SMU Data Science Review

Volume 5 | Number 2 Article 10

2021

PokéGAN: P2P (Pet to Pokémon) Stylizer

Michael B. Hedge
Southern Methodist University, mhedge@smu.edu

Morgan Nelson
Southern Methodist University, nelsonms@mail.smu.edu

Thomas Pengilly
Southern Methodist University, tpengilly@mail.smu.edu

Michael Weatherford
Southern Methodist University, mweatherford@mail.smu.edu

1 Introduction

Style transfer, a deep learning technique used to stylize an image's content in the fashion of a reference image, is gaining popularity in various commercial applications, including social media products, gaming, photo, and video editing, and artistic content generation. Advances in style transfer techniques will enable applying these methods to more complex television, film, and gaming problems. One application made possible by these techniques is a pet to Pokémon stylizer.

Pokémon—a franchise spanning television, movies, games, and more— has been wildly popular since its inception in 1998. With a 24-season TV show, 23 movies, 122 video games, and over 30 billion trading cards sold, it is safe to say Pokémon generates tremendous excitement and revenue amongst its large fanbase. The popularity of the franchise, with hundreds of millions of fans, all but guarantees a market for a mobile or web app that can take user-submitted images of pets and transform them into another one of their favorite Pokémon characters.

To effectively accomplish this task, an ideal style-transfer algorithm must preserve the essential source image information, such as the pet's shape and recognizable features, while applying nuanced stylistic differences to the various semantic segments within an image. While dozens of techniques are developed to address this class of problems, this paper focuses on two methods of approach: Neural Style Transfer (NST) and Image-to-Image translation with generative adversarial networks (GANs).

The first method developed to address this problem is the Neural Style Transfer method. The central issue addressed by style transfer algorithms is the separation of an image's content—the objects of interest within the image and their structure—and its style—aspects like color, color gradients, shading, and blending. Supervised style-transfer methods rely on corresponding image-stylized image pairs that allow the style to be inferred from the difference between the object representation in both image domains. However, given the creative nature of cartoon content generation and the lack of stylized image examples for newly created content, unsupervised style-transfer methods will be the most versatile and functional for most applications.

Most current methods build on deep neural network architectures. Convolutional Neural Networks (CNNs) have been a powerful basis for style transfer problems [1,2,3,4]. CNNs effectively extract high-level feature information as well as stylistic information from images. CNNs alone, however, tend to produce images with distorted content, artifacts, and a single style. Much research has been done to reduce distortions and expand output images' complexity, incorporate multiple styles, control color schemes, and blend styles [3,4,5,6]. These rely on semantic image segmentation, object recognition, and various color transformation techniques. Another popular approach is to employ encoder-decoder architectures [7,8]. Encoders and decoders are used to process image content and image style information separately. Images are iteratively restyled until style loss, and content loss is minimized compared to the original photos [8]. These methods are exceptional in their ability to adapt to unseen styles and contents. Still, they require multiple images of a given object in many distinctive styles and multiple objects in a given style to function.

The technique of Transfer Learning was employed to limit the number of images required to transfer style from one image to another successfully. Transfer learning allows for a model to be built by taking an existing, trained neural network and reusing the weights and biases in that network for a different purpose, replacing the final layers of the network with ones trained to the current task [9,10,11,12]. A neural style transfer mechanism can be built with minimal additional training by reusing the weights and biases in the VGG class of trained neural networks.

However, while these methods have succeeded in transferring style, they still have many shortcomings that more recent approaches have overcome. The style-transferred output images created by the NST model were full of random edges in the image background and on the animal itself. These edges resulted from confusing low-level style features with low-level structural features, which the NST model could not distinguish. However, the model did an excellent job transferring the colors, color gradients, and shading found in the Pokémon images. To limit the occurrence of these artefacts, a preprocessing data pipeline was created. Given the application's nature, real-world animal images were used to simulate the images the models are expected to handle in operation. YOLO object detection was used to detect animals within an image and create bounding box coordinates that contain the animal. These coordinates and the corresponding images were fed into OpenCV's GrabCut algorithm to perform image segmentation. The algorithm iteratively learned an image mask, the pixel labels for the image (background, foreground), for each image, and applied the mask to remove the background pixels. The segmentation process successfully segmented roughly 30-40% of animal images without manual intervention. Images that could not be segmented via mask learning were supplemented with manually created masks, which resulted in significantly improved segmentation results. This was too cumbersome to perform for this research. Still, the app could include a feature to allow users to submit their masks for complicated images to prevent segmentation from being a performance bottleneck. The masks generated in this step can be used to remove background artifacts from the output images from style transfer models downstream.

The next stage of the preprocessing pipeline involved an image-to-image translation approach. A modified version of CartoonGAN [13] was used to toonify the full user-submitted picture. The network was modified to include ten residual blocks, stronger content retention, and increased pre-training to allow the GAN to learn and better retain structures within the image. The animals retained their shape and color scheme while being rendered in a Pokémon cartoon-like style. The optimized network successfully stylized most images but tended to introduce artifacts and distort images if the image had a bland color scheme. In output images that were not appealing, the model created vibrant colored artifacts near the edges of the image, overlaid a single-colored filter, or imposed a pixelated grid-like filter. The wide range of effects suggests a lack of convergence on a single style. The masks generated for the images can then be used to segment the toonified pet in downstream image blending models.

Researchers also explored taking two trained StyleGANs and "blending" the two models to transfer both the high-level and low-level features [14]. The concept is to blend the base StyleGAN model with a finely tuned StyleGAN model. A method to interpolate between the two trained StyleGANs weights that can control independently which model you take the high-level weights and low-level weights from in the "blended" model will be utilized. To create Pokémon from a pretrained StyleGAN the

weights will be interpolated between a trained StyleGAN from a Pokémon dataset [14], and a pretrained StyleGAN from Massachusetts Institute of Technology (MIT) researchers [15] on cat faces.

2 Background

2.1 Style Transfer Using Convolution Neural Networks

One of the first applications of neural style transfer (NST) to address stylistic variations within a style-source image was introduced by Gatys et al. [2]. They extracted high-level semantic features from target images using the VGG network, pre-trained object recognition, and localization CNN. They matched them to feature representations within a collection of stylized images, allowing control over which regions of the stylized image are used to style specific target image regions. It also benefited from generalizing across datasets and other visual information processing tasks [2]. The algorithm constrained the style synthesis by feature representations from CNNs, which reduced the style transfer method to an optimization problem. The algorithm performed a pre-image search to match target image features to features within a corpus of example style images based on deep image representations.

