DeepGIF: Video Stylization with Mask Tracking

Description of code given in README.md (in ZIP). Also
`https://github.com/yashpatel5400/DeepGIF`

Jason S., Yash P., Richard D.

1 Introduction

Image style transfer in the form of neural networks came on the scene slightly over a year ago, ushering in a unique method of generating images [1]. Image transfer was a natural extension from the already developed texture transfer problem, as described in Gatys' original paper, though such original implementations did not rely on neural networks. Such stylizing algorithms have become quite sophisticated and efficient since Gatys' original conception. In particular, Fei Fei Li et. al in [2] achieved results with the same degree of precision as those produced by Gatys, in several orders of magnitude less time. Clearly, despite the initial success of the style transfer algorithm, much was to be refined.

In that line, the method we discuss herein proposes a further refinement on the algorithm. Unlike the aforementioned contribution of speedup, ours differs in that it expands the applicable domain of the style transfer algorithm. Specifically, our "Video Stylization with Mask Tracking" algorithm allows users to specify styles on the first frame of an image, from which all subsequent frames of the video are stylized appropriately. While this reduces to the same result if only a single style is applied over the video, it expands the horizon to include allowing multiple styles to be applied across a video.

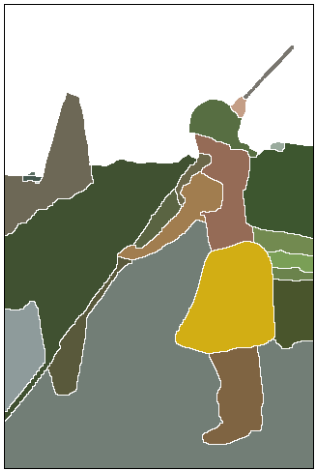

Figure 1: More explicitly, often users only wish to style a subset of an image or style different parts different, i.e. in the figure above, style the helmet with "Starry Night" and the background with "The Scream" and retain the rest. [10]

While masks have been implemented in the GitHub community for style transfer, to the best of our knowledge they have been limited to styling a single image at most into at most two different manners (often isolating the foreground/background). Thus, the first novel development we propose here is creating a manner by which multiple styles can be masked onto a single image. To do so, we implemented a neural network segmentation scheme of images. Though many of the direct segmentations implementations turned out to be fruitless, as more fully expounded in 2.1, by complementing edge-segmentation and connected component algorithms, sufficiently refined results were produced.

In addition to these multi-stylizable masks, however, the main repercussion of our implementation is in its applications to video (and thus, GIF) stylization. While style transfer has been applied to images, its applications to videos has been hereto limited to simply treating a video as a stack of independent frames. By incorporating tracking methods (where we assume the video is a smooth, continuous shot), the coherence of styles from masks frame to frame is further enhanced, as is apparent in the final results we present.

2 Method

The methods below describe how the style transfer and masking was implementation, where the former was simply a reimplementation of that presented in previous papers and the latter from a combination of novel methods and papers. Herein are also described some of the routes investigated that were attempted but did not result in proper results, more so the case for segmentation.

2.1 Style Transfer

Since the style transfer algorithm has been greatly developed and investigated in the past, the version we applied is fully based on the results of [1] and [2]. Note that, since these are implemented identically to how they were originally presented in their papers, only summaries of the relevant method sections are provided, where they may be more thoroughly described in their respective sources.

