Towards Automatic Icon Design using Machine Learning

Moses Soh (msoh) MOSES.SOH@GMAIL.COM

Abstract

We propose a deep learning approach for automatic colorization of icons. The system maps grayscale outlines to fully colored and stylized icons using a Convolutional Neural Network (CNN). The colorization is performed in a single feed-forward pass, enabling colorizations in less than 0.5s per image. Compared to previous research in colorization of grayscale images or anime characters, icon colorization is significantly more multi-modal and icon outlines have less structure to guide colorization. This poses a challenge to traditional supervised learning training regimes. Our main contributions are (1) framing this as a supervised learning problem and presenting the results of optimizing a U-Net CNN architecture on per-pixel L1 loss, (2) performing ablative analysis on our data augmentation pipeline and (3) demonstrating that introducing an adversarial loss term can produce color reproduction more faithful to the original icon styles. This is the first system we know of that applies CNNs to the colorization of icon outlines. Our code and models are available at https://mosessoh.github.io/iconcolor

1. Introduction

Despite breakthroughs in image generation after models such as Neural Style Transfer and Generative Adversarial Networks (GANs) came out, the majority of these have dealt with photo-realistic images (e.g. ImageNet or LSUN datasets) or paintings. This work explores the problem of training generative models on a vector graphic dataset of icons. Purely unsupervised generation or even conditional generation of icons is in some ways a more challenging task than generating photo-realistic images due to many possible colorizations outcomes for a given input, and less structure to guide the colorization (e.g. vegetation is usually green in photographs). Yet, icon design (e.g. for logos) is a common problem faced by many people (e.g. "logo maker" is searched 9x more than "website maker"), who need to choose between hiring professional designers from Dribbble / Behance, freelancers from sites like 99designs and Fiverr, or doing it themselves manually. All these still result in a human manually choosing colors, shapes and styles to craft an icon. This research explores how machine learning techniques can be employed to convert icon outlines to fully stylized icons.

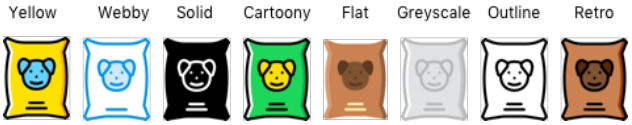

Figure 1. We explore different deep learning architectures and hyperparameters to find a mapping from outline icons to other styles.

2. Related works

2.1. Image editing using deep networks

This research draws upon recent literature showing that neural networks perform impressively on various image processing tasks such as super-resolution (Hayat, 2017), style transfer (Gatys et al., 2015) and grayscale image colorization (Zhang et al., 2016). They differ in terms of model architecture and test time performance. The initial style transfer methodology used back-propagation on raw pixel values to transfer styles. This was set up as a supervised learning problem, where the target to achieve was specified by the content and style layer activations (and transformations thereof) of a pre-trained deep convolutional model after separately forwarding a style image and a content image through the net (Gatys et al., 2015). This work showed that layers low down in a convolutional network stored content-related information whereas the Gram matrix of higher layers contained information on style such as colors and repeated patterns. However, this method of style transfer was slow as back-propagation had to be carried out every

Stanford CS229 Final Project

time users wanted to create a new image. Further work improved on this by training a deep convolutional neural net (CNN) that took in a content image as input and generated the style transfer in one forward pass of the CNN (Johnson et al., 2016). This also improved on prior work that had also trained deep CNNs for style transfer, colorization and super-resolution by using perceptual losses rather than a per-pixel loss (Dong et al., 2015). Recent systems are primarily focused on end-to-end training of CNNs that produce the desired image transformations using a single forward pass at test time for performance (Isola et al., 2016; Zhang et al., 2017). This is the approach we adopt in this paper.