The VGG network utilized a feature space described by a normalized version of the 16 convolutional and five pooling layers of the 19-layer VGG network. The normalization was conducted by scaling the weights within layers. The mean activation of each convolutional filter over images and positions was equal to 1, which was found to perform better than max pooling. The representation of image content was done in the following manner. Each layer of filter banks stores the filter response to the image in a matrix of dimensions N (number of filters in the layer) x M (the height times the width of the feature map). The objective function utilized to train the network is a squared-error loss defined within each layer, and the loss is calculated by differencing the original image's feature representations and the generated image within a given layer. The derivative is calculated with respect to activations in each layer, and a gradient is computed using standard error backpropagation. This method reduced image content to a series of high-level features determined by CNNs trained explicitly for object recognition. By contrast, the representation of image style in this method is built upon a feature space defined by the correlation between filter responses in any layer of the network. The expected value is determined within each feature map's spatial extent independently. This had the effect of defining style as the correlation of nearby pixel values within each feature independently. This approach is extended to include feature correlations between multiple layers, which allows the capture of style information excluding global arrangements within the image [2]. Style transfer is accomplished by generating a new white-noise image that matches both the target image's content and the reference image's style as defined by a minimized content and style loss over all layers. The aggregate loss is a composite of loss functions that can be weighted independently to favor content or style in the output image. The most visually appealing

output images were created by matching style features to higher layers in the network. The initialization of all images was done via white noise; however, the initialization could be done with the content image at the cost of biasing the output image towards the content image.

In subsequent work, Gatys et al. extended the method to control key perceptual factors, such as spatial location, color information, and spatial scales [3]. To effect spatial control, a segmented version of each image (consisting of R spatial guidance channels of value 0 or 1, representing foreground and background) is provided as input. Each channel in the target image is given the style that corresponds to the channel of the style image with the same value. In the case of multiple style images, the regions are indexed overall example style images. The first method multiplies each layer's feature maps by R guidance channels to compute one spatially guided Gram Matrix for each R region in the style image. Each of these guided Gram Matrices is then used as the optimization target for each corresponding region in the content image. The second method stacked the guidance channels with the feature maps to spatially guide neural patches, reducing the computation required to one Gram Matrix with R additional channels at the expense of style quality. Color control was implemented using one of two methods: luminance-only transfer and color histogram matching. Luminance-only Transfer extracts luminance channels from the style and content images then apply the NST algorithm to produce and output luminance image. The color information of the content image is then combined with the output luminance image. This method best preserves the color scheme of the content image. Color histogram matching replaces the style image with the colors of the content image, and then performs the NST algorithm unchanged. This method was found to preserve best the color distribution of the style image [3,4]. The main drawbacks of these approaches are that retained color schemes apply to the entire image rather than segments, and output images are prone to having artifacts introduced.

Zhao et al. (2019) introduced a novel method to automatically segment objects in both content and style images and extract their 'soft semantic mask' to reduce the prevalence of artifacts in output images. The soft masks and source images are provided multichannel input to an augmented deep CNN framework incorporating a generative Markov random field model [6]. Soft masks contain the probabilities of an image containing an object. The effect of this approach is to extend the spatial guidance channels developed by Gatys [3] beyond two regions, giving more fine-tuned control over the transferred style, and to allow more robustness in regions where multiple objects are probable. The architecture was based on VGG-19 and incorporated soft semantic masks that were downsampled to the appropriate resolution as input. The model's semantic style transfer optimization was accomplished using a patch-based augmented loss function that penalizes patch matches with inconsistent semantic masks. The style loss function is computed on a matched patch basis, independently optimizing the style transfer for each patch.

In contrast, the content loss function minimizes the squared error using backpropagation with L-BFGS. The main innovation in this approach was semantic image prediction using a CRF-RNN architecture. Semantic probability maps are created by extracting the neural activations before max pooling and scaling them to between 0 and 1. They are then treated as the probabilities that a pixel belongs to each object category. This method was most effective for photographs and experienced

increasingly degraded performance for increasingly stylized images. This method's poor performance regarding stylized images suggests a different semantic segmentation method is necessary for cartoon-style transfer.

2.2 Image-to-Image Translation Using GANs

The Generative Adversarial Network (GAN), developed by Goodfellow, et al. [16], offered an alternative approach to the domain transfer problem. Goodfellow employed two competing neural networks, a generator, and a discriminator, to learn the underlying statistical distribution of image data. The generator network receives a noise vector and produces an output image with an assumed statistical distribution. The generator's outputs are shuffled evenly with images from the training set and fed to the discriminator network. The discriminator classifies the images as real or generated and sends the associated probabilities to a cost function, which uses backpropagation to update the generator or discriminator's weights based on the training phase. To stabilize the network, only one network is trained, and it is trained until its performance has been optimized for a given adversarial state. Once the network has been optimized, the adversary begins its training phase. This process is iterated until the approximated distribution of the generator is sufficiently close to the training data distribution. When built with CNNs this network can learn an image style and content described in terms of statistical distributions.

One such approach by Zhu et al. in 2017 developed an image-to-image translation method (CycleGAN) that did not require paired image inputs. It could perform multiple tasks like collection style transfer, object transfiguration, season transfer, and more [17]. This utilized a GAN to translate content images to the style domain represented by the style-image data. Without image pairs to explicitly learn style, Zhu et al. assume an underlying mapping between the two sets of images that can be known. A mapping G: X --> Y such that G's output is indistinguishable from the images y in style set Y. They found difficulties in optimizing the adversarial objective in isolation, which often led to mode collapse issues. To address this, they imposed cycle-consistency on style transfer. The function that maps content images to stylized images is trained simultaneously with an inverse mapping function (F: Y --> X), and a cycle consistency loss is added to encourage F(G(x)) ≈ x and G(F(y)) ≈ y—constraining the image sets' relationship allowed for convergent training of the unpaired image-to-image style transfer algorithm. The architecture utilizes Dx to distinguish content images from style-to-content domain translated images, and Dy to distinguish style images from stylized-content images. The objective function was defined as the sum of the adversarial loss and cycle consistency loss functions, defined below.

$$L_{GAN}(G, D_y, X, Y) = E_{y \sim Pdata(y)}[\log D_y(y)] + E_{x \sim Pdata(x)}[\log (1 - D_y(G(x))] \quad \text{(eq 1)}$$

$$L_{CYC}(G, F) = E_{y \sim Pdata(y)} \left|\left|\mathrm{G}(\mathrm{F}(\mathrm{y})) - y\right|\right|_1 + E_{x \sim Pdata(x)} [|| F(G(x)) - x||]_1 \quad \text{(eq 2)}$$

$$L(G, F, D_x D_y) = L_{GAN}(G, D_y, X, Y) + L_{GAN}(F, D_x, Y, X) + \lambda L_{cyc}(G, F)$$

(eq 3)

where λ is a weighting factor for the cycle loss, the optimal solution to the problem then becomes:

$$G^*, F^* = \arg\min_{G,F}\max_{D_x D_y} L\left(G, F, D_x, D_y\right)$$

(eq 4)

To stabilize the training procedure, the researchers implemented two techniques. The first was to replace the negative log-likelihood objective (eq 1) with a least-square loss, which is more stable and produces higher quality results.

$$L_{LSGAN}(G, D_y, X, Y) = E_{y \sim Pdata(y)}\left[D_y(y) - 1\right]^2 + E_{x \sim Pdata(x)}\left[\left(D_y\left(G(x)\right)\right)^2\right]$$

(eq 5)

The second implements Shrivastava et al.'s strategy to update the discriminators using a history of generated images rather than those produced by the latest generative networks to reduce model oscillation.

