers to explore the design space. A new GUI programming framework can be developed that leverages this technique to support a large variety of styles, possibly supplied by third-party vendors or even users. With further speed optimization and faster GPU hardware, this technique may be used directly to restyle any screen in real time in the future.

Our GUI-focused style transfer algorithm is successful in generating novel, interesting, and artistic interfaces. When we style with art, we find the resulting GUI unlike any that we have encountered before. The details on each UI element, and on each page, and across different pages are more intricate, and yet stylistically consistent. Resembling art work, they are often beautiful and stunning, a significant departure from the GUIs today. But beauty is in the eye of the beholder, hence ImagineNet’s ability to create a new GUI from any piece of art is ever so important.

RELATED WORK
The primary function of our application is based on style transfer, a technique introduced by Gatys et al. that uses deep neural networks (DNNs) to recreate images in the style of another specified image [14]. This work showed that image representations learned by DNNs encode both the content and style of an image. This algorithm uses a content and style image as input, and iteratively optimizes for a new image that minimizes the content loss and style loss, which are computed using the images representations learned by the DNN from the input images.

Gatys’s style transfer method was expanded on by Chambard to allow for semantic disambiguation in the generated images [8]. In addition to input style and content images, the semantic style transfer algorithm also makes use of semantic annotations for localized style transfer. We refer to models based on these two techniques as descriptive style transfer models because a DNN is used to describe the style and content of an image.

Semantic style transfer maintains the global structure of an image through the annotation map, but performs poorly at the local level. UI elements generally have perfect geometry. For example, buttons are composed of straight lines, with color gradients. This is problematic for semantic style transfer because this algorithm was designed to stylize real world images, which rarely have perfect geometry due to camera skewness and other factors. Thus, semantic style transfer on UI images produces buttons that don’t have straight lines and circles that are not perfectly circular.

These descriptive style transfer models are optimization-based methods, and therefore suffer from a major drawback: efficiency. These methods take minutes to converge per frame, which is not ideal for our application that needs to run in real time. Therefore, we alternatively look to the feed-forward style transfer model described by Johnson et al. In this framework, the network is trained with respect to a style image to transform an input image to an image in that style image in a single pass. This model requires hours of training time, but can transfer an image for a trained style in real time.

In this paper, we use style transfer as a means to automatically customize the style of a GUI. Previous work from Gajos et al. sheds light on methods to automatically customize the functionality of a GUI, and created a tool that automatically adapts UIs for people with motor impairments. Their system, called Supple, models interface generation as a discrete constrained optimization problem, and produces optimized UIs given an interface specification [12].

The work of Kumar et al. introduces a platform called Webszeitgeist, which allows users to parametrically search through a design space using a concept called design mining [24]. Previous work on web data mining focuses on the content of webpages and entirely discards the design and presentation of said content. However, style is tantamount to content in the sense of user interaction, and this platform enables users to analyze and learn from style.

While there have been several previous works on the automatic generation of functional layouts for GUIs, generating new styles of GUIs is a novel idea. Previous works are well suited for optimizing a list of components, but do not take into account visual design considerations or aesthetics when
When creating ImagineNet, we faced many design questions. In this section we describe the principles of design that informed the choices made.

1. **Form follows function.** The functionality of the GUI is crucial and should be preserved after being stylized. Functionality in our usage means that UI elements are clear and that text and numbers are readable. The style shouldn’t overwhelm the purpose of the GUI.

2. **Aesthetics.** Our second goal is to create aesthetically appealing designs. With style transfer, we faced a three-way tradeoff: quality, speed, and generality. This paper focuses on quality.

3. **Non-incrementality.** We are not trying to find a practical solution that fits into today’s drag-and-drop toolkit approach, but rather a fresh direction in GUI design that can leverage the incredible artwork in the world.

**STYLE TRANSFER**

Our style transfer algorithm is based on the feed-forward image transformation network [20] from Johnson et al., as shown in Figure 2. This system consists of two networks, the image transform network that styles inputs according to a particular trained style, and a loss network used to train the image transform network. The image transformation network is trained with content images from the Microsoft COCO [28] dataset. A fixed loss network, the VGG19 network pre-trained on ImageNet [35], is used to calculate loss between the generated image and target image. The loss calculated from VGG19 is used to backpropagate the weights in the image transform network.

