User-Centered Non-Photorealistic Rendering

Nabib Ahmed, Cole Bateman, Fraser Darling, and Arnav Srivastava
ENG-SCI 143: Computer Vision
Fall 2020 Final Project

Abstract

This project constructs a user-friendly and robust non-photorealistic rendering. A great deal of cutting edge methods are present when dealing with non-photorealistic rendering technologies, but as a whole these techniques are both not accessible to the average artist, and not in a centralized location. Our tool explores the interactions between the artistic methods of tone mapping, image filtering, and neural-network models with options for combining rendering techniques and adjusting relevant image processing parameters.

1. Introduction

Non-photorealistic rendering (NPR, and also known as rotoscoping) is a technique which allows artists to convert real-world photos and videos into a realistic, cartoonish effect for use in their artwork.

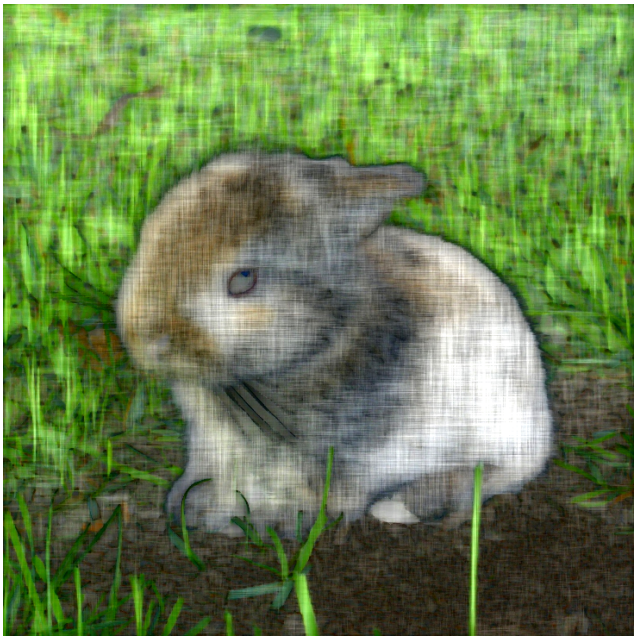

Figure 1. Stylized image using a combination of methods described throughout the report.

Recently, there has been an increase in research surrounding automatic rotoscoping, however we have not been able to find any projects that prioritize user friendly interaction, variability, and modularity. Our project aims to do just that by pulling from both HCI and Computer Vision. Thus the project goal is to build an application that will allow users to upload photos and videos, and then apply a wide variety of photo effects in addition to art rotoscoping effects. In regards to ES 143 specifically, our project will be focusing on utilizing convolutional properties, image processing, and feature detection. Much of the functionality covered in class from OpenCV will be incorporated as features for our project.

The project implements numerous automatic rotoscoping techniques alongside color mapping operations and neural-network-based foreground/background serparation, all unified by the user interface. While there are many available techniques to perform NPR-relavent effects, most of these are not accessible to the average user with control over specific modifiable parameters. The project builds on the image processing pipeline, color filters, and tone-corrections (such as gamma correction) covered in Modules 2 and 3 of ES 143. The user is given the ability to select various parameters from the implemented rendering options, and returned the processed NPR image.

1.1. Related work

The main inspiration for this project comes from video stylization work in Winnemoler et al.[8], where they construct a real-time video and image framework for abstracting luminance and color opponency from RAW image files, and manipulating contrast with diffusion and difference-of-Gaussian convolution. Finally, abstracted images can optionally be stylized using soft color quantization to create cartoon-like effects. The framework places emphasis on GPU-based, real-time implementation.

Additional motivation comes from Xu et al.[**Xu**], which uses global small-magnitude gradient removal in order to globally retain and sharpen edges while blurring over small

color changes. This L0 smoothing is particularly useful for cartoon image artifact removal, and NPR transformation for cartoon and pencil sketch based on gradient suppression or edge mapping.