Similar work by Bousmalis et al. [18] imposed a similarity requirement on the GANs input and output images to preserve content from the source image. The generator creates images from both a noise vector and the original content image. The objective function is a weighted min-max argument of two log-loss functions: one for the domain loss and one for a task-specific loss, in their case an object classification SoftMax cross-entropy loss. The addition of a task-specific loss forced the model to produce images that more closely resembled the input image. To reduce content-similarity loss further, they incorporate the image's segmentation into the foreground and background layers and augment the loss function with a penalty for large differences between source and generated images in the foreground only. This grounds the generation process to the original image and stabilizes the minimax optimization [18]. The loss function was a masked pairwise mean squared error to penalize the differences between pairs of pixels [18].

Taigman et al. took a similar approach to unsupervised domain adaptation. Still, they developed a more generalized method that considers a distribution within the target space rather than replicating the style of an image set [19]. Given a source set (S), target set (T), and classification function f, they learned a mapping G: S -> T such that f(x) ~ f(G(x)). The function G is a composition of classifier f and a learned function g. The network utilized a compound loss consisting of an adversarial loss, a classification loss that minimizes ||f(x) - f(G(X))||, and a regularization term that encourages G to be the identity mapping for all x in T [19]. Taigman achieved generalized domain adaptation by inverting the classification function f in the domain T by generating training samples (f(x), x) for x in T and learn a function h from f(T) = {f(x)|x in T} to T. Domain adaptation was used to map f(S) = {f(x)|x in S} to T [19]. The domain transfer portion of their network contains two modifications: the classifier f(x) is the baseline representation sent to the generator function G. During training, the algorithm considers the generated samples G(x) for x ∈ t. The researchers successfully transferred face images to a cartoonized emoji space capable of retaining identifying facial features.

In 2018, Yang Chen et al. developed CartoonGAN to translate images from the real world to a cartoon image space. Instead of learning the intricacies of an image collection's style, they assume that cartoon images have simplified high-level abstractions and clear edges, smooth color shading, and simple textures. Two novel loss functions are employed: semantic content loss and edge-promoting adversarial loss. Initially, the generator begins with a flat convolution stage followed by two down-convolution blocks to compress and encode the images [13]. Useful features are extracted at this stage to be manipulated later. The content and manifold features are constructed from 8 residual blocks. Cartoonized images are produced using two up-convolution blocks. They utilized a patch-level discriminator that classified N x N patches of the output image as real or fake rather than the entire image. To classify the images, the discriminator employed two strided convolutional blocks to reduce image resolution and encode essential features. A feature construction block and convolutional layer are used to classify the image. The loss function is composed of an adversarial loss and a weighted content loss. A standard discriminator network is insufficient because clear edges are a defining feature of cartoons but only a minor fraction of image pixels [13]. To address this, the cartoon image training set is preprocessed to remove clear edges using a standard Canny edge detector, dilation of edge regions, and the application of a Gaussian smoothing function in the edge regions. The adversarial loss is also adjusted to include a third term over the edge-removed images. The edge-promoting adversarial loss function becomes:

$$L_{adv}(G,D) = E_{ci\sim Sdata(c)}[\log(D(c_i)] + E_{ej\sim Sdata(e)}[\log\big(1 - D\big(e_j\big)\big] + E_{pk\sim Sdata(p)}[\log(1 - D(G(p_k)))]$$

(eq. 6)

The discriminator is trained to maximize the probability of assigning the correct label to the generated images G(p), the edge-removed cartoon images (e), and real cartoon images (c). Content loss is prevented by using high-level feature maps from a pre-trained VGG network and is formulated as:

$$L_{con}(G,D) = E_{pi\sim Sdata(p)}[||VGG_l(G(p_i) - VGG_L(p_i)||_1]$$

(eq. 7)

Where l refers to the feature maps of a specific VGG layer, and the loss is defined using the L1 sparse regularization of VGG feature maps between the input and output images. The L1 norm is used due to the major style difference between real-world images and cartoons, which causes the L2 norm to underperform. Their experiments found that the use of the L1 norm was instrumental in the success of the algorithm. The researchers also used a modified initialization phase during training. Initially, the generator reconstructs the content of input images and is trained using only the semantic content loss. This sped up convergence while creating sufficient cartoon-styled images early on. The network was trained on 5,402 real-world photos and thousands of cartoon images from 4 different cartoons, which allowed the GAN to learn multiple styles. The high-quality images produced outperform state-of-the-art techniques using neural style transfer and CycleGAN.

This paper proposes a pipeline for a pet-to-Pokémon stylizing app utilizing a style transfer method. Pictures of pets can be mapped from the real world to a Pokémon-stylized cartoon rendering of the animal. It should handle multiple Pokémon styles and

preserve important input image information such as body structure and fur patterns while allowing for the target image color, gradients, and bold edges to be imparted to the pet within the image. It will build upon the results of CycleGAN and CartoonGAN. The proposed architecture will consist of a Generative Adversarial Network that utilizes a pre-trained VGG-16 network and an image segmentation pipeline to impart the target image style and remove unwanted artifacts.

3 Methods

3.1 Neural Style Transfer

The paper started with performing basic style transfer as a proof of concept. This proof was to show that it was possible to get a photo of a pet and an image of a Pokémon character and blend these two images. The result from the proof of concept was to get output images that were recognizable as a blend of the two input images.

The Pokémon data used for this method was the Pokémon Images Dataset from Kaggle user kvpratama [20]. This consists of 819 images of Pokémon from the first to the third generations. This data was also used in the model blending approach found in section 3.3. The animal images were freely available and downloaded from [21,22,23,24]

This approach started with a standard VGG19 neural network from the python PyTorch library. Input images were reduced to 256 x 256 pixels, and their color values were normalized. The convolutional layers of the VGG19 model were then chosen for their contribution to either style or content, and the remainder of the network was discarded. The content loss was calculated as a simple mean squared error between the content convolutions and the input content image. Style losses were calculated as a normalized Gram matrix between the style layers and the input style image. Style losses were weighed to give preference over the content loss. The training was stopped when the weighted losses were minimized.