The image transformation network, $T$, is a deep residual convolutional neural network with deconvolutional layers [34, 32]. The model consists of three down-sampling convolutional layers with stride two, five residual blocks, three upsampling convolutional layers with stride two, followed by a tanh output layer. The model is parameterized by weights $W_s$, which are learned with the help of the loss network to capture how to transform an image for a given style $s$.

$T$ transforms an input image $x$ to an output image $y$ with $y = f_{W_s}(x)$. The goal of training is to find weights $W_s$ of the image transform network for a specific style $s$ to minimize the loss functions that measure the difference between the output image, $y = f_{W_s}(x)$, and the pair of input $x$ and style image $s$.

$$W_s^* = \text{arg min}_{W_s} \mathbb{E}_x [\lambda_1 L_{\text{CONTENT}}(x, y) + \lambda_2 L_{\text{TEXTURE}}(x, y) + \lambda_3 L_{\text{STRUCTURE}}(s, y) + \lambda_4 L_{\text{TV}}(y)]$$

The original algorithm uses three loss functions, $L_{\text{CONTENT}}$, $L_{\text{TEXTURE}}$, and $L_{\text{TV}}$. To make the transformed GUI functional, we introduce the $L_{\text{STRUCTURE}}$ loss function to capture how the structure of objects is represented. The overall loss is a weighted sum of these loss functions, and the $\lambda$ values are the weights in the sum. We will summarize the original functions below before discussing $L_{\text{STRUCTURE}}$.

**Content loss**

To ensure that the output $y$ retains the content of the input $x$, the loss content function $L_{\text{CONTENT}}(x, y)$ computes the difference in features between $x$ and $y$ in a high layer of VGG.

Let $F^\ell_{h,w}(x)$ be the activation of the $h$-th filter at position $w$ in layer $\ell$ for input image $x$.

$$L_{\text{CONTENT}}(x, y) = \sum_{h=1}^{H_s} \sum_{w=1}^{W_s} (F^\ell_{h,w}(x) - F^\ell_{h,w}(y))^2$$

The layer representing content, $\ell$, is “relu4_2”.

**Texture loss**

The texture of an image is captured by the lower layers. Gatys et al. represent the texture of a layer as the Gram matrix between pairs of features from the same layer, the inner product of a pair of feature maps:

$$G^\ell = \frac{1}{C^\ell H_s W_s} \sum_{h=1}^{H_s} \sum_{w=1}^{W_s} F^\ell_{h,w}(x) (F^\ell_{h,w}(x))^\top$$

To copy the texture of the style image, the texture loss function compares the textures between the output image and the style image:

$$L_{\text{TEXTURE}}(s, y) = \sum_{\ell \in \text{layers}} \beta_{\ell} (G^\ell(s) - G^\ell(y))^2.$$ 

Where $\beta_{\ell}$ is the loss weight factor for layer $\ell$. The layers used are: “relu1_1”, “relu2_1”, “relu3_1”, “relu4_2”. The weights are shown in Table 1

**Total Variational Loss**

Prior work on style transfer adds a total variation regularization [29] to the loss function to ensure smoothness in the optimized image [14, 20].

$$L_{\text{TV}}(y) = \sum_{h,w} |y_{(h+1,w)} - y_{(h,w)}| + |y_{(h,w+1)} - y_{(h,w)}|$$
This denoising loss function helps remove artifacts that result from the striding and padding mechanisms in the convolution layers of the CNN. Total variation encourages spatial smoothness in the image and prevents pixelation, at the cost of blurriness. We have adjusted the total variational weight lower to reduce blurriness in the images.

**OUR STYLE TRANSFER FOR GUIS**

Existing style transfer algorithms [20, 14] have predominantly been used and optimized for transferring the style of a painting to an image of natural objects, like humans, animals, and sceneries. When applying style transfer to GUIs, the resulting image lacks definition. Figure 3 shows a comparison between existing style transfer techniques applied to two different GUIs using two different styles. We first observe that the result copies the style of the paintings well in terms of color, brush strokes, and the boldness of lines. The result also maintains the core content of the original.