2.2. Colorization

User-guided colorization. The colorization literature can be broadly divided into two camps. Early user-guided colorization methods focused on incorporating user strokes (Dani et al., 2004) and propagating those user inputs via optimization routines throughout the image. As strokes were propagated based on local similarity metrics between pixels, effective colorization required numerous user hints. More recently, deep learning techniques have been employed to learn colorization mappings taking in user-provided pixels and strokes as inputs with excellent results on anime art (Networks, 2017) and line drawings (Frans, 2017). However, these rely on users being able to choose sensible pixel color hints.

Fully-automatic colorization. These range from the earlier semi-automatic methods that transferred statistics such as the color histogram, contrast and brightness from reference images to target images (Liu et al., 2014), to more recent fully automatic methods that use priors learned over large natural image datasets to automatically color or generate images (Isola et al., 2016; Deshpande et al., 2015). Some papers also improve upon user-guided colorization by not only taking in hints, but giving the user a histogram of colors to choose from that is derived from the natural image manifold. The best qualitative results have come from combining UI insights from user-guided colorization (i.e. guiding users to the right color) and using automatic colorization to figure out the most natural colorizations conditioned on these user choices (Zhang et al., 2017). In this research, we experiment with the fully automatic colorization approach, as pairing and combining icon colors is something we hope users can avoid having to do.

3. Methods

We train a deep network to produce the fully colored version of an icon given its outline. The objective of the network is described in Section 3.1. The architecture of the network is described in Section 3.2. Finally, our training methodology and data pre-processing is described in Section 3.3.

3.1. Data

Our data $\mathcal{D}$ consists of ∼70,000 icons. There are ∼9,000 unique icons, which are drawn and colored in eight different styles (Figure 1)[1].

Data preprocessing. In this research, we focus on converting "Outline" icons to "Yellow" icons. There are 7,000 Outline-Yellow icon pairs. We convert each "Outline" icon into $\mathbf{X}^{(i)} \in [0,1]^{1xHxW}$, since the only colors in the "Outline" icon are varying shades of gray. For each "Yellow" icon, we leave it in RGB space. Since the majority of the 7000 icons are roughly 300 x 300px, we remove 13 outliers dramatically different aspect ratios. We then paste all icons onto a 300 x 300px white background to remove the α layer. All icon sizes are normalized to occupy the same space in the frame.

Data splits. We split the dataset of $7,000$ "Outline" to "Yellow" icon-pairs into a training set of size $6,488$, validation set of size 256, and a test set of size 256.

3.2. Model setup

Our system takes as input an "Outline" icon $\mathbf{X} \in [0,1]^{1\text{x}128\text{x}128}$. The output of the system is $\hat{\mathbf{Y}} \in [0,1]^{3\text{x}128\text{x}128}$ in RGB space. In previous colorization literature, training data was generated by converting colored images into *Lab* color space, holding the L layer fixed, and making the model learn to predict the ab values from the L layer. Following previous literature's methods would mean that users would have to supply our network with grayscale icons instead of outline icons, and the former are extremely difficult to find online. Our problem is more challenging because we are trying to go from icon outlines to fully colored icons, which means the network must also learn to hallucinate the L layer of the colored image. Since the network has to predict all three layers of the color space, we choose to leave the output in RGB.

3.2.1. Variable definitions

To summarize, our inputs are $\mathbf{X} \in [0,1]^{1\text{x}128\text{x}128}$ which are icon outlines. Our goal is to produce

[1]This dataset was purchased from smashicons.com

$\hat{\mathbf{Y}} \in [0,1]^{3\text{x}128\text{x}128}$ which is to be compared with the designer-produced "truth" which is $\mathbf{Y} \in [0,1]^{3\text{x}128\text{x}128}$.

3.2.2. Network architectures

Define `Ck`, `CDk` and `CUk` as k-layer convolutions followed by Batch Normalization (Ioffe & Szegedy, 2015) and ReLU non-linearity. `CD` and `CU` down-sample and up-sample by a factor of 2x respectively.