2.1.1 Slow Style Transfer

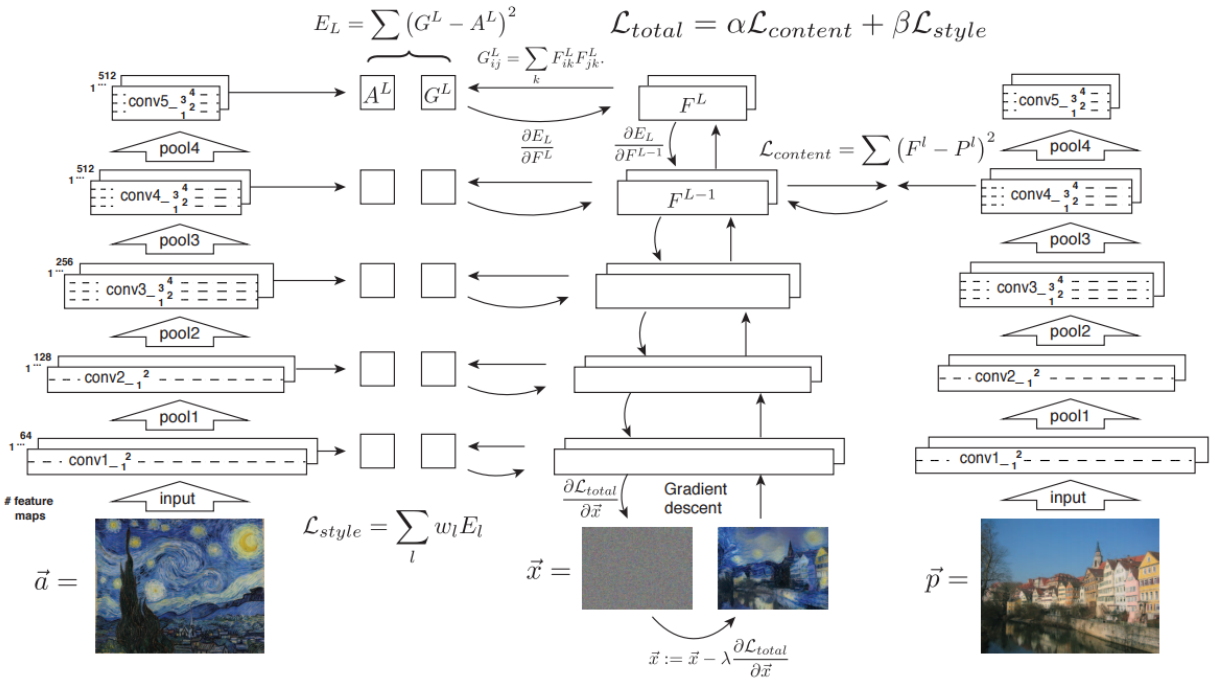

Figure 2: Above is the diagram given in [1], which fully describes the loss layers in their correspondence with the standard VGG outputs of processing the input image.

We use a 19-layer VGG neural network, trained on the ImageNet database, namely with weights corresponding to a network used for classification. This conceptually allows us to exploit the intermediate layers to extract content features from the input image. In other words, by using a network trained for ImageNet, the intermediate layers will have embedded in it the ability to identify what is "captured" in an image, since only by extracting this information is such a network able to identify that different images correspond to the same class. For example, VGG is only capable of classifying pictures two different dog species both as "dog" because its intermediate layers identify that there are features in both images' contents that are typically

associated with features of a dog. Thus, in taking such layers, we can reasonably precisely identify the contents of any inputted image. Denoting (as in the original paper) F^l to be the lth layer of the input/base content image and P^l to be that of the image being molded (i.e. our final result), we can define a content loss function to be:

$$L_{content} = \frac{1}{2}\sum_{i,j}(F_{ij}^l - P_{ij}^l)^2$$

This is simply a pixel-wise mean square error difference between the contents captured in the generated image and input image. Since we assume the higher layers of a neural network capture more "high-level" features of the image contents, we choose to only use $l = 3$ in our implementation of the algorithm. Thus, in attempting to minimize this sum, we develop an image that presumably captures the same contents as those captured in the initial image.

In addition, in a similar sense, we develop a style loss function. Clearly, from Gatys, we wish to develop a style loss that "captures its texture information but not the global arrangement." In other words, we wish to extract a quantity that captures how an image "looks" but without regards to its *global* spatial placement, i.e. determine how to capture the style of Starry Night regardless of the stars being to the left or right of the moon. In line with this, Gatys proposed the Gram matrix, which captures feature correlations and is, thus, not determined or influenced by global spatial properties. This matrix is numerically defined (using the identical notation as above) by:

$$G_{ij}^l = \sum_k (F_{ik}^l F_{jk}^l)^2$$

We wish to similarly use this to define a loss function such that its minimization causes the final image to have a Gram matrix similar to that of the original style image. Unlike the content image, however, where only the "higher features" are desirable (i.e. the low-level ones only correspond to individual or small packets of pixels), all intermediate layers of the style contain useful information as to what constitutes its "style." Thus, for a given layer l, we define:

$$S_l = \frac{1}{4|P_l^2 S_l^2|}\sum_{i,j}(G_{S,ij}^l - G_{P,ij}^l)^2$$

Where $|P_l|, |S_l|$ corresponds to the respective sizes (number of filters) of the given layer l, P_l is the layer of the image being molded, S_l the desired final style image, and $G_{S,ij}$ and $G_{P,ij}$ respectively the (i,j) entry of the Gram matrix of S_l and P_l. As described previously, the overall style loss is (unlike the content) contingent upon all its intermediate layers, meaning the overall style loss is simply:

$$L_{style} = \sum_l w_l S_l$$

Where the set of $\{w_l\}$ is a set of hyperparamters that determine the weighting of the various intermediate style layers. Finally, despite these being the only two pieces of the loss function described in the paper, many implementations developed online had an additional loss, which we refer to as the "coherence loss." Clearly, in using a loss function with solely two methods described above, we can obtain final results that roughly correspond to a mixture of the content imbued with the desired style. However, there was much pixelation in our initial experiments, specifically caused by the artifacts present in the higher levels of the content intermediate layers. In fact, this was wholly apparent in Gatys' original paper, as seen in 3:

Thus, to avoid this local dissonance, we have an additional localized loss function, which creates local coherence in the generated image. Namely, by ensuring nearby pixels have values that do not differ greatly from one another, we can ensure the image looks roughly like a "continuous" image, as we would expect. This is roughly akin to the Deep Learning problem of achieving super resolution (i.e. enhancing the resolution of blurry images), as described in [3]. Per their results, ℓ_2 error norm produced clean results, which is why