Our project seeks to build upon this work by mimicking similar NPR effects in an accessible interface to the user, alongside offering functionality for other NPR effects as discussed in the Methods.

1.2. Coding Methods

Our photo and video rendering pipeline was implemented using Python3 and can be run in the command line. We have included a README file which guides new users through the set up process; upon first run, be sure to follow the README carefully to ensure proper setup. We have also included a setup.sh file that ensures the directory's organization is correct.

The following packages are required to run our application: tensorflow, PIL, cv2, numpy, io, sys, datetime, and PySimpleGUI. tensorflow was used to implement a deep-learning image foreground-background separator and PySimpleGUI was used to implement our application's front end.

1.3. Computational constraints

The methods described in this paper and implemented into the styling tool were processed using state-of-the-art hardware including a NVIDIA 3080 Graphics Processing Unit. Some of the video rendering options require an extremely high computational cost and might not be possible on lower end hardware. Extensive testing of what is needed was not completed, please let the authors know if you are running into difficulties. We suggest down-scaling the quality of your images and videos as the first thing to try. The real-time video processing is expected to be very slow on a normal computer.

2. Methods

The image processing pipeline (ignoring user-interface construction) consists of three distinct parts: color mapping operators, selected image filtering techniques, and machine learning image convolution. The final interface allows users to choose multiple image transformations within each of these categories and choose relevant parameters for different operators, with potential for different operations applied to different parts of the image. For instance, foreground/background separation allows for image compositing with NPR transformation in the foreground and background.

2.1. Color mapping operators

Some of the most fundamental image processing transforms are known as point operators, where each output pixel's value depends corresponds to the input pixel value, potentially alongside some global information about the entire image. Examples of such operators include contrast adjustments (gamma correction), color transformations, and intensity rescaling operations (such as histogram equalization and contrast stretching). These operators are all inspired by Szeliski 3.1. All of these operations were coded into the database to provide the users more parameters to modify output NPR images by enhancing certain features or making images more non-realistic.

2.1.1 Gamma correction

Gamma correction corresponds to the relationship between a pixel's intensity and its perceived luminance. Since human perception of brightness is nonlinear (such that doubling brightness of screen will not double intensity of pixels human perceive), a nonlinear gamma function adjusts the brightness of an image to match our expected change in brightness.

Image brightness corresponds to pixel intensity, and gamma correction modulates pixel intensity by nonlinear parameter γ according to the relationship $I' = I_(max) * (I/I_(max)))^{\gamma}$

2.1.2 Color transform

In the interest of providing users with an additional aspect of non-realism for NPR, we provide users with the ability to transform images set using pseudocoloring. Based on the specified color map, the intensity of every pixel in a given image is mapped to a certain color. We provide users with 12 different color maps, which allow users to transform the appearance of their image. Users are further allowed to scale how much of this pseudocolored image is averaged with the original image.

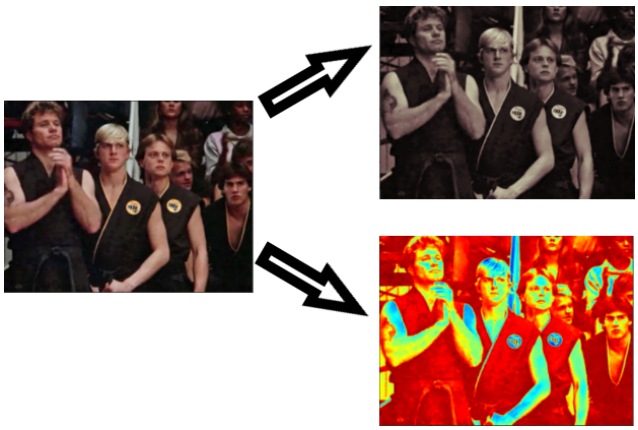

Figure 2. Example of original image (left) being mapped to 2 different pseudocolored images (right) based on different color maps.