3.2 Style Transfer Preprocessing with CartoonGAN

The style transfer results above, found in Figure 2, show that the Pokémon-style transferred to the animal included unwanted artifacts scattered throughout the background and on the animal itself. These artifacts result from the transfer of low-level structural features of the Pokémon, like well-defined edges and borders. To address both background and foreground artefacts, a method was devised by which a toonified version of the pet image could be obtained for blending in the above model. By toonifying the pet image before performing neural style transfer, the low-level structural features of the pet would already be similar to that of the Pokémon, and the neural style model could focus on other aesthetic aspects of the Pokémon such as colors

and color gradients. Image segmentation masks are created for each pet image to prevent background artifacts in the output images, which filter out the background while retaining the pet. These masks can then be used to combine the original pet photo's background with the stylized pet image, effectively converting the pet into a Pokémon in the original image context. The CartoonGAN model developed by researchers Chen et al. was modified and retrained with Pokémon images for this style transfer approach [13].

The process must use real-world images with complex backgrounds and multiple objects since these are the kinds of images the app is expected to receive as input in operation. The source data for this model consisted of mostly real-world animal images, while the target-style data consisted of frames from the Pokémon TV series.

3.2.1 Dataset Description

The target image dataset was manually created using FFmpeg to extract 640x480 pixel frames from each episode in the first two seasons of Pokémon available on the Internet Archive. These images were resized to 256x256 pixels. Early iterations found that the GAN generator learned to fool the discriminator into thinking real-world images were from the TV show by inserting artifacts resembling pokeballs into the image without further stylization. This necessitated the removal of all images containing pokeballs. The final dataset contained 5,706 images.

The source data used was a modified version of the Animals-10 dataset from Kaggle, containing only the 10,207 images of cats, dogs, cows, chickens, and butterflies [25]. The training set included 6,803 images balanced across each animal, and the test set contained 3,404 images. The real-world images in this dataset did not have a standard size and included complex background environments, which required normalization and substantial cleaning. All images were resized to 256x256. The background needed to be removed from the images via image segmentation to prevent artifacts and focus the first model's style transfer on the pet. This was done using YOLOv5s object detection and the OpenCV GrabCut algorithm.

3.2.2 Data Preprocessing

To segment the images with OpenCV's GrabCut algorithm, it was required to create bounding boxes containing each image's animal. This process was automated using YOLOv5s object detection constrained only to classify animals, including birds (chickens), cats, dogs, horses, sheep, cows, elephants, bears, zebras, and giraffes. It is not important that the algorithm correctly classify each animal, only that it detects them and creates the bounding box coordinates. Once the animal was seen, bounding box coordinates were generated and saved for use in the GrabCut algorithm.

Each animal image and its corresponding bounding box coordinates were passed to the GrabCut algorithm, which learned the image mask and outputs the segmented animal. To improve the segmentation performance on complex images, segmentation masks were manually created, which resulted in significantly improved results at the cost of labor-intensive mask creation. Manual mask creation was not explored further

since the bounding box method provided enough cleaned images for the models downstream. The masks created in these steps provide the additional benefit of removing artifacts from style-transferred images by segmenting the stylized animal from any artifacts in the background.

3.2.3 Cartoon Style Transfer

The network employed for cartoon-style transfer was a modified version of CartoonGAN. A brief overview will be given, but a detailed description of the network is beyond the scope of this paper. For a complete discussion of the network, see [13]. A GAN is an architecture that utilizes two competing neural networks. In this case, these are convolutional neural networks (CNNs): a generator and a discriminator. The generator's goal is to create images that resemble the target domain (Pokémon) as determined by the discriminator. The discriminator's goal is to classify images as real or generated accurately. The architecture employed can be seen in figure 1 [13]. The generator and discriminator networks use transfer learning by employing a pre-trained VGG-16 network to extract high-level features from the images. The final layers of the networks are trained using the Pokémon images to learn low-level stylistic features of the Pokémon image domain.

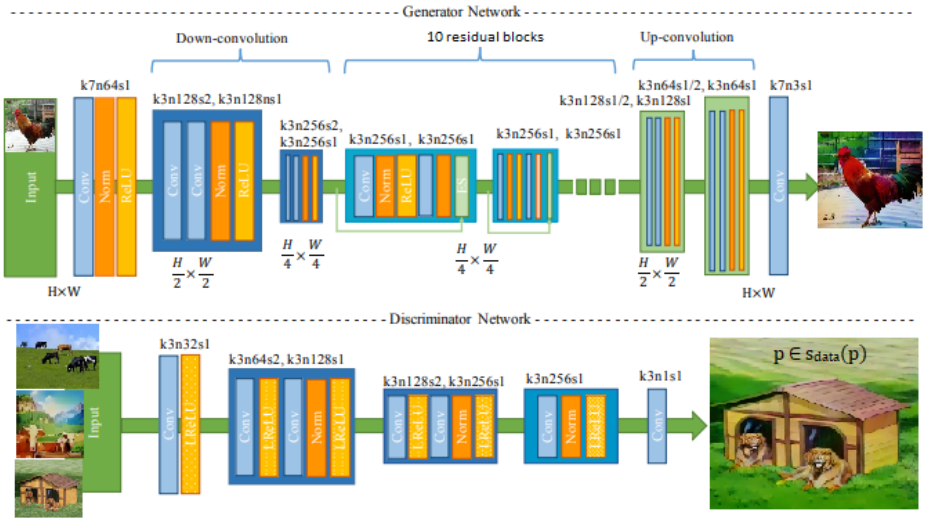


Figure 1: The structure of the modified CartoonGAN network. K represents the kernel size for the filter, n is the number of VGG feature maps, and s is the stride. Image Credit: [13]

The generator flattens and normalizes the images before feeding them to a series of down-convolution filters. After the down-convolution stage, the images are fed through 10 residual blocks, which send the layer output to the next layer and the third layer after itself. These skip connections allow the early layers to learn nearly as fast as the final layers of the deep network. The generator then uses two up-convolution blocks to reconstruct the image in the new style domain.

The discriminator's goal is to classify a given image from the source domain or the target domain based on the low-level style features. Since style is a property of the image at a small scale, a patch-level discriminator reduces the number of train parameters. The discriminator discriminates between real and cartoon images by focusing on local features in image patches rather than the image as a whole, enabling a much shallower discriminator network. A leaky relu loss function is employed after each convolution phase.

The network employs a compound loss function (eq 8) that incorporates a modified adversarial loss (eq. 9) and a content loss (eq. 10) to balance the trade-off between style transfer and content loss. One of the defining characteristics of cartoon images is bold, well-defined edges, but these edges are not prominent; they represent a small fraction of pixels in the image. Because of this, the adversarial loss was modified to incorporate edge-loss penalties, which guide the generator to retain well-defined edges in its output images (eq 9).

$$L(G, D) = L_{adv}(G, D) + \omega L_{con}(G, D) \quad \text{(eq. 8)}$$

$$L_{adv}(G, D) = E_{ci \sim Sdata(c)}[\log(D(c_i)] + E_{ej \sim Sdata(e)}[\log(1 - D(e_j)] + E_{pk \sim Sdata(p)}[\log(1 - D(G(p_k)))] \quad \text{(eq. 9)}$$

$$L_{con}(G, D) = E_{pi \sim Sdata(p)}[||VGG_l(G(p_i) - VGG_L(p_i)||_1] \quad \text{(eq. 10)}$$

The content loss is measured by the VGG-16 network's feature maps and is controlled by the content loss weight, ω, which is tuned to optimize the model's performance. Larger ω's force more image content to be retained, resulting in less style transfer. The initialization stage of the algorithm pre-trains the generator to reconstruct the content of training images using only the content loss portion of the loss function. This speeds up model convergence by enabling the generator to create realistic content before training and focusing on optimizing the adversarial loss responsible for learning the cartoon image's style.