Generator. Our icon colorization model $\mathcal{G}$, which we termed the generator, is a function $\mathcal{G} : [0,1]^{1\text{x}128\text{x}128} \rightarrow [0,1]^{3\text{x}128\text{x}128}$. It can be represented by an encoder and decoder.

The encoder:
`C32-CD64-C64-CD128-C128-CD256-C256-CD512-C512`.
The decoder:
`CU512-C256-CU256-C128-CU128-C64-CU64-C32-C3`
followed by a sigmoid non-linearity to squash pixel values to between 0 and 1.

Our U-Net architecture utilizes skip-connections between encoder and decoder layers. The Ck layers in the encoder are concatenated with the corresponding Ck layers in the decoder before being fed into the next decoder layer. All the ReLUs in the encoder are leaky with slope = 0.2. The ReLUs in the decoder are not leaky. This U-Net architecture was first shown to be effective for image segmentation (Ronneberger et al., 2015) using only ReLUs as the non-linearity. It was then shown to be effective used as the generator in an adversarial training context using Leaky ReLU's in the encoder part of the U-Net, due to vanishing gradients using ReLUs (Isola et al., 2016). The shortcut connections in the U-Net allow information that cannot flow through the bottleneck to still be used in the decoding and colorization process.

Discriminator. The discriminator's job is to differentiate between real and generated images. It is a function $\mathscr{D} : [0,1]^{3\text{x}128\text{x}128} \rightarrow [0,1]$ that produces a probability of the input being real or generated. Its architecture can be described by
`CD32-C32-CD64-C64-CD128-C128-CD256-DENSE`
where the `DENSE` layer produces a single probability. Leaky ReLUs with slope = 0.2 are used.

3.2.3. Loss functions

We run two experiments in this paper. Firstly, we explore the use of L1 loss without an adversarial loss term. Previous colorization work have experimented with using the L1 loss (Charpiat et al., 2008; Larsson et al., 2016). However, these also note that the L1 loss is not robust to the multi-modal nature of the outputs, where multiple colorizations can be acceptable. Usage of the L1 loss in this case tends to produce the average color in the dataset, resulting in less vibrant colors in the output. We experiment to see if we face the same problems with this dataset. Secondly, we train the generator with an additional adversarial loss term given by the discriminator and analyze the differences.

Generator without adversarial loss. The L1 loss is used to evaluate the generator. We minimize $\mathcal{L}(\mathcal{G})$ using the Adam optimizer with default parameters.

$$\ell_{L1} : [0,1]^{3\text{x}128\text{x}128}\text{x}[0,1]^{3\text{x}128\text{x}128} \rightarrow \mathbb{R}$$

$$\ell_{L1}(Y,\hat{Y}) = \frac{1}{3 \cdot 128 \cdot 128} \sum_{i=1}^{1} 28 \sum_{j=1}^{1} 28 \sum_{c=1}^{3} |Y_{ijc} - \hat{Y}_{ijc}|$$

$$\mathcal{L}(\mathcal{G}) = \frac{1}{m} \sum_{i=1}^{m} \ell_{L1}(Y^{(i)}, G(X^{(i)}))$$

Generator and Discriminator with adversarial loss. Recent developments in the use of generative adversarial networks (GANs) create an adversarial loss term, where a discriminator network is trained to differentiate between real images from $\mathcal{D}$ and fake images $\hat{\mathbf{Y}}$ created by the generator $\mathcal{G}$, and the generator $\mathcal{G}$ is trained to fool the discriminator (Goodfellow et al., 2014). In the pix2pix model, Isola et al. train the generator $\mathcal{G}$ to minimize the L1 loss as well as an adversarial term, where the L1 loss is used to learn low-frequency structure in the original image (e.g. shapes, large swatches of color) and the adversarial term handles more nuanced patterns (e.g. colour distribution and textures) (Isola et al., 2016). We define our discriminator and generator loss terms under this model as follows.