Figure 3: Above is a subset of the diagram given in [1], namely where the image clearly has random noise as a result of irrelevant behavior in the neurons on higher layers

we used it herein. Thus, we define the coherence loss to be (where w and h are respectively the width and height of the image):

$$L_{coherence} = \sum_{\substack{i,j \\ i \neq w, j \neq h}} \left((G_{i,j} - G_{i+1,j})^2 + (G_{i,j} - G_{i,j+1})^2 \right)$$

Thus, to combine the above, we define an overall loss function, which we seek to minimize using the standard mini-batch gradient descent scheme, namely with:

$$L_{total} = w_{content} L_{content} + w_{style} L_{style} + w_{coherence} L_{coherence}$$

Where $w_{content}, w_{style}$, and $w_{coherence}$ are hyperparameters we tuned that correspond roughly to how much we wish to weight the image content and their style.

2.1.2 Fast Style Transfer

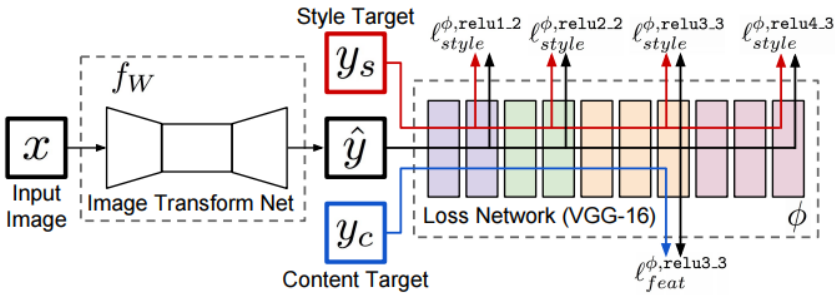

Figure 4: Above is the diagram given in [2], describing the network. Notice how, unlike the implementation of the standard neural style transfer, there is no backpropagation through the VGG network.

In contrast, Wei Wei observed in [2] that much of the above work was, while producing incredible results, unnecessary for the extent of results we desire. Namely, their insight was that the VGG network that was used (pretrained for image classification) already has the capacity to measure semantic and stylistic differences. Thus, we could train a VGG network on a particular style image and fix it for future runs, when we wish to transfer different content images. In other words, we can simply use this entire style-pretrained network as a *fixed* "loss network" ([2]). In doing so, we no longer have to perform optimization with back

propagation on VGG and therefore avoid much unnecessary optimization computation. We, therefore, use this version to produce results, given the significant boost in speed. Because of how this model is formulated, however, each style must have a corresponding pre-trained loss network. This is clearly in contrast to the original implementation, where no training data was required and instead each run was essentially run from a clean slate. Thus, rather than having a moderately slow transfer as in Gatys' implementation, you train a loss network (expensive) and can subsequently feed any desired content image (super-cheap). As each such networks requires significant training time, however, we made use of publicly-available pre-trained models for our purposes.

2.2 Segmentation

Unlike the previous section, there were no pre-built networks that directly completed our desired segmentation objective to the best of our knowledge. Specifically, the end result of segmentation was to automatically produce an area-based segmentation of an input image, i.e. output an image similar to Figure 1. While there is much ongoing semantic segmentation research, such was not directly pertinent, as the semantic predictions were of no relevance herein. The main dissonance between semantic segmentation applications and our segmentation was that the former must be trained on particular objects and thus, have very limited scope of input images. For example, SegNet (developed through [4]) has become a wildly popular network in the scope of autonomous vehicles. While such is often useful for domains where the scope is largely pre-determined (i.e. autonomous vehicles), the larger scope of input images considered herein was clearly far too large to be allowable. Nevertheless, we tried out such networks to see if their input training images were varied enough to at least be able to segment the input images, shown fully in Section 3.2.
In turn, we decided to pursue implementations that make use of edge segmentation (i.e. transforming an image into just its edges) and subsequently doing thresholding and connected components analysis:

2.2.1 Edge Segmentation

The final objective of this portion of the pipeline was to produce an edge segmentation of any given input image, where items in the image that humans would consider "separate" are separated by edges on all sides. While what humans consider "separate" often varies from application to application (i.e. whether a woman and her hat are "separate" objects may differ from image to image), we sought to create a result where sufficiently large objects spatially separated were marked as separate. By "separated on all sides," we specifically mean there are no missing pixels along the edge boundary, that may result in some of objects being undesirably "merged" into the same object. Clearly, the most desirable route was to make use of an algorithm that does not rely on neural networks, specifically the Canny Edge Detector algorithm. However, despite tinkering with its parameters in the OpenCV package, the set of test input images could never be tamed to produce desired output, often producing edges that were too sparse.
In turn, we explored neural network implementations, specifically N4, UNet, and HED. Each of these produces the their edge outputs in equivalent manner, with standard supervised learning on training sets of edge segmented images, optimized over mini-batch backprop with MSE training loss. Each of the three architectures described in the papers are more fully explained in the figures below:

Figure 5: The N4 network is an extremely simple architecture, almost elementary in nature largely owing to the fact it was developed/proposed in 2014. Clearly, its performance relies on extracting out features from the convolutions of local patches of the image from which we seek to optimize weights to map particular pixels to be edge pixels. [6]

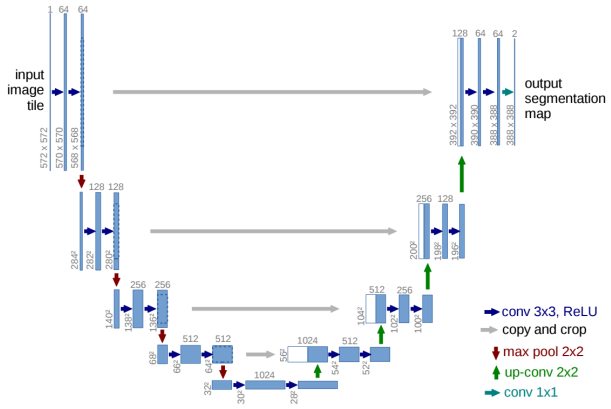

Figure 6: Very unlike the N4 architecture, UNet relies on a sophisticated combination of downsampled feature extraction and upsampling recombination. In particular, along the downward pass of the algorithm, we seek to obtain "high-level features" (similar to N4). However, as we wish for the output to be an edge map, we must upsample, essentially "expand" the compressed image (which theoretically contains all the information regarding where the edges are in the image) to the desired output size. In doing so, however, we lose much of the resolution that was present in the original image, as this is essentially equivalent to enlarging a low-resolution picture, which is why the images on the downsampling side are sandwiched on top of their corresponding upsampling layer, as they roughly correspond to the same patch of the image and improve resolution. [5]

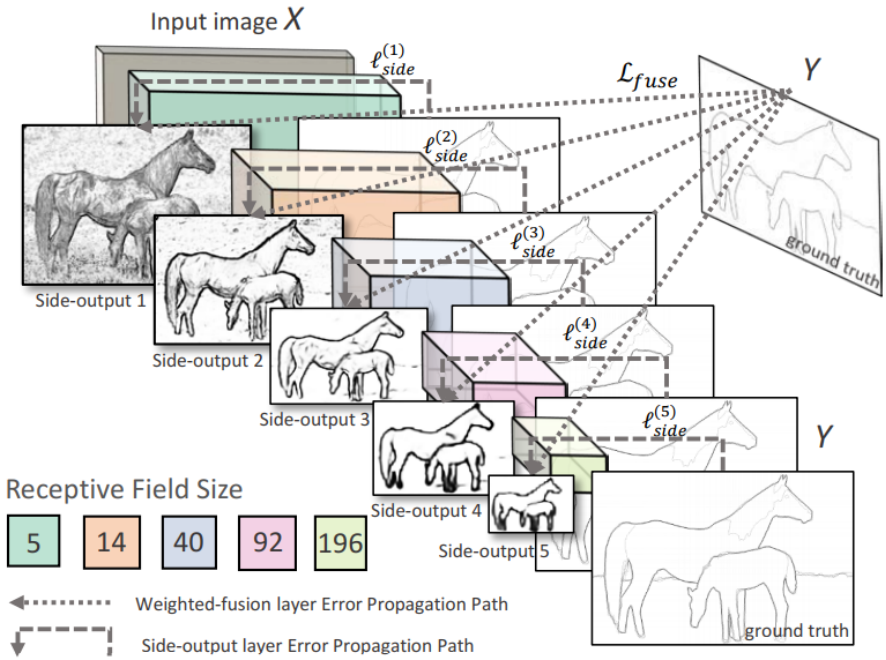

Figure 7: Akin to the UNet, HED relies on several layers of repeated convolution and downsampling, which are finally recombined to obtain a final segmented image. This once again relies on how the higher convolutional layers have extracted "higher-level" features but have, as a result, lower resolution. Thus, the main difference between this architecture and UNet is that the combinatory layer in this architecture takes all five intermediate outputs rather than recombining them one at a time. [7]

2.2.2 Otsu Thresholding

After getting a properly edge segmented image, we sought to apply various area-segmenting algorithms and eventually settled on a custom developed one. Specifically, the overarching idea is as follows. The edges given by the HED algorithm should bound each object in the image completely. Thus we want to extract each object separately, where different objects in the image are separated completely by the black background pixels such that there exists no path of non-background pixels between the two objects. To do this, we first needed an image where a clear distinction was made between edge and non-edge pixels. The best way we found to do that was through an application of Otsu Thresholding.[11]
Otsu Thresholding involves calculating the threshold at which the intra-class variance is minimized, where the two classes are the greyscale values of pixels less than the threshold, and the greyscale valus of pixels above the threshold. This value can be calculated both globally (for the entire picture) and locally (for a small neighbhorhood around each pixel). Then the pixels that are above the threshold are changed to the maximum value (255) and the pixels that are below the threshold are changed to the minimum value (0). We obtain better results through applying local Otsu thresholding rather than global Otsu thresholding due to variances across the image in intensity of the edges detected. This allowed us to process the edges of the images more cleanly.