3.3 StyleGAN Model Interpolation

Once the pet has been segmented from the rest of the image and stylized, the image blending may occur. The proposed architecture will use a pre-trained StyleGAN network to map high-level features in the content image and introduces a content loss that compares high-level feature maps from both the content and the target image to ensure the retention of the input image's key features. The resulting blended structure would have the structure of the Pokémon (with some deviations resulting from pet shape in the blending process). In contrast, the color of the blended object would use the colors of the pet as a baseline while imparting new color gradients from the selected Pokémon image. The StyleGAN uses a generator and discriminator in its combative

modeling technique. The question is how we can quantitatively validate the output quality of the images from the StyleGAN. Unlike other pure mathematical models and formulae, the GANs' input and output images cannot be easily compared because the quality of the output is based on how well the created images look to the human eye. It has not yet been possible to recreate the generate human visual interpretation from computer electronics electronically. In 2016, the concept of inception scoring (IS) was introduced. IS is an objectively applied metric to generated images to evaluate their quality to an original image. This method was specifically designed to be used for synthetic images that were the resulting outputs of GANs. IS did not provide optimal or clearly quantifiable results, which led to issues with its application and use. In 2017, a new metric was introduced called the Fréchet inception distance (FID) specifically to assess the quality of images created by generative models such as GANs. Unlike its predecessor, IS, FID evaluates only the distribution of the generated images. FID compares the distribution of the real images that are used to train the generator to the distribution of the generated images. In this way, the FID value is used similarly in the way of a loss function and provides us with a quantitative measure used to provide quantitative proof of the quality of the GAN images without overfitting our model. Similar in behavior to other loss functions, the value of FID will decrease as the GAN is evaluated for the model based on the outputs from, and the FID will reach a minimum number for each run. After this minimum value, the FID value will increase, like other loss functions, which is a sign of model overfitting.

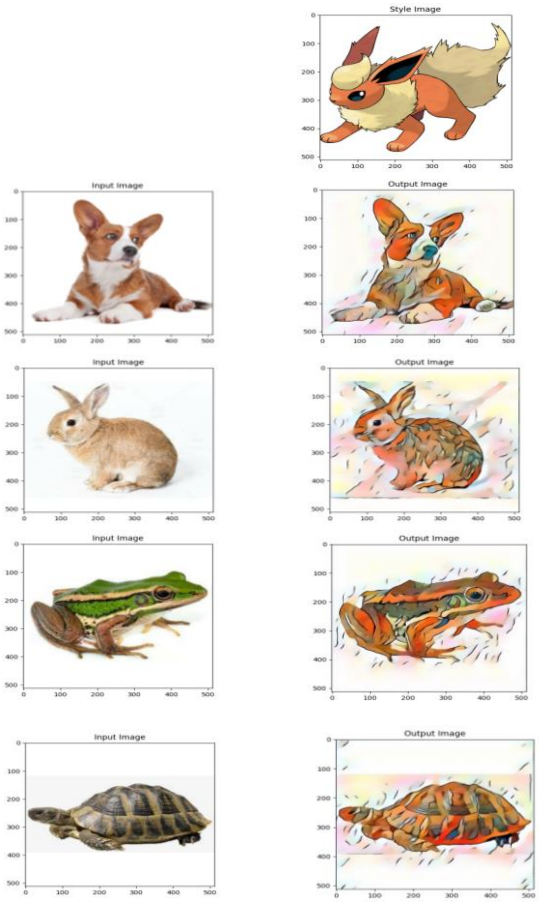
4 Results

4.1 Initial Style Transfer

The results of the initial investigation into neural style transfer are shown in figure 2 below. The images on the left are the source content images, with the source style image on the upper right. Output images for the combination of content and style input are shown in the lower portion of the right-hand column. With varying degrees of success, the neural style transfer gave impressive results, with some caveats.

The sweet spot for training this network appears to be around the 300-400 iteration mark. Fewer iterations led to poor image quality, while more than that number of iterations caused the loss to behave erratically, sometimes increasing exponentially. Further research is warranted to improve the loss calculation to see if it can get a more consistent result.

The output images retain most of their original content. The dog still looks like a dog, and the frog is still a frog. The style of the input image also transferred well. The overall look and feel of the input image were modified to appear to be more like the input style image. Still, there is evidence of visual artifacts outside of the original content of the image. The worst example of this is the final tortoise image. In the input image, there is a discernible but exceptionally light gray background that shows up in



the output image as a colored band horizontally centered on the tortoise. In this image, the network noticed the gray band in the input image and attempted to transfer the style. This is an undesirable result to human viewers but proved too difficult to remove in the research period. For this reason, image segmentation to separate the content from the image background was attempted as a corrective action.

Figure 2: These are the original images of pets: dog, rabbit, frog, and turtle in the left-hand column. The Pokémon Flareon atop the right-hand column above the blended images of the Flare-dog, Flare-rabbit, Flare-frog, and Flare-tortoise.

4.2 Style Transfer Preprocessing with CartoonGAN

Image Segmentation Results

The image segmentation pipeline was tuned to provide the highest number of well-segmented images while maintaining a low-performance cost. The pre-trained object detection pipeline required a detection confidence threshold for its classifications. A confidence level of 0.3 was found to classify multiple animals in a single image accurately. This was unnecessary for the initial stages of the application. The concept needs to be proven possible first; the additional animals in each image would only complicate the style transfer process. A confidence threshold of 0.6 resulted in the most prominent animal in each image being correctly classified/detected and was used as the final threshold value.

The GrabCut algorithm was tuned to learn image masks over 25 iterations, balancing processing time and segmentation quality. Iterations above this value did not significantly improve segmentation quality, while iterations below 20 could result in poor segmentation results for mildly complex images. This resulted in roughly 30-40% of images being sufficiently segmented for downstream neural style transfer use. These are images in which most of the background has been removed, with a minimal amount of the foreground object (pet) being removed from the image. The images that were not successfully segmented were of two types. The first is complex images with large variations in color and color gradients, multiple objects, or inconsistent backgrounds. The second type are those images in which the animal is very prominent and takes up most of the image space. In these images the bounding box coordinates include the majority of the image, which creates a lack of background pixel labels for the GrabCut algorithm to train on, resulting in poor segmentation. Image masks were manually created for a sample of complex images. They were found to dramatically improve segmentation results at the cost of time-consuming mask creation on an image-by-image basis.