However, the output information is no longer discernible. In the case of the music player interface, we cannot read the title of the song or recognize the progress bar. Similarly, the buttons, the volume slider, and the heart to indicate “liking” are no longer clear. Colors in the stylized image also bleed across object boundaries. Even the two panels, the top with the song information and the bottom the control, are not well distinguished from each other and with respect to the background.

Thus, we identify two major problems in style transfer for GUIs: (1) the lack of clarity on text after the transformation and (2) the loss of *structure*, which we define as the lines or boundaries of objects and the consistency of colors and textures within an object.

**Ensuring Text Clarity**

Text is an important element across all the GUI screenshots in our dataset. The original style transfer algorithm does not preserve the clarity of text from the content image. Text is treated and stylized the same as other visual elements, resulting in hard-to-read, albeit artistic, text.

We explored various alternatives such as removing text from the input image using optical character recognition[16], and reinserting the text afterwards, but we found the quality of results unacceptable. Instead of making a special case out of text and handling it out of band, we perform the full image transform using a neural model. Text, after all, may be conceptualized as detailed lines and curves from an imagery point of view.

Our first task is to understand how text is learned and represented in a convolutional network. We visualize the neuron activation maps of VGG19 to identify the layers of neurons that are salient to text in an image. Neuron activation maps are plots of the activations during the forward propagation of an image. Since GUIs are not a class in ImageNet, which VGG19 was trained on, we take a random image from an arbitrary ImageNet class and overlay text on it. We then forward pass this combination image into the network to obtain the activations, and identify the layers with the highest activations with respect to text.

From our visualizations, we observe that neurons in the “relu1_1” and “relu3_1” layers of VGG carry the most salient representations of text in images. The intuitive explanation is that earlier layers in image classification neural networks tend to be edge and blob detectors, while later layers are texture and part detectors [26]. Text is primarily composed of straight lines, which supports the observation that neurons in the earlier layers have activations toward text. These results suggest that greater weights must be placed in these layers to ensure that text-like information is carried over in the output image.
We capture the design of structural elements by linking the higher-level features to the lower-level features that make up those higher-level features. Like Gatys et al., we also use the Gram matrix to represent the correlation between cross-layer features, and introduce a structure loss function to minimize the difference in the correlation of features between the output image and the style image across layers.

\[
L_{\text{structure}}(s, y) = \sum_{\ell \in \text{layers}} \sum_{m \in \text{layers}} \gamma_{\ell, m} (G'_{\ell, m}(s) - G'_{\ell, m}(y))^2
\]

where

\[
G'_{\ell, m} = \frac{1}{C_H W_H} \sum_{h=1}^{H_{\ell}} \sum_{w=1}^{W_{\ell}} F_{h, w}^\ell(x)(F_{h, w}^m(x))^T
\]

and \(\gamma_{\ell, m}\) is the loss weight factor. This loss function between two layers will copy over the correlation between different layers from the style image to the output image. As the higher layer spans over a larger space extent \([26, 17, 18]\), this correlation helps keep the features of the lower layer consistent.

As we go up the VGG network, the width, height, and the number of filters change. The first layers have a small width and height, as the layers increase their width and height increase and depth decrease. When taking a convolution between layers of different dimensions, we upsample the width and height of the smaller layer to match the larger layer.

The configuration we found to produce the best result adds the cross-layer correlations of “relu1_1” × “relu1_2”, “relu1_1” × “relu2_1”, “relu2_1” × “relu2_2”, “relu2_1” × “relu3_1”.

**Structure Loss**

Extensive experimentation with style transfer leads us to the following understanding. The content representation at the high layer of the convolutional neural network is coarse, since the convolution filters at the upper layer cover a large spatial area. The texture representation on the other hand is computed by correlating pairs of features across the entire image, a layer at a time. There is no linkage between the high-level objects and the texture used.

The image transformer has the freedom to randomly choose textures as long as the objects are still recognizable at a coarse granularity. Boundaries are lost in this representation because the higher-level content is too coarse to dictate the placement of a straight edge, and the texture information is not positional. Thus, a straight edge rendered with different textures and colors creating blurry and jagged lines. Similarly, large areas with variance in luminocity may result in patches of inconsistent styles. In fact, the inconsistency in neighborhood textures is the likely reason why total variation is needed to smooth out the image and provide consistency during training.