2.1.3 Histogram equalization and contrast stretching

Histogram equalization is an alternative method to maximize ideal pixel intensities in a global sense. Histogram equalization constructs a histogram of pixel intensities in a given image, and uses an equalization function to flatten this distribution of intensities in the image. By effectively spreading out the most frequent intensity values, this allows for areas of lower contrast to be emphasized. The equalization function corresponds to integrating the intensities of the histogram $h(I)$ to obtain the cumulative distribution of intensities, where we can look up the a given pixel's intensity I in order to determine its equalized intensity $c(I)$, given as follows by Szeliski 3.1.4 [6]:

$$c(I) = (1/N)\sum_{i=0}^{I} h(i) = c(I-1) + (1/N)h(I) \quad (1)$$

Histogram equalization works well for intensities values for RGB images, which corresponds to equalization for grayscale outputs. Histogram equalization for colored images can be performed by histogram equalization over the Y value of an image converted to the YUV color space, and then reverting the image back to RGB.

An alternative method which allows for contrast optimization in an image is contrast stretching. In contrast stretching, the highest intensity pixels are mapped to the maximum allowed intensity, whereas the lowest intensity pixels are mapped to minimum allowed intensity, making the input image use the entire range of values available to them similar to histogram equalization. However, for contrast stretching, there exists a one-to-one relationship of the intensity values between the original image and stretched image. Contrast stretching and histogram equalization both produce slightly different outcomes, with different tone final results.

2.2. Filtering techniques

Filtering is an extremely common way of altering the information stored in images. Many technologies exist where users can apply and customize image filters, (Photoshop, Snapchat, etc) but the next generation of image processing tools will also include these methods as they both create cool effects, and are the fundamentals of more advanced methods.

2.2.1 Convolution and correlation

The fundamental idea behind filtering is the creation of a 'kernel'. The kernel represents a mapping of pixel values to a weighted average of their neighboring pixels. Convolutional filtering is the process where pixels are weighted as the mean of their neighboring pixels, and correlation filters also use a kernel-style mapping but are not necessarily the weighted average of the neighboring pixels and can apply different weightings (ex. Convolutional filter rotated by 180 degrees). Mathematically the discrete convolution operator is given as:

$$f[x] * g[x] = \sum_{n=-\infty}^{\infty} f[k]g[x-k] \quad (2)$$

Where the function $f[x]$ is the image processing function we are trying to apply, and $g[x]$ is the function that is defined by pixel values of the source image, and the result is a filtered image. Some examples of functions are explored below.

2.2.2 Gaussian blurring

One of the most common image processing functions is a Gaussian Filter, it is used to create a smoothing effect throughout the image by removing noise that is assumed to come from a Normal distribution. The distribution used for sampling weights in this case is the 2D 0-Mean discrete Gaussian function

$$g[i,j] = e^{\frac{-(i^2+j^2)}{2\sigma^2}} \quad (3)$$

Here, σ determines the strength of smoothing: gaussian blurring replaces a pixel's value with the weighted sum of its neighbors, where the size of this neighborhood (and thus how much impact neighbors have in "averaging" a pixel value) is controlled by σ. Evidently, the gaussian acts as a low-pass filter in the spatial domain. This is easy to implement, and creates solid results that you can see in the gallery below.

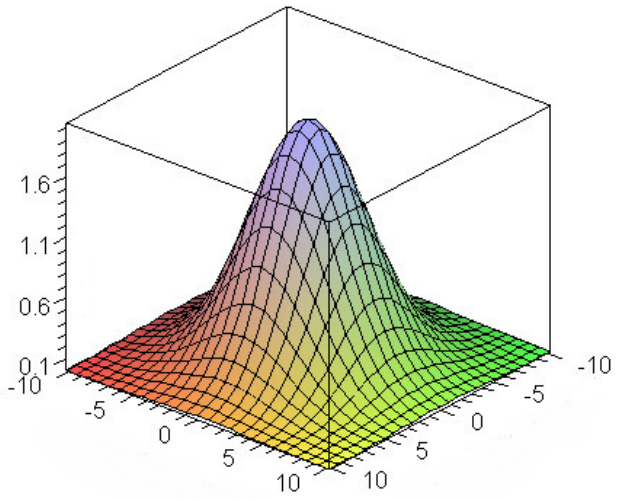

Figure 3. Visualization of a general 2D Gaussian Filter.