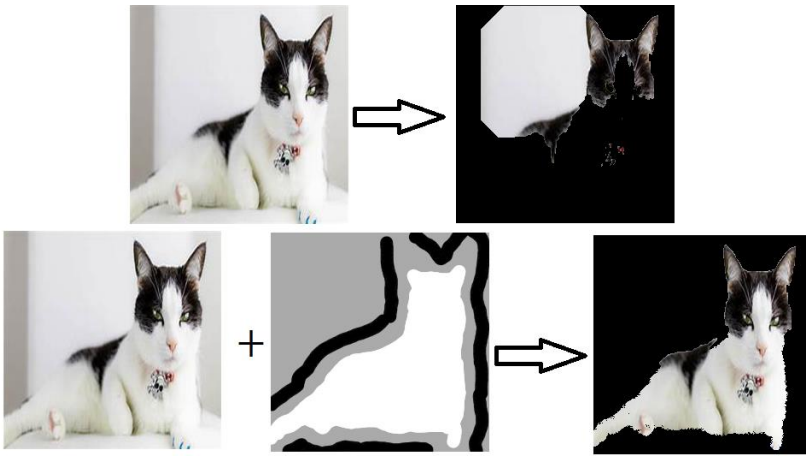


Figure 3: An example image that did not segment well due to lack of background pixel labels. The manual creation of a mask (bottom middle) dramatically improved the segmentation results.

Cartoon Style Transfer

Since the stylistic features of many anime cartoons are very similar, modifications to the generator and discriminator networks were not considered necessary. The network architecture was modified to improve the effect of retraining on the network layers and fine-tune the process to learn the Pokémon style specifically.

The default network parameters caused the generator loss to stagnate early in the training process, and output images contained multiple artifacts. Artifacts consisting of pixelated overlays and vibrantly colored patches of patterns were found in select images indicating the generator would strategically sacrifice image structure to impose a style. The size and shape of the artifacts indicate that the generator had learned to trick the patch-based discriminator by applying enough stylistic features to regions with little relevant structure (edges, corners, etc.), allowing the discriminator to verify the original structure and the new style in output images.

Figure 4: Results with default parameters. The generator strategically placed patches of stylized pixels to trick the discriminator without stylizing the whole image.

To address the artifacts the generator was pre-trained for 20 epochs instead of 10 to ensure it would preserve more image structure. This dramatically improved the style transfer quality and reduced the number of artifacts but did not completely remove them. Interestingly, in some network configurations, the generator learned to insert Pokeball artifacts within the image without further stylization to trick the discriminator into thinking the photo was from the Pokémon TV show (figure 5).

Figure 5: These images were classified as being from the Pokémon TV show, even without stylization, because the generator placed Pokeballs in the images.

In all of the experiments, there was a wide variety of stylistic effects applied to the images that appeared to be dependent on the animal in the image and the color scheme of the input image. While many images were stylized well, chicken images tended to have more artifacts and little-to-no style transferred. Images with little color variation or a lack of bright blue or green tended to get a blue, green, or sometimes yellow filter across the whole image. This may be the result of the Pokémon TV shows most prominent and frequent colors being green and blue.

Figure 6: Example images showing the input image (left) and the stylized image (right). The variety of styles applied suggests a lack of convergence.

All models produced several well-stylized images and differed only in the type of artifacts introduced and their frequency in output images. Butterfly images were consistently stylized without issue. Cow and cat images were also stylized well in most models. This suggests that the model is learning the appropriate style, but that variations in color schemes in the Pokémon TV show might represent multiple styles. There may be too much variation in the features of the animal images to apply the style to with consistent results. The best model, defined by producing the fewest images with artifacts, had ten residual network blocks, 25 generator pre-training epochs, and a

content loss weight of 8. The resulting images from this network can be seen in Figure 7 below.

Figure 7: Output images resulting from the optimum network after 100 training epochs.

Segmentation of Toonified Animals

The segmentation of the resulting toonified animals with their corresponding mask was not achieved due to data anonymization within the PyTorch data loader and time constraints. This can be achieved by developing a custom data loader to better track the images throughout the style transfer process. These segmented images can then be passed to a downstream neural style transfer model for a final style transfer.

4.3 StyleGAN Model Interpolation Results

Pokémon StyleGAN

Using the PyTorch ada StyleGAN2 implementation, a StyleGAN was trained using the Pokémon dataset of all 819 Pokémon of size 256 x 256. Data augmentation was utilized to increase the number of training images. Figure 8 highlights the first generated fake Pokémon. At this stage, the StyleGAN is learning basic structures like edges. After training for 10,000 kimg on a single Graphics Processing Unit (GPU) Figure 9 shows the StyleGAN has learned the basic structure of a Pokémon. The lowest FID score achieved was 56. While this is not a good FID score, when considering the variability in the dataset compared to a human face StyleGAN it was believed it would

be sufficient to blend with a much higher tuned model. Training was stopped early of the 20,000 kimg goal due to not converging and fears of overtraining the StyleGAN.

StyleGAN Interpolation

The trained PokeGAN and a pretrained StyleGAN from MIT on cat faces were "blended" together by interpolating between the weights of the two models. Figure 10 shows the results of the blended model where the low-level features extend from the PokeGAN model and the high-level features extend from the Cat Face model. Conversely, Figure 11 shows the results where the low-level features extend from the Cat Face model and the high-level features extend from the PokeGAN model. Each blend model was reproduced using the square resolutions of 8, 16, 32, 64, 128, 256.