Boundaries, borders, and consistency of colors are the elements that help us see the structure in the image. The design of these elements is part of the style; a painting can choose to mark boundaries between objects with bold lines, or add shadows around borders. Clearly, the structure design is present in the convolutional neural network, it is just missing from the original style transfer model, hence objects lose definition.
Decreasing batch size has previously been found to increase the quality of the model [37]. The optimal batch size depends on the curvature of the loss function. Larger batch sizes have been found to tend toward local minimum more so than smaller batch sizes when the loss function is nonlinear [22]. For each step the optimizer takes, when the cost function is highly curved, the approximation made by the optimizer is only accurate for small step sizes [15] and leads to better quality models.

From a design and usability perspective, having smaller batch sizes is ideal. With a small batch size, a user can tell if their model is promising in less than a minute. Being able to get fast feedback has been shown to improve the quality of design [11, 6]. Smaller batch sizes also allow people with smaller GPUs to train their own models. We were able to train a batch size of 1 on a 4GB GPU. We are already seeing high-end mobile devices with custom GPUs [9, 2]. In terms of run-time effects with a batch of size two, two frames could be sent through the image transformation network simultaneously, and a buffer of anticipated frames could be created so that only every other frame has to be re-rendered on the GPU.

IMPLEMENTATION
For development and benchmarking we used a computer with 6 core 3.50 GHz processor with 48 GB of RAM. The computer was running Cuda V9.0.176 and cuDNN V6.0 with an Nvidia Titan V 12GB, Nvidia Titan X 12GB and Nvidia GTX 980Ti. The code was written in TensorFlow 1.5 with Python 3.5.

EVALUATION METRICS
Our metrics to evaluate the success of style transfer are:

- **Functionality.** Functionality is the most important metric: all the user interface elements and the text should be clear.
- **Aesthetics.** The aesthetics and colors should be pleasing[1], and the result should appear well designed.
- **Artifacts.** The feed-forward image transform may introduce artifacts as it reconstructs an image [32]. Are there any areas of the image that have color blemishes or lines that are going in the wrong direction?

Based on the above metrics, we experimented with the weights of the loss functions to arrive at the results shown in Table 1. When we assigned too much weight to the lower layers in the loss function, mostly only color and smaller elements of styles were transferred over, and the results look flat. So, we began to introduce more weight to layers 3 and 4. However, this caused the lines again to blur. We then added more weight to structural layers, and arrived at a model that transfers good style and shows clear text. To determine the total variational loss weight, we lowered the weight incrementally until artifacts like checkerboarding appeared. Our style-transfer algorithm can be adjusted to match different evaluation metrics by changing the weights of the loss functions.

## RESULTS
### Quality of Style Transfer
For a complete perspective on our algorithm, we show our results in Table 2 across multiple visual styles, described in Table 3, and GUI content archetypes, described in Table 4.

- **Aesthetics.** First, we observe that the results are faithful in color and texture. The colors and styles are balanced, like the original artwork. For example, in the pizza app, the buttons are well balanced with the background. The background is well textured and interesting.
- **Functionality.** The home screen, the map, and the game, have important information in the displays. Our style transfer algorithm succeeds in presenting all the detailed information, while still showing the style of the original artwork. We note that the information density of an icon, or the boundaries in countries, is no less than text. The neural network approach of not specializing on words generalizes to other information types.
- **Tradeoff between style and information.** Our style transfer algorithm seems to “understand” where it has room to apply artistry, yet also “knows” to reduce the artistry when displaying information. For large areas with no informational content, the style transfer becomes more creative, copying over the