2.2.3 Connected Components

After applying Otsu Thresholding, each foreground object should be completely bounded by white pixels. This makes it very convenient to extract the foreground images through a creative application of connected-component segmentation. [12]
The first application of connected component segmentation was simply to separate the background and foreground of the image. The background of the image was taken to be the connected component that contained the greatest number of pixels. Next, we re-labeled every pixel, where every pixel in the largest connected component was relabed as 0 and the pixel in every other component was relabeled as 255 (the maximum pixel value). Thus, the image contained pixels of exactly two labels - 0 for background pixels and 255 for foreground pixels. Next, one more application of connected components allowed us to extract each separate object (where separate object is defined as in the previous section) as objects of different labels. This allowed us to get the masks for each separate object in the foreground as well as a mask for the entire background of the image.

2.2.4 Tracking

The next challenge was to track the foreground objects between frames of the animation. To infer which label in the frame at time $t-1$ that an object in the frame at time t corresponded to, we simply took the foreground object in the frame at time $t-1$ with maximum overlap with said object. This allowed us to consistenly apply the style transfer to the same object in each frame.

3 Results

In implementing the above, we achieved the following results. Note that we also include the results of trials we ran but did not end up using in the final version

3.1 Style Transfer

3.1.1 Standard Transfer

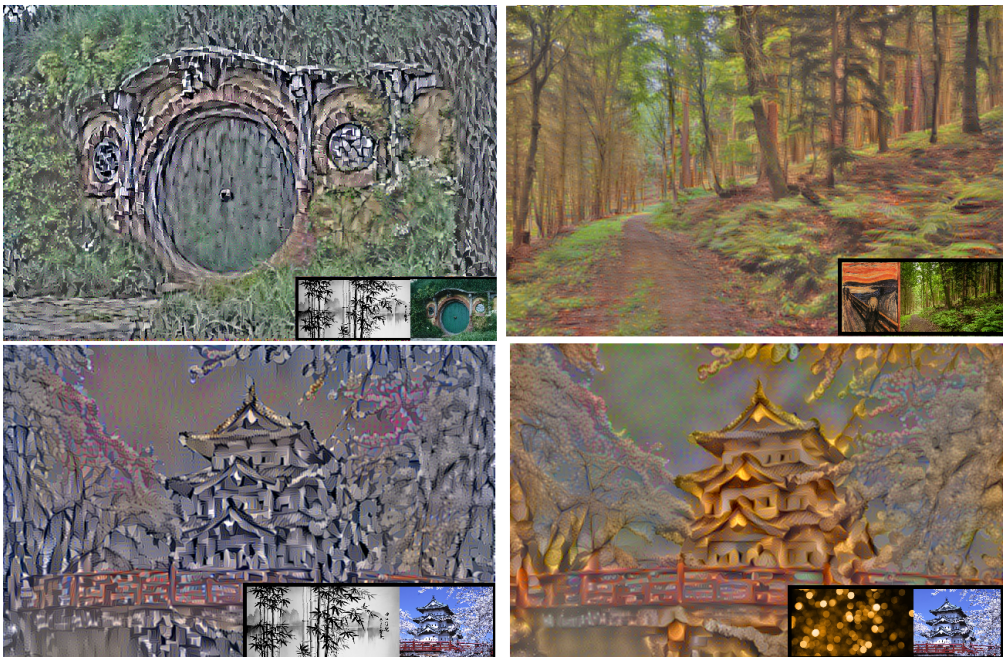

Figure 8: We were able to achieve the expected the style interpolation results described in Gatys' original paper. Note that these are labelled with their style and content images in the bottom right corner. Clearly, these are obscure styling images, meaning these would be more applicable to the standard implementation rather than the fast implementation, where a whole new loss network would have be trained correspondingly. We used $w_{content} = 0.025$, $w_{style} = .75$, and $w_{coherence} = 1.0$ for the network. Each takes around 5-10 minutes of GPU compute time to complete.

3.1.2 Fast Transfer

Figure 9: For these more mainstream stylizations, the fast transfer produces the above results, which are largely indistinguishable from the standard transfer algorithm. Note that (identically to the previous style transfer figure) these are labelled with their style and content images in the bottom right corner. Each of these takes around 5-10 seconds of CPU compute time to complete.