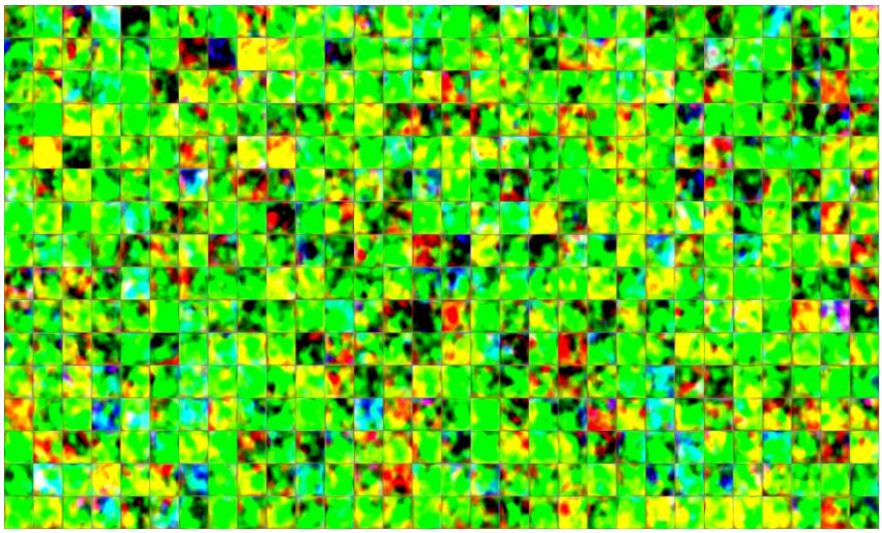


Figure 8: First Pokémon StyleGAN generated fakes

Figure 9: Last Pokémon StyleGAN generated fakes

Figure 10: StyleGAN blending with low-level features from PokeGAN model

Figure 11: StyleGAN blending with low-level features from Cat Face model

Table 1: FID evaluation metric values

{"results": {"fid50k_full": 161.53432660965208},
{"results": {"fid50k_full": 159.50613867223507},
{"results": {"fid50k_full": 137.20510298680932},
{"results": {"fid50k_full": 113.66827320351852},
{"results": {"fid50k_full": 105.11048855403487},
{"results": {"fid50k_full": 93.69253234441443},
{"results": {"fid50k_full": 97.46586409441375},
{"results": {"fid50k_full": 85.46439140488454},
{"results": {"fid50k_full": 80.83827854413724},
{"results": {"fid50k_full": 68.4378085460448},
{"results": {"fid50k_full": 70.96046685942804},
{"results": {"fid50k_full": 70.54898323063446},
{"results": {"fid50k_full": 59.103657947644564},
{"results": {"fid50k_full": 69.44869986205205},
{"results": {"fid50k_full": 58.59892177103728},
{"results": {"fid50k_full": 61.49667888137908},
{"results": {"fid50k_full": 56.280171485416844},
{"results": {"fid50k_full": 64.26374343747938},
{"results": {"fid50k_full": 71.56613421727334},
{"results": {"fid50k_full": 70.172117817697},
{"results": {"fid50k_full": 77.96363065281446},
{"results": {"fid50k_full": 93.60860238782399},
{"results": {"fid50k_full": 85.44940605597489},
{"results": {"fid50k_full": 95.43020930879177},
{"results": {"fid50k_full": 88.56208719211827},

The arrow in the table shows the minimum FID value for this StyleGAN run and designates which model will be used as starting image for next run.

5 Discussion

This research paper aimed to determine how pictures of animals could be transformed into Pokémon-like creatures. The popularity of Pokémon and the massive revenue made by the franchise suggests there is a market for an app that can do just that. In itself, this is an ambiguous problem with many potential solutions. The approaches taken in this paper showed promise, but further research must be done to obtain high-quality style-transferred images if these are to be implemented in a commercial app.

One of the exciting results of the experimentation in this study was the clever tricks employed by some generator networks within the modified CartoonGAN network. In some iterations, these networks learned to create an artifact resembling a pokeball and

to insert it into the original image with no further style transfer applied. This was enough to trick the discriminator into classifying the image as from the Pokémon TV show, given the frequency of Pokeballs in the TV show images.

Several unexpected challenges were found to complicate the quality of output images and prevented the timely improvement of the neural style transfer approach. The first of these challenges was the image segmentation pipeline. It could segment simple images without any issues. Still, complex images with multiple objects or backgrounds and foregrounds with large color variations consistently led to poor input data for our style transfer mechanism. Any style transfer app will be limited by the quality of the image segmentation pipeline. The app could utilize the current preprocessing pipeline if it includes a feature that allows users to submit their masks for each image if they desire. Another option is to look at some of the more state-of-the-art deep segmentation models at the cost of increased computation times.

Another challenge that prevented the application of segmentation masks to our results was anonymizing image data within PyTorch's base data loader. This also prevented the segmentation of the toonified animal images created with CartoonGAN, which were to be blended with Pokémon images in the NST model. Further work is necessary to create a custom data loader that can track its source data and use that information to create corresponding files for additional output.

The training of a StyleGAN presented three main difficulties. The first challenge is the amount of computing power and time it takes to train a StyleGAN. The second challenge is the low amount of training images. Even with data augmentation can cause concern, most StyleGANs are trained with 10,000 images or more. The third challenge is the amount of variability in Pokémon. While human faces are all different, the structure is similar, I.e., two eyes, one mouth, two ears. Conversely, all Pokémon are very structurally different, making it exceedingly difficult for the StyleGAN to converge. The PokeGAN, at the end of the training, because it ran out of options to reduce the FID score, began to change the color of the entire fake dataset, which is an indicator of overfitting. Since the PokeGAN was not able to converge with a low FID score, the model blending was reduced. The interpolation between the two models between the PokeGAN and the cat face model did not produce quality results at any of the resolutions, which reflects the variability of Pokémon and the animals combined with the PokeGANs inability to converge.

The next step to further this research would be to incorporate a softening edge algorithm, similar to that employed in CartoonGAN, within the NST approach. Softening the edges of the Pokémon images could significantly reduce the number and prominence of edges distributed throughout the stye-transferred output image. It would allow the networks to focus more solely on patterns, colors, and color gradients.

Ethical Concerns

Converting pictures of pets into Pokémon does not handle as sensitive of data as other industries such as healthcare. However, data ethics still applies and requires discussion. Ownership of the supplied pictures of pets may include pictures of people and the pets' owners inadvertently. This can be addressed by not storing supplied images and using image segmentation to isolate the pet from any other portion of the image to ensure the model is only applied to a pet. Another ethical question is whether using a trademarked brand like Pokémon for monetary purposes infringes on trademark

rights. This would have to be solved through private contract law with the Pokémon trademark and brand owners. If the application including personal information to receive the results of the model, i.e., name, address, email address, payment, and billing address, it would be pertinent to have transparency to the consumer of how their personally identifiable information will be stored, what it will be used for, and how it will be protected from theft.

6 Conclusion

The research conducted in this paper laid the groundwork for a viable commercial application. It demonstrated that neural style transfer is capable of achieving what was set out to be accomplished. However, there are still significant improvements to be made. The approaches in this paper set out to improve upon the quality of the Pokémon-styled images produced by the initial neural style transfer model, which accomplished the sort of style transfer that was desired but had many artifacts in the output images. The artifacts in the background of the images can be easily removed with the use of image masks developed in the preprocessing pipeline. The main problem is to remove the artifacts from the animal in the styled images from the NST model. The model distributes edges and borders from the Pokémon images indiscriminately throughout the styled output images. Further work should consider techniques to soften the edges of the source Pokémon images. The generator will not learn to consider bold, well-defined edges as a defining feature of the style of the source images.