$$\ell_{BCE} : [0,1]\text{x}\{0,1\} \rightarrow [0,1]$$

$$\ell_{BCE}(o,t) = t \log o + (1-t)\log(1-o)$$

$$\mathcal{L}(\mathscr{D}) = \frac{1}{m} \sum_{i=1}^{m} \ell_{BCE}(\mathscr{D}(G(X^{(i)})), 0) + \ell_{BCE}(\mathscr{D}(Y^{(i)}, 1)$$

$$\mathcal{L}(\mathcal{G}) = \frac{1}{m} \sum_{i=1}^{m} \ell_{L1}(Y^{(i)}, G(X^{(i)})) + \lambda_{adv} \ell_{BCE}(\mathscr{D}(G(X^{(i)})), 1)$$

We then alternate between minimizing $\mathcal{L}(\mathscr{D})$ and minimizing $\mathcal{L}(\mathcal{G})$ with Adam using default parameters. The loss function we are minimizing for the discriminator is the binary cross entropy loss between the dis-

criminator's predictions for each image and the image's true label (1 for real, 0 for fake).

4. Experiments

4.1. How well does the L1 loss work?

We first test the system using only the L1 loss (i.e. excluding the adversarial loss). We use the performance of the model only trained with L1 loss as the baseline.

After running for 120 epochs over the entire dataset, the model had an average training set mean absolute error (MAE) of 0.0103 and validation MAE of 0.0320.

4.1.1. Error analysis

We visually inspected the batch of 64 validation set images with the highest MAE (Figure 2). The majority of errors come from two areas: **Incomplete coloring.** This mostly occurred on shapes that are uncommon in the original dataset (e.g. see the flower petals in the top row), or on shapes at a scale uncommon in the original dataset (e.g. the two circular icons at the bottom left). **Bias towards yellow.** This error type is when the model chooses to color something yellow instead of a different color. For example, in the target images, the paw and food in the dog-food icon are blue, whereas the model chose to color this yellow.

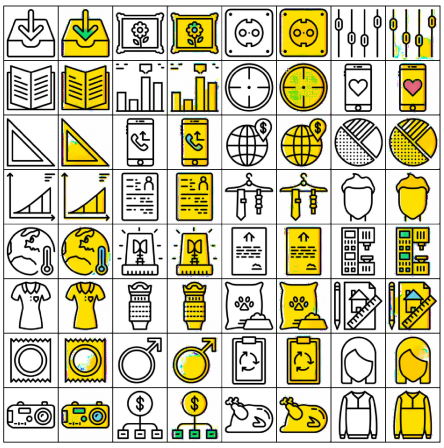

Figure 2. Error analysis on validation batch with highest MAE found two major sources of error.

4.2. How can data augmentation help?

To overcome the incomplete coloring problem, we hypothesized that since our dataset contained images of a fixed scale (i.e. we scaled them to occupy the full 300px x 300px frame in the pre-processing step), the model was unable to color objects of unusual scales (e.g. the large circles in the bottom left of Figure 2). In addition, after we tested the model on completely out-of-sample outline icons and found the results wanting, we also hypothesized that since our icons were mostly all centered and drawn in a certain line width, the model was over-fitting to these nuances of the dataset. Hence, we proposed a data augmentation regime during training that would help our model achieve the following: **Scale invariance.** Randomly re-scale the image to 0.2 - 0.8x its original size. **Position invariance.** Randomly re-positioning the image in the frame. **Outline style invariance.** Randomly blurring or sharpening the input outline. **Noise invariance.** Randomly introduce noisy pixels into the input.

Table 1. Ablative analysis on data augmentation pipeline.

Component	MAE
All Data Augmentation	0.0213
(-) Adding noise	0.0225
(-) Blurring/Sharpening	0.0249
(-) Re-position	0.0272
(-) Re-scaling	0.0320
= Baseline model	0.0320

After introducing the data augmentation regime into our training and training for an additional 50 epochs (i.e. we fine-tuned the weights from the model trained with L1 loss only), we achieved a MAE of 0.0213, a reduction of MAE of 33%. We then carried out ablative analysis to identify the relative contributions of each type of data augmentation, by excluding successive types of data augmentation and re-running each experiment for 25 epochs from the L1-trained model (i.e. without prior data augmentation).