<table>
<thead>
<tr>
<th>Loss Function</th>
<th>Layer</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_{\text{CONTENT}}$</td>
<td>$\text{relu}_4$</td>
<td>5.6</td>
</tr>
<tr>
<td>$L_{\text{TEXTURE}}$</td>
<td>$\text{relu}_1$, $\text{relu}_2$, $\text{relu}_3$, $\text{relu}_4$</td>
<td>1.1, 1.3, 0.5, 1.0</td>
</tr>
<tr>
<td>$L_{\text{STRUCTURE}}$</td>
<td>$\text{relu}_1$ × $\text{relu}_2$, $\text{relu}_1$, $\text{relu}_2$, $\text{relu}_2$, $\text{relu}_3$</td>
<td>1.5, 1.5, 1.5</td>
</tr>
<tr>
<td>$L_{TV}$</td>
<td></td>
<td>150</td>
</tr>
</tbody>
</table>

Table 1: Weights for the loss functions.
<table>
<thead>
<tr>
<th>Style</th>
<th>GUI</th>
</tr>
</thead>
</table>

Table 2: Results of transferring the style of the paintings from Table 3 to the GUIs from Table 4. The details can best be viewed on a computer.
We observe that for the home screen GUI, all styles keep the same grid structure and button layout. The style is composed of two primary components, edges and colors. Paintings with vibrant colors work the best. The hard-edge paintings do the best on these two metrics, because they had the most distinguished lines and colors. Older paintings, whose color has faded, do not transfer well.

Paintings with soft strokes create a GUI unlike any GUI that we had seen. While maintaining the look of a painting, with our new style transfer algorithm, the UI components are still easily visible and usable.

Style transfer works well when there is sufficient complexity in both the style and the GUI. For simple GUIs, such as a note taking app which is primarily blank, the benefits of style transfer are not fully realized. We found that many GUIs have two to three colors, a primary, secondary, and sometimes a tertiary color that acts as an accent. If there is not enough complexity in the style image, the resulting image is constrained to the inputed style, and the resulting image looks too uniform.

Our style transfer algorithm copies how the structure is rendered. If the style image uses simple black lines to separate adjoining surfaces, the resulting target image will also have simple black lines. Similarly, if the painting does not have interesting textures, then the style transfer will result primarily in a transformation in a color domain.

Lastly, we noticed that the style transfer algorithm can create unexpected effects with some image assets within apps. The jpeg format is designed to minimize the human perception of compression, but it introduces visual artifacts that are not icons in the original, whereas style transfer does a better job in making the two rows more distinct and the icons more visible.

Borders. The borders generated by our algorithm are crisp; this is easily observable from the home screen GUI. The style of the borders from different paintings are carried over by our style transfer algorithm. The strong borders in the original artwork in columns 2, 4, and 5 can be seen in the boundaries of the continents in the map GUI.

Choice of Styles
To understand what kind of style images can generate good looking GUIs, we tested style transfer extensively with the paint-by-numbers dataset [21]. We found that style transfer works well in general, but some paintings produce better results than others. The style transfer works best when using art work that is stylistically interesting.

Style is composed of two primary components, edges and colors. Paintings with vibrant colors work the best. The hard-edge paintings do the best on these two metrics, because they had the most distinguished lines and colors. Older paintings, whose color has faded, do not transfer well.

Paintings with soft strokes create a GUI unlike any GUI that we had seen. While maintaining the look of a painting, with our new style transfer algorithm, the UI components are still easily visible and usable.

Style transfer works well when there is sufficient complexity in both the style and the GUI. For simple GUIs, such as a note taking app which is primarily blank, the benefits of style transfer are not fully realized. We found that many GUIs have two to three colors, a primary, secondary, and sometimes a tertiary color that acts as an accent. If there is not enough complexity in the style image, the resulting image is constrained to the inputed style, and the resulting image looks too uniform.

Our style transfer algorithm copies how the structure is rendered. If the style image uses simple black lines to separate adjoining surfaces, the resulting target image will also have simple black lines. Similarly, if the painting does not have interesting textures, then the style transfer will result primarily in a transformation in a color domain.