3.2 Segmentation

3.2.1 Edge Segmentation

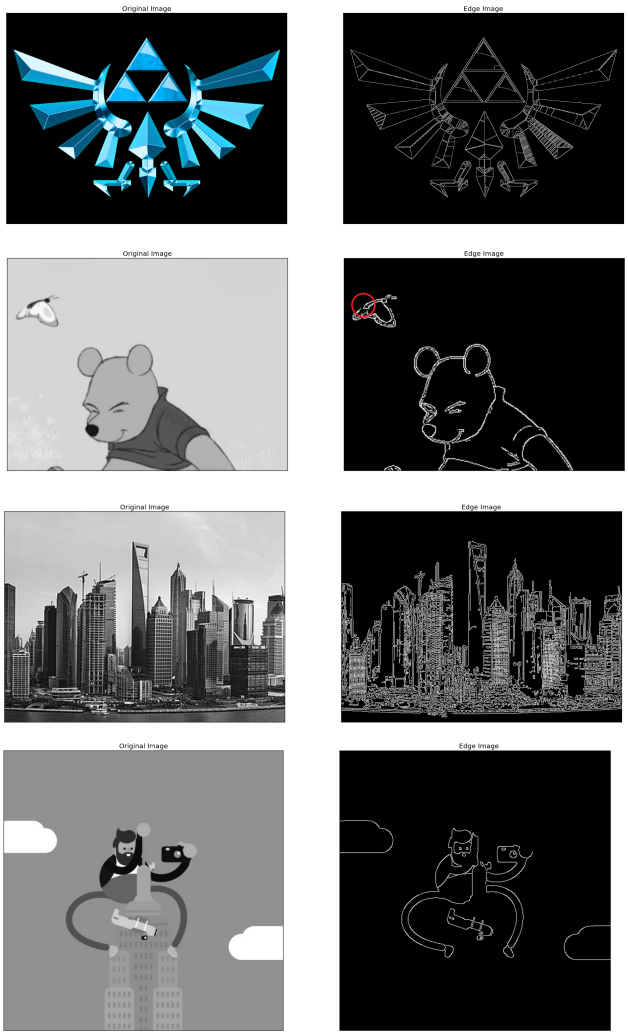

Figure 10: The Canny Edge detector does surprising well for being an out-of-the-box algorithm that requires no training, particularly at segmenting images that are well defined and large. However, if the edge is blurred (as in the case of the NYC giant picture) or too detailed (in the NYC skyline image), the edge detector fails, rendering area segmentation impossible. In fact, even in the Pooh segmentation, there is a small hole in the butterfly wing, which would result it being treated as simply a part of the background.

Out of the NNs, however, only HED showed significant improvement over the Canny outputs. Namely, N4 had the same issues as the Canny Edge detector, in that there was a lot of extra noise added to the images, which created more segments than were originally present in the original image. UNet, on the other hand, cleaned the image too much on its uppass of the network. Namely, it removed many of the pieces of edges that should have been in the final iteration of the image, resulting in segments of the image being merged. Thus, HED was the main contender for neural network improvement and the one we ended up using. Unlike the algorithm described in the HED paper, however, we used the outputs solely from one of the intermediate layers, as the fusion layer output was significantly more blurry than that produced through the layer two output, as visible in 11. The results, thus, indicated HED would serve well for our purposes.

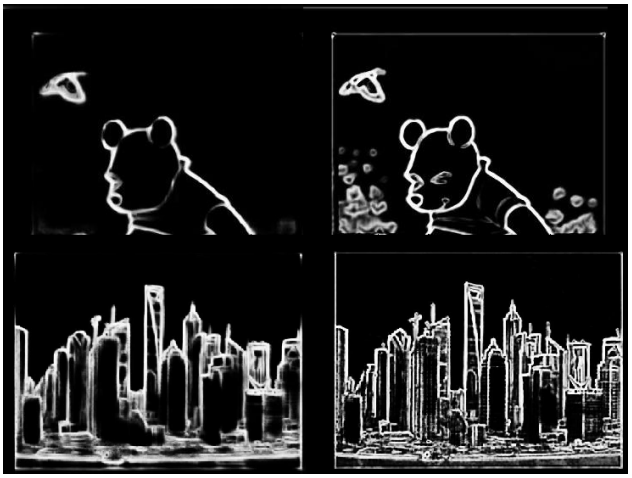

Figure 11: Though the fusion layer appears to hide the details away, as desired (to prevent outputs from being as thoroughly details as in the Canny output), it does so too actively, as clear in the outputs from the fuse layer (left) in comparison to layer two (right).