2.2.3 Edge detecting filter

When looking to determine features of an image, the best way to determine points of interest is to look at the edges. The filter that is used in our implementation of this project is centered around a filter called the "Laplacian". In our case it is implemented after Gaussian Smoothing to reduce high frequency noise that would have unwelcome effects on the overall filter and it is implemented generally following:

$$LoG(x,y) = \frac{1}{\pi\omega^2}(1 - \frac{x^2+y^2}{2\omega^2})e^{\frac{-x^2-y^2}{2\omega^2}} \quad (4)$$

This operator essential calculates the second derivative of the image space. An 'edge' corresponds to a change in intensity between neighboring pixels, and the second derivative format will create a really high output result after being applied.

2.2.4 Bilateral filter

As we continue the journey towards non-photo-realistic rendering, a filter that builds off of the previous two to achieve the result is the bilateral filter. This filter can be used to create sharper cutoffs between regions of intensity for a cartoonish effect, effectively combining the smoothing of pixel values from gaussian blurring while preserving edges/color boundaries to retain image definition. This generation of "false edges" is sometimes considered a limitation of bilateral filtering, but serves well for NPR-driven image transformation. The derivation of this filter (from Szeliski) is as follows. The output pixel value is formed by first taking a weighted sum of neighboring pixels.

$$p[x,y] = \frac{\sum^{k,l} f[k,l]g[i,j,k,l]}{\sum^{k,l} g[i,j,k,l]} \quad (5)$$

The weights are uniquely determined by using a combination of two more distributions, in our case Gaussian distributions, taking the form of range and spatial kernels. The range kernel smooths the intensity change between the neighboring pixels given by:

$$r[i,j,k,l] = e^{\frac{(i-k)^2+(j-l)^2}{2\sigma_r^2}} \quad (6)$$

This is then multiplied against another function that smooths the difference of coordinates, which is called a spatial kernel. Again Gaussian in our implementation.

$$s[i,j,k,l] = e^{\frac{|f[i,j]-f[k,l]|^2}{2\sigma_s^2}} \quad (7)$$

Which sum to the weight function. We now arrive at the complete definition of the filter. For coordinate x of the current pixel to be filtered, we have:

$$B[x] = \frac{1}{W_p}\sum^{|x|<1} p[x_i]r[|p[x_i]-p[x]|]s[|x_i-x|] \quad (8)$$

Where the normalized term of W_p is equal to

$$W_p = \sum^{|x|<1} r[|p[x_i]-p[x]|]s[|x_i-x|] \quad (9)$$

This is the definition of the entire filter, and the parameters of σ_r and σ_s present in the weighting term provide the degrees of freedom for the user in our created tool. These parameters smoothen larger features, and flatten the Gaussian respectively. We encourage the reader to give them a try using the tool to create their desired effects.

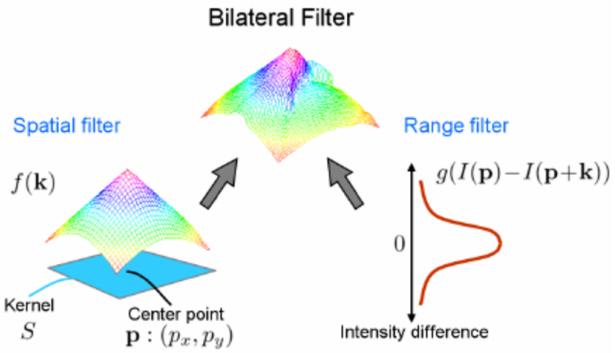

Figure 4. Visualization of a Bilateral Filter

This bilateral filter effectively uses a spatial Gaussian to ensure that only neighboring pixels are considered for blurring and a range Gaussian of intensity difference makes sure that only those pixels with similar intensities to the central pixel are blurred, in order to ultimately preserve sharp edges while blurring (since image edges present large variance in intensity). [7]