Lastly, we noticed that the style transfer algorithm can create unexpected effects with some image assets within apps. The jpeg format is designed to minimize the human perception of compression, but it introduces visual artifacts that are not

<table>
<thead>
<tr>
<th>Content</th>
<th>Content Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home Screen</td>
<td>Many detailed icons with strong information content.</td>
</tr>
<tr>
<td>Pizza App</td>
<td>A large blank area with a varying black shade and a logo and large words.</td>
</tr>
<tr>
<td>World Map</td>
<td>Detailed graphical and textual information.</td>
</tr>
<tr>
<td>Coffee App</td>
<td>Simple graphics with clean lines.</td>
</tr>
<tr>
<td>Game</td>
<td>Lots of details critical for the game play.</td>
</tr>
<tr>
<td>Music App</td>
<td>App for drum sounds. App has many large buttons of uniform color.</td>
</tr>
</tbody>
</table>

Table 4: Description of content GUIs[10] used in Table 2.
Table 5: Run time for a forward pass on a GUI through the image transform network.

<table>
<thead>
<tr>
<th>Width</th>
<th>Height</th>
<th>Pixels</th>
<th>Run Time (ms)</th>
<th>Standard Deviation (µs)</th>
<th>Time/Pixel (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>397</td>
<td>306</td>
<td>121482</td>
<td>1.31</td>
<td>0.46</td>
<td>0.015</td>
</tr>
<tr>
<td>512</td>
<td>288</td>
<td>147456</td>
<td>2.82</td>
<td>0.46</td>
<td>0.019</td>
</tr>
<tr>
<td>552</td>
<td>736</td>
<td>406272</td>
<td>9.62</td>
<td>0.55</td>
<td>0.024</td>
</tr>
<tr>
<td>861</td>
<td>506</td>
<td>435666</td>
<td>8.42</td>
<td>0.90</td>
<td>0.019</td>
</tr>
<tr>
<td>964</td>
<td>559</td>
<td>538876</td>
<td>16.41</td>
<td>0.54</td>
<td>0.030</td>
</tr>
<tr>
<td>680</td>
<td>1200</td>
<td>816000</td>
<td>16.82</td>
<td>1.31</td>
<td>0.021</td>
</tr>
<tr>
<td>1275</td>
<td>765</td>
<td>975375</td>
<td>40.21</td>
<td>0.84</td>
<td>0.041</td>
</tr>
<tr>
<td>1334</td>
<td>750</td>
<td>1000500</td>
<td>14.20</td>
<td>0.82</td>
<td>0.014</td>
</tr>
<tr>
<td>1920</td>
<td>1080</td>
<td>2073600</td>
<td>102.42</td>
<td>1.34</td>
<td>0.049</td>
</tr>
</tbody>
</table>

clearly visible to the human eye. However, when the style transfer algorithm is applied, the visual artifacts are recognized as style, generating a glowing effect. This problem can be fixed by using high-quality lossless images.

**Speed of the Algorithm**

In the following, we report the time it takes to train the transformation network and to apply it to an input.

**Training Time**

The time to train the image transformation network is dependent on the quality of the results desired. To get the best quality results we had to train, batch size of one, for 10 hours. Figure 6 shows the results with a varied amount of training. As shown, the background, text, and textures develop over time.

**Run Time**

Once the image transformer is trained for a style, the transformation of an image is fast. As the size of the model for the image transformer is fixed, the run time is invariant to the style used to train the image transformer. This is validated by running the same GUI input for ten different styles and finding that there was no statistical variation in time it took to render the target image.

To determine the style transfer time on a trained model, we constructed a test suite of seven 3-channel color images of varying sizes. An image transform network is constructed for each image dimension, where the initial size of the input is determined by the image size. Averaged over five runs, a 512x288 sized image took 3.02 ms while a 1920x1080 sized image took 104.02 ms. On average, the transformation took 0.026 µs/pixel. Full results are shown in Table 5.

**USER TEST**

We conducted a user test to determine how people would compare our algorithm to other style transfer techniques, to determine if the text was legible, to see how users would use such a system, and to gather open-ended feedback. We conducted an interview with seven users, four male and three female, four undergraduate and three graduate students.

The first test was to determine how well our style transfer worked in comparison to others. We showed two results from all three algorithms. In all cases, users chose ours as the GUI that they would prefer using.

Next, we conducted a visual acuity test by having people read the text on the transferred style images. Everyone was able to read all the text.

We then tested how interpretable the transfer of style to content was for users. We constructed a matching game whereby we placed six styles on the left and six stylized GUIs on the right and had users try and match the pairs. Two users missed a pair. Overall users got the right answers 90% of the time.