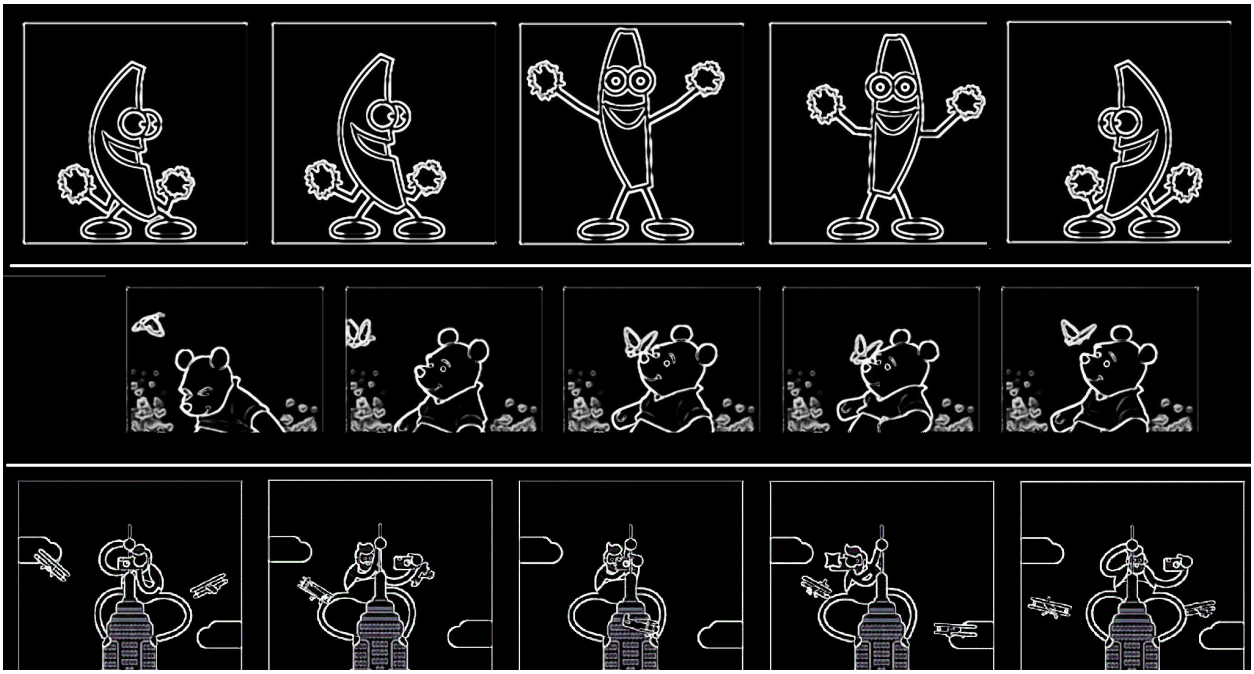

Figure 12: Unlike the other networks, HED produces sufficiently delineated boundaries for dealing with the final segmentation, shown in the above segmentation of GIF frames.

3.2.2 Connected Components Tracking

The masking works allows us to extract clearly defined object masks for various objects. It works especially well if the edges acquired from the HED algorithm are well-defined, with strong pixel values. Weaker edges result in parts of the foreground object to be connected to the background.

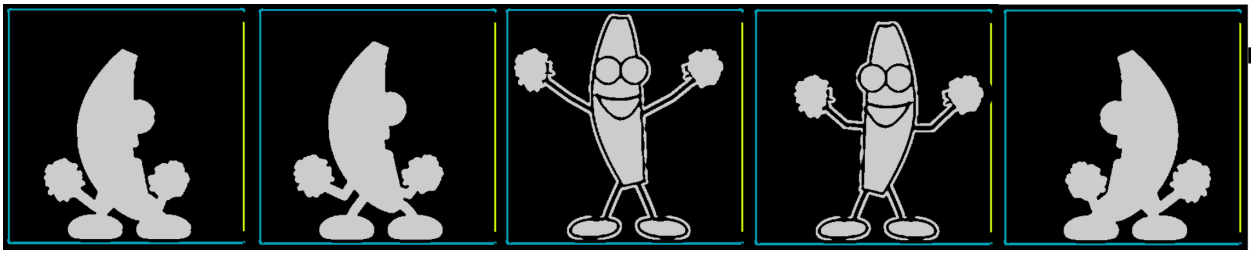

Figure 13: The masking works well for objects bounded by very strongly defined edges. As you can see from the third and fourth frames, some of the masks are not competely opaque, but some weak edges allow for background pixels to seep into what a human would defin to be 'inside' the foreground object.

Our tracking procedure to detect corresponding foreground objects in subsequent frames works well when the foreground objects have small displacements across frames. This is because overlap is required for the label to be strongly considered for the across-frame object correspondence. However, for users to choose one style for all foreground images and another style for all background images, this inconsistency of labels produced by fast-moving foreground objects would not cause an issue. However, future renditions of this project may consider using tracking algorithms like Lucas-Kanade.

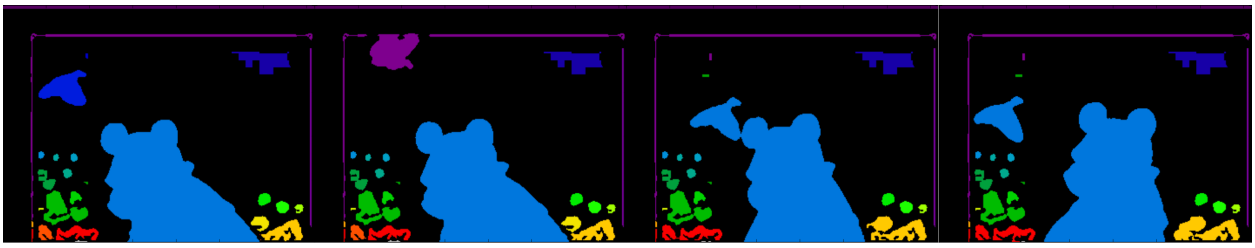

Figure 14: Tracking works well for foreground objects that do not move much. As you can see, the butterfly's label changes as it flies across the four frames because it does not overlap much with its previous position in the previous frame.