References

1. Jing, Y., Yang, Y., Feng, Z., Ye, J., Yu, Y., & Song, M. (2020). Neural Style Transfer: A Review. IEEE Transactions on Visualization and Computer Graphics, 26(11), 3365–3385. https://doi.org/10.1109/TVCG.2019.2921336
2. Gatys, L. A., Ecker, A. S., & Bethge, M. (2016). Image Style Transfer Using Convolutional Neural Networks. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2414–2423. https://doi.org/10.1109/CVPR.2016.265
3. Gatys, Leon A., Ecker, A. S., Bethge, M., Hertzmann, A., & Shechtman, E. (2017). Controlling Perceptual Factors in Neural Style Transfer. ArXiv:1611.07865 [Cs]. http://arxiv.org/abs/1611.07865
4. Gatys, Leon A., Bethge, M., Hertzmann, A., & Shechtman, E. (2016). Preserving Color in Neural Artistic Style Transfer. ArXiv:1606.05897 [Cs]. http://arxiv.org/abs/1606.05897
5. Yin, R. (2016). Content Aware Neural Style Transfer. ArXiv:1601.04568 [Cs]. http://arxiv.org/abs/1601.04568
6. Baldonado, M., Chang, C.-C. K., Gravano, L., & Paepcke, A. (1997). The Stanford Zhao, H.-H., Rosin, P. L., Lai, Y.-K., & Wang, Y.-N. (2020). Automatic semantic style transfer using deep convolutional neural networks and soft masks. The Visual Computer, 36(7), 1307–1324. https://doi.org/10.1007/s00371-019-01726-2
7. Zhang, Y., Zhang, Y., Cai, W., & Chang, J. (2018). Separating Style and Content for Generalized Style Transfer. ArXiv:1711.06454 [Cs]. http://arxiv.org/abs/1711.06454

8. Kotovenko, D., Sanakoyeu, A., Lang, S., & Ommer, B. (2019). Content and Style Disentanglement for Artistic Style Transfer. 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 4421–4430. https://doi.org/10.1109/ICCV.2019.00452
9. Zeiler, M. D., & Fergus, R. (n.d.). Visualizing and Understanding Convolutional Networks. 818–833. https://doi.org/10.1007/978-3-319-10590-1_53
10. Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., & Liu, C. (2018). A Survey on Deep Transfer Learning. 270–279. https://doi.org/10.1007/978-3-030-01424-7_27
11. Zhu, W., Braun, B., Chiang, L. H., & Romagnoli, J. A. (2021). Investigation of transfer learning for image classification and impact on training sample size. Chemometrics and Intelligent Laboratory Systems, 211, 104269. https://doi.org/10.1016/j.chemolab.2021.104269
12. Wei, W., Huerta, E. A., Whitmore, B. C., Lee, J. C., Hannon, S., Chandar, R., Dale, D. A., Larson, K. L., Thilker, D. A., Ubeda, L., Boquien, M., Chevance, M., Kruijssen, J. M. D., Schruba, A., Blanc, G. A., & Congiu, E. (2020). Deep transfer learning for star cluster classification: I. application to the PHANGS–HST survey. Monthly Notices of the Royal Astronomical Society, 493(3), 3178–3193. https://doi.org/10.1093/mnras/staa325
13. Chen, Y., Lai, Y.-K., & Liu, Y.-J. (2018). CartoonGAN: Generative Adversarial Networks for Photo Cartoonization. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 9465–9474. https://doi.org/10.1109/CVPR.2018.00986
14. Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Change Loy, Yu Qiao, & Xiaoou Tang. (2018). ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks.
15. Zhao, S., Liu, Z., Lin, J., Zhu, J.-Y., & Han, S. (2020). *Differentiable Augmentation for Data-Efficient GAN Training*. https://hanlab.mit.edu/projects/data-efficient-gans/models/stylegan2-AnimalFace-cat.pkl.
16. Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative Adversarial Networks. ArXiv:1406.2661 [Cs, Stat]. http://arxiv.org/abs/1406.2661
17. Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. 2017 IEEE International Conference on Computer Vision (ICCV), 2242–2251. https://doi.org/10.1109/ICCV.2017.244
18. Bousmalis, K., Silberman, N., Dohan, D., Erhan, D., & Krishnan, D. (2017). Unsupervised Pixel-Level Domain Adaptation with Generative Adversarial Networks. ArXiv:1612.05424 [Cs]. http://arxiv.org/abs/1612.05424
19. Taigman, Y., Polyak, A., & Wolf, L. (2016). Unsupervised Cross-Domain Image Generation. ArXiv:1611.02200 [Cs]. http://arxiv.org/abs/1611.02200
20. Kvpratama. (2017, December 11–10). Pokemon Images Dataset (Version 2) [Dataset]. Kaggle. https://www.kaggle.com/kvpratama/pokemon-images-dataset
21. Wallpapers-all.com. https://wallpapers-all.com/uploads/posts/2016-11/4_dog.jpg
22. Yakham, S. *frog* [Photo]. Shutterstock. https://www.shutterstock.com/image-photo/frog-186253451
23. CuteWallpaper. https://cutewallpaper.org/download.php?file=/21/animals-with-white-background/Animal-pets-white-background-animals-rabbit-pet-.jpg
24. XPixel. *Turtle isolated on white background testudo hermanni, (Herman's Tortoise)* [Photo]. Shutterstock. https://www.shutterstock.com/image-photo/turtle-isolated-on-white-background-testudo-78129739
25. Alessio, C. (2019, December 12). Animals-10 (Version 2) [Dataset]. Kaggle. https://www.kaggle.com/alessiocorrado99/animals10
26. Wang, X., & Gupta, A. (2016). Generative Image Modeling Using Style and Structure Adversarial Networks. In B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), Computer Vision – ECCV 2016 (Vol. 9908, pp. 318–335). Springer International Publishing. https://doi.org/10.1007/978-3-319-46493-0_20

27. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. ArXiv:1706.03762 [Cs]. http://arxiv.org/abs/1706.03762
28. Emami, H., Aliabadi, M. M., Dong, M., & Chinnam, R. B. (2021). SPA-GAN: Spatial Attention GAN for Image-to-Image Translation. IEEE Transactions on Multimedia, 23, 391–401. https://doi.org/10.1109/TMM.2020.2975961
29. Digital Library metadata architecture. International Journal on Digital Libraries, 1(2), 108–121. https://doi.org/10.1007/s007990050008
30. Bruce, K. B., Cardelli, L., & Pierce, B. C. (1999). Comparing Object Encodings. Information and Computation, 155(1–2), 108–133. https://doi.org/10.1006/inco.1999.2829
31. Chen, D., Yuan, L., Liao, J., Yu, N., & Hua, G. (2017). StyleBank: An Explicit Representation for Neural Image Style Transfer. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2770–2779. https://doi.org/10.1109/CVPR.2017.296
32. Karras, T., Laine, S., & Aila, T. (2019). A Style-Based Generator Architecture for Generative Adversarial Networks. ArXiv:1812.04948 [Cs, Stat]. http://arxiv.org/abs/1812.04948
33. Park, D. Y., & Lee, K. H. (2019). Arbitrary Style Transfer With Style-Attentional Networks. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 5873–5881. https://doi.org/10.1109/CVPR.2019.00603