Users liked the beauty of the applications and the overall way that they looked. Some users liked the way all of them looked but other users had a preference toward particular styles. Out of the six we showed people, everyone found at least two that they liked.

Users had a wide variety of interest in how they would use the tool. For one user, visual consistency was important, and they wanted to use the tool to make the style of all their apps the same, so their phone would be visually consistent. Users also commented that they would take more pride in using an application that they created. One user said that they would like to create a mood board [13] to help them when designing application. Overall, the experience most users wanted was to quickly see many possible styles and then choose their favorite.

**DISCUSSION**

**End-to-end vs. component-based styling**

GUIs are typically created with toolkits, like Twitter’s Bootstrap or Google’s Material, where styles are bundled together. We considered using style transfer to allow designers to create similar bundled styles for UI elements. This would have avoided the problems we had to tackle, to preserve straight edges and eliminate color bleeding across boundaries. However, this approach lacks enforcing global consistency across all elements in a full UI.
We experimented with breaking down a GUI by removing all the text and segmenting the UI elements, applying style transfer on the individual elements, with the plan to re-assemble the individual elements later. We discovered that the results do not look good. When the elements were re-compose back together, they did not fit together as well compared to the end-to-end approach. We hypothesize that style transfer needs to take into account the relative importance of each element based on the properties that it learns: their size, position, and proximity to other elements. Furthermore, the end-to-end approach adds interstitial material to the GUI, which helps bring all the components together, creating an overall gestalt. In a sense, the end-to-end style transfer approach adds the garnishes to give the apps a professional and artistic look.

The results of ImagineNet look stylistically distinct from other user interfaces that we have encountered because of the depth of style that is created. This depth of style is enabled by artists who perfected their art through years of artistic endeavor.

**Real-Time Transform**

Our GUI style transform algorithm can transform an image in real time on a PC with a GPU. GPUs are already available in many mobile devices and will become more powerful and pervasive [19]. Along with further software optimizations, we expect real-time transformation on mobile devices to be viable in the future.

**Applications of GUI style transfer**

The style transfer algorithm can be used in several different ways.

**At GUI Design Time**

Developers can leverage the style transfer algorithm to get inspirations and to quickly explore the design space. While training for best quality takes up to 10 hours, our algorithm supports exploration because it can also return results after one minute of training. Training for multiple styles can be parallelized by using multiple GPUs. We anticipate that the work flow for a designer will be as follows. They will watch the first few iterations to see if the design looks promising; if so, they can let the design train to completion while working on another design on a different GPU.

**Dynamic GUIs**

This approach also makes possible apps that vary the themes based signals from the user or computer. For example, a music playing app can style itself according to the album art of each song; a fashion store can style their app differently for different category of clothes; graphical virtual assistants such as Alexa Show can show different GUI styles for different skills. Using style transfer, an image gallery could automatically customize itself for the images being displayed.

Styles could also be varied automatically based on the user’s mood, weather, or time of day. Today, Apple’s night-shift mode automatically adjusts the color of the display in the evenings to help users fall asleep. Overall, the computer can take an active role in creating delightful user experiences.

**CONCLUSION**

This paper shows that we can use a style transfer algorithm, based on deep learning, to restyle a GUI with fine art. Previous work on style transfer is unsuitable for GUIs because the loss in informational content renders the GUIs not functional. The problem is that the existing algorithm only captures the high-level content and the low-level texture information. We show that information can be retained, while being re-styled, by transferring over the structural elements from the style image, as represented by the correlation of features across the lower layers in a VGG network.

By introducing the structure loss function into style transfer, we successfully restyle GUIs with varying complexities. The small detailed are clear enough to be played in Candy Crush, the countries in a map are clear enough for planning. Not only is the text clear, the algorithm copies over detailed information found in icons.

GUIs, when styled with art, look beautiful. There is texture in the background, contrast in the borders, consistency of design across buttons, and an overall cohesiveness to the design.

This technique can be used to create new and varied GUIs. It can be built into future GUI design frameworks. With further optimization and better hardware, it may even be used by end users to personalize their interfaces dynamically in the future.

**REFERENCES**