2.2.5 Domain Transform for edge-aware filtering

For 2D images, bilateral filtering can be interpreted as a 5D spatially-invariant kernel (3 color channels alongside pixel coordinates), whose response decreases as the distances among pixels increase in 5D. Given the high dimensionality of the filtering process, bilateral filtering presents high computational costs. In interest of a user-focused interface, we have included an additional faster edge-preservation filtering technique which can provide NPR relevant effects in linear time to the user (the user can see the filtered image result in real time so that they have better control over implementing their NPR vision). Gastal et al. provide a domain transformation for edge preserving smoothing which holds close results to bilateral filtering but is significantly faster[2], providing multiple stylization options relevant to NPR.

The domain filter defines an isometry (transformation where distances are preserved) in a space of lower dimensionality, such that many spatially-invariant filters are still

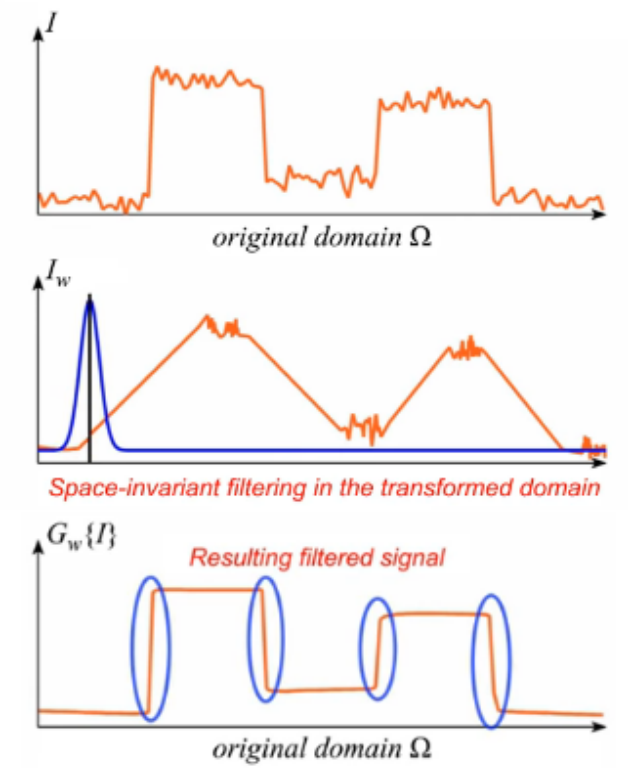

Figure 5. Pipeline demonstrating the image transform from original Ω domain to transformed domain for filtering, and then reverted back to original domain.

edge-preserving but the computational cost of determining filter strength has decreased. For a given 1D signal $I : \Omega \rightarrow \mathbb{R}^2$, I defines a curve C in $\mathbb{R}^2$ by the graph $(x, I(x))$, for $x \in \Omega$. For any two points u and w in Ω, $w \geq u$, we define the distance between them in the domain transform ct as:

$$ct(w) - ct(u) = \int_u^w |1 + I'(x)|\, dx \tag{10}$$

such that the geodesic distance between all points on the a curve of C is preserved even following dimensionality reduction. Scaling this finding to 2D RGB images, the domain transform for all channels is:

$$ct(u) = \int_0^u (\sigma_r/\sigma_s) \sum_{k=1}^{c} |1 + I'_k(x)|\, dx \tag{11}$$

Ultimately, by applying the domain transformation to high dimensional space, we protect detected edges in the frequency domain, allowing us to then efficiently smooth over the signal without disturbing the edges. Reverting the signal to the original Ω domain gives a similar but faster result compared to bilateral filtering.

This domain transform provides various real-time applications. While the domain transformation filtering itself offers similar rotoscoping to bilateral filtering, the subtraction of this module with the original image provides "detail enhancement" capabilities. Meanwhile, the gradient of the filtered image superimposed to the filtered image itself produces additional high-contrast edges around important features. Additionally, by assessing the transform neighborhoods in the pixel domain, we can resemble pencil sketch functionality. All of these can provide useful NPR functionalities.

2.3. Machine learning methods

Computer Vision as a field deals with the process of extracting information from images. Artificial Neural Networks are perfectly suited for this information extracting task. They are made up of an array of connected 'neurons' which take input data and apply an activation function, then pass along the result. The network "learns" the optimal parameters to feed into the activation functions through a back-propagation process, and a prediction is made after a number of iterations. However, there are some problems

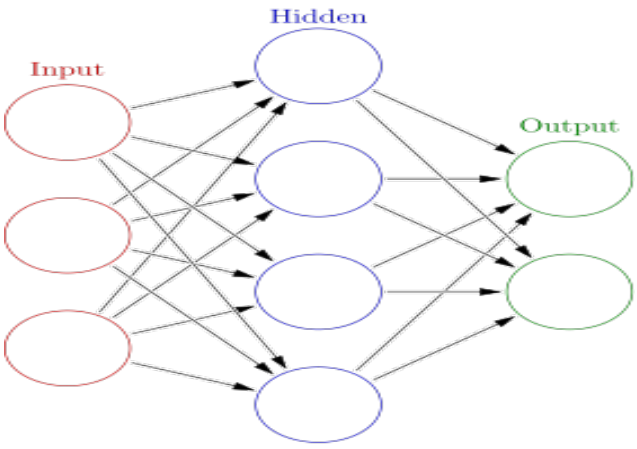

Figure 6. Artificial Neural Network

with the regular neural network model when it comes to image processing. Firstly, most interesting images contain at least hundreds of pixels and three channels, which will result in millions of parameters and a high likelihood of overfitting. Second, the process becomes increasingly expensive computationally, and is generally harder to work with along the lines of interpreting results and debugging.

2.3.1 Convolutional Neural Networks

At the heart of many Computer Vision focused models lies a Convolutional Neural Network. This network solves the above problems by not having all of the neurons in a single connected system. Instead, the connection is divided up into three stages. The first stage is the convolutional operator, see the definition of a convolutional filter for an

explanation, that creates a map of probabilities that a goal feature is being selected for. The second layer consists of

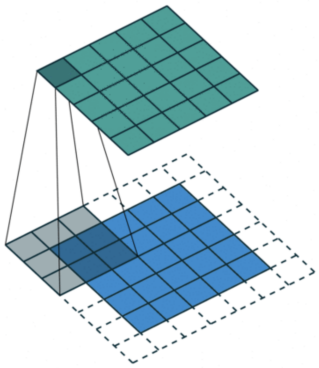

Figure 7. Convolution Stage

pooling, which usually consists of lowering the dimension of the data by just taking the highest value from each pixel area, a process called 'Max Pooling'. This allows us to retain the most important features and further reduce the complexity. Finally, the results of the second stage are fed into a

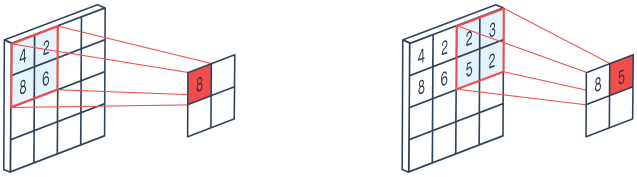

Figure 8. Pooling Stage

smaller fully connected neural network as described above and a prediction is made about where the goal features are on the image if any are present.[1]

2.3.2 Foreground-Background CNN extraction model

The main focus of the machine learning aspect of the tool is a foreground background separation. There are many reasons why this would be desirable, and combining it with the other filtering techniques one the separated parts creates incredible visual effects.

The feature that this model is trying to extract is the edges around the foreground. The method that we selected came from a github author[5], who in the pre-processing uses two CNN models to detect boundaries, cut them out, and piece back foreground into the resulting image. Then a smoothing filter is applied that gives a cleaner result.

2.4. Future directions - Models on the horizon

While we have created a platform that starts to address the growing methods of image processing, the number of up-and-coming methods gets larger every day. For a tool like this to stay relevant, there is a need for it to be constantly updated and expanded upon. Looking forward, there are a couple of different machine learning models in particular that would add unique processing aspects not captured in the other parts.

Figure 9. Example of the CNN's used to distinguish between foreground and background for compositing applications.

2.4.1 Generative Adversarial Networks

One possible method that achieves great results is a Generative Adversarial Neural Network. In simplest form, there are two smaller entities (networks) that are playing a game against each other. A blank canvas for an image is created, and then the first 'Generative' network attempts to place some mixture of two images together on a particular spot on the canvas. Then, the second "Discriminator" network distinguishes the created data from the characteristics that we are looking for. It then tells what is wrong with the generated section of the image to the other network, the generative settings are changed and then the process begins again. The process then ends when the discriminator's setting changes do not affect the image to a certain degree, having captured a balance of style and content between two images. This could provide effects greater then any filter due to the multiple image dependence of the output. [3]

2.4.2 K-Means classifications

A clustering algorithm is one that attempts to identify closely related segments of a dataset. The algorithm runs with the goal of the squared distance of all points to a cluster center. The objective function is:

$$J = \sum^{k} \sum^{n} |x_i - c_j|^2 \tag{12}$$

Where k is the total number of clusters, n is the number of cases, x_i is the specific case, and c_j is the centroid for the jth cluster. The algorithm runs as follows:[4]

Figure 10. Sample NPR compositing example. The user's input image (left) is transformed by applying a color map (middle). The user then additionally decides to add both style and sketch domain transformation filtering for a finished cartoon rendering.

```
1. Choose Number of Clusters K
2. Select N Points
3. Assign each point to nearest K
4. Compute the new Centroids
5. Reassign data until there is
nothing else needed to change.
```

With this method, we can combine the values that come out of other filters and processes to create different styles of segmentation. Saturation, average values, lightness are just some of the aspects that we can improve upon when we look towards improving the methods we work with.

3. Results and Discussion

Our application is comparable to mobile apps with image filtering functionality, like Instagram or Snapchat. Where we differentiate, however, is in the flexibility of our application, the ability to layer effects, and the cutting-edge nature of our implementation. For example, within Instagram's image editing suite, you are only able to change the color tone and structure of your image. Our tool allows for the addition of one or more complex effects on top of similar functionality to Instagram.

We consider our implementation to be a successful attempt at implementing our original idea. We were able to combine a variety of different tone mappings, image filters, and deep-learning tools under a usable interface, which makes unique NPR image editing accessible to anyone.

3.1. Limitations

Our application is not without its limitations. First and foremost, the processing power necessary for a smooth experience in the interface (with both images and video) is quite large. Additionally, we have found that there are some combinations of effects that can cause the system to crash, namely combinations including Threshold, Canny, Hue, or Enhance. We believe this is due to the change in color channels these filters create in the input image. Finally, some functionality is not usable between both image and video processing, either due to processing constraints or unfinished implementation. For example, we cannot run our background-foreground separation utility on a video without the process taking extremely long. We will talk more about additional functionality in the next section.

3.2. Future Functionality

For future iterations, there are other features we would like to implement for increased functionality and a smoother user interface. Due to computational constraints and in interest of having a decent runtime when processing videos, we expect less NPR options available for video modification. We would ideally have a separate panel that allows users to