Results. We found that re-scaling the image randomly helped reduce validation MAE by 15%. followed by re-positioning the image which further reduced validation MAE by 7.2%. Blurring and sharpening also reduced MAE by 7.5%. Adding noise reduced the validation MAE by only 3.5%. This was roughly in line with our expectations as the inputs to the validation set are non-noisy. However, we expect this to help with out-of-sample images.

Performance on test set. The difference between the training loss of 0.0103 and our final validation loss of 0.0213 seems large, but qualitative analysis of the output images is also important due to the multi-modal nature of the solution space. Having examined the validation set output images and find them to be substantially improved, we ended our experiments with the L1 model and evaluated the model on our unseen test set. This achieved a test set MAE of 0.0245.

Qualitatively, the results are also visually appealing. Examples of colorization on our test set can be found at our Github page.

Performance on out-of-sample icons. We also run this model on out-of-sample images, and find that the model performs quite well (Figure 3). However, we also note that the model is still heavily biased towards using the color yellow, instead of the darker shades of orange, blue and green also present in the training set. Hence, while it is able to capture certain aspects of the "Yellow" style well (e.g. a white edge on the left of the image, and a darker orange edge on the right) that appear on every icon, it fails to capture the richer color distribution of green and blue that makes the "Yellow" style really pop.

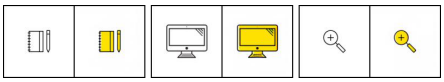

Figure 3. Testing model on out-of-sample outline icons

4.3. Can an adversarial loss help the model overcome the imbalance in training set color statistics?

To address the over-usage of the color yellow by the model, we run an additional experiment using conditional generative adversarial network. We used the pre-trained L1 loss model with data augmentation as the start point for the generator, and train the discriminator and generator simultaneously for 120 epochs.

Results. Quantitatively, the model trained with adversarial loss and L1 loss performs worse in terms of MAE (MAE = 0.0471) than the model trained just to minimize the L1 loss. However, this was largely due to the model choosing to insert the colors green and blue where the target images used the color yellow. Qualitatively, the model trained with adversarial loss is better able to produce icons that look like the original "Yellow dataset. Two examples from an out-of-sample test is reproduced in Figure 4. Note how the model is better able to use the colors blue and shades of orange on both the rugby ball and the media player icon.

5. Limitations and Discussion

One benefit of our system is that colorization is completely automatic, meaning that no user input is necessary beyond the selection of an appropriate "Outline" icon. However, the "Outline" to "Yellow" mapping has some characteristics that make it an easier problem to solve. There is a predominant color hence, after data augmentation, outputs rarely contained odd mixtures of colours or splotches when the model could not make up its mind on what colour to use. When we re-trained the same model architecture (i.e. L1 + adversarial loss + data augmentation) on the "Outline" to "Retro" dataset, the outputs were substantially less visually appealing due to the heterogeneity of color choices in the "Retro" icon set (Figure 5). More specifically, since the "Retro" dataset uses colours somewhat arbitrarily, there are no right answers as to what the color of a particular part of an icon should be given the outline of an icon.

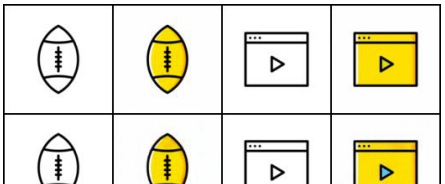

Figure 4. The adversarial model reproduces more faithfully the color distribution of the original dataset. (Upper row) Results using L1 Loss (Bottom row) Results using Adversarial + L1 Loss