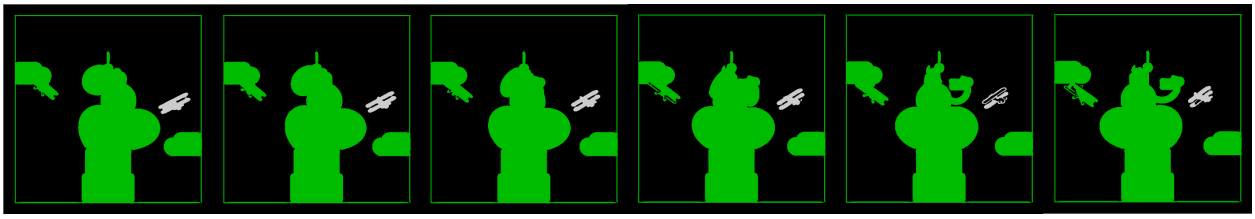

Figure 15: Here is a better example of mask tracking, where the frames are more closely tied together in terms of spatial continuity. We expect most of the input images to be of this form, clearly showing much more acceptable results than the former.

3.2.3 Final Style Transfer

The masks we obtained were not completely opaque so that when we applied different styles to the foreground and background of the image, parts of foreground objects may also have had the background style applied to them. The final image was still continuous enough so that the result was pleasing to the eye, and most of the foreground objects were styled to the appropriate style to give it a convincing overall effect in that desired style. Additionally, in the images that we tested, there were some edge effects from each style that

seemed to be derived from the edges of the foreground object, however these effects were visible beyond the edges of the foreground object. Thus, they were extracted along with the background when applying the style to the background of the final image. You can observe this affect in the following figure.

Figure 16: These are examples of frames from style transfer with various masks. Note that this is not the typical final product, as the main use case of our program was for doing such style transfer for GIFs. This link is provided below.

Plesae see the final compiled gifs at this link: `https://drive.google.com/open?id=0B1HvHLG6sd6pVy1PQjRydHZXYkk`

4 Conclusions

Looking to the main strengths and weaknesses of our system, we see that the processing is capable of largely capturing segmentation and providing alternate stylizing for continuous, well-defined images. That is, for images where the objects do not significantly overlap or shift rapidly, the program performs well and applies the style efficiently and properly. However, for weaknesses of the system, in videos where images move too rapidly, we cannot track the objects sufficiently well, which cause either objects to be merged into a single one or objects to be classified into a different class. Thus, the system fails to account for these cases, as mentioned in the stylizing section. Note that we will improve the implementation with the integration of Lucas Kanade, for tracking. In a similar line, the system fails to work well if an object full separates the background in half. In other words, if there is a wide object (i.e. a bridge in front of a sky backgound), then the entire sky will not be "classified" as a single background: instead, it will be either the sky above the bridge or the sky below the bridge. While this is not too significant, it becomes non-intuitive for the user to have to select that "both" skies be styled identically.

Therefore, in conclusion, we have developed herein a means of applying multiple styles quite efficiently across videos. Due to the speed in each piece of the algorithm, this method is highly scalable to longer videos and those of higher size and resolution. As with most neural network implementation, however, this does rely on having pre-trained models on GPUs to successfully work, as otherwise the style transfer time bottleneck becomes far to great to overcome, even on GPUs. Thus, further progress in this line may include developing a means of doing style transfer at rate comparable to those found in the Fei Fei version without pre-trained networks. Doing so would enable highly transferable video styling products.

Further, with the refinement of VR technology, the applicability of such image styling in VR too becomes a real possibility. The main applications here would be in prototyping for video game graphics designers, for it would enable them to simply sketch a scene and subsequently apply various pre-trained style models to see how they appear in the scene. Similar masking strategies can also be potentially extended to other domains, such as to natural language processing. Namely, by having an equivalent of "language masks," we may be able to produce more logical text outputs from RNNs, as different parts of a textual output may be written differently. Specifically, the introduction and body of a book or essay are often written in severely differing styles, which would roughly correspond to having a CNN produce outputs through two different masks: intro and conclusion. This is similarly the case for doing style transfer on music or sounds.

Thus, by enabling the mixture of multiple styles across isolated regions of a given image and integrating this with a unique tracking scheme, we enabled the creation of coherent, multi-styled videos.

References

[1] http://www.cv-foundation.org/openaccess/content$_c vpr_2$016/$papers/Gatys_I mage_S tyle_T ransfer_C VPR_2$016$_p aper.pdf$

[2] https://arxiv.org/pdf/1603.08155v1.pdf

[3] https://arxiv.org/pdf/1511.08861v2.pdf

[4] https://arxiv.org/abs/1511.00561

[5] https://arxiv.org/pdf/1505.04597.pdf

[6] https://arxiv.org/pdf/1406.6558v2.pdf

[7] https://arxiv.org/pdf/1504.06375v2.pdf

[8] http://www.scipy-lectures.org/packages/scikit-image/

[9] http://cmm.ensmp.fr/ beucher/wtshed.html

[10] https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/files/seg.png

[11] http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL$_C OPIES/MORSE/threshold.pdf$

[12] http://dl.acm.org/citation.cfm?doid=128749.128750