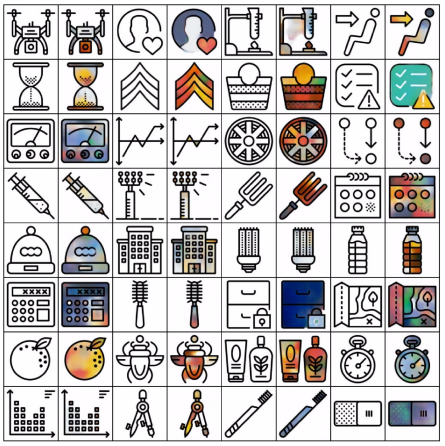

Figure 5. The generator trained with adversarial loss still faces difficulties in the Outline-Retro conversion.

We think that the most promising avenue for future work is in combining the adversarial training regime we have explored in this paper with user hints (Zhang et al., 2017). We can view adversarial loss and user hints as different approaches to solving the multi-model nature of colorization, which is especially acute for icons.

References

Charpiat, G., Hofmann, M., and Schölkopf, B. Automatic image colorization via multimodal predictions. In Forsyth, D., Torr, P., and Zisserman, A. (eds.), *Proceedings of the 10th European Conference on Computer Vision (ECCV 2008)*, pp. 126–139, Marseille, France, 10 2008. Springer. URL `http://eccv2008.inrialpes.fr/`.

Dani, Anat Levin, Lischinski, Dani, and Weiss, Yair. Colorization using optimization. *ACM Transactions on Graphics*, 23:689–694, 2004.

Deshpande, Aditya, Rock, Jason, and Forsyth, David. Learning large-scale automatic image colorization. In *ICCV*, 2015.

Dong, Chao, Loy, Chen Change, He, Kaiming, and Tang, Xiaoou. Image super-resolution using deep convolutional networks. *CoRR*, abs/1501.00092, 2015. URL `http://arxiv.org/abs/1501.00092`.

Frans, Kevin. Outline colorization through tandem adversarial networks. *CoRR*, abs/1704.08834, 2017. URL `http://arxiv.org/abs/1704.08834`.

Gatys, Leon A., Ecker, Alexander S., and Bethge, Matthias. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015. URL `http://arxiv.org/abs/1508.06576`.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative Adversarial Networks. *ArXiv e-prints*, June 2014.

Hayat, Khizar. Super-resolution via deep learning. *CoRR*, abs/1706.09077, 2017. URL `http://arxiv.org/abs/1706.09077`.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL `http://arxiv.org/abs/1502.03167`.

Isola, Phillip, Zhu, Jun-Yan, Zhou, Tinghui, and Efros, Alexei A. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016. URL `http://arxiv.org/abs/1611.07004`.

Johnson, Justin, Alahi, Alexandre, and Li, Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016. URL `http://arxiv.org/abs/1603.08155`.

Larsson, Gustav, Maire, Michael, and Shakhnarovich, Gregory. Learning representations for automatic colorization. In *European Conference on Computer Vision (ECCV)*, 2016.

Liu, Yiming, Cohen, Michael, Uyttendaele, Matt, and Rusinkiewicz, Szymon. AutoStyle: Automatic style transfer from image collections to users' images. *Computer Graphics Forum (Proc. Eurographics Symposium on Rendering)*, 33(4), June 2014.

Networks, Preferred. Paintschainer, 2017.

Ronneberger, Olaf, Fischer, Philipp, and Brox, Thomas. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL `http://arxiv.org/abs/1505.04597`.

Zhang, Richard, Isola, Phillip, and Efros, Alexei A. Colorful image colorization. *CoRR*, abs/1603.08511, 2016. URL `http://arxiv.org/abs/1603.08511`.

Zhang, Richard, Zhu, Jun-Yan, Isola, Phillip, Geng, Xinyang, Lin, Angela S., Yu, Tianhe, and Efros, Alexei A. Real-time user-guided image colorization with learned deep priors. *CoRR*, abs/1705.02999, 2017. URL `http://arxiv.org/abs/1705.02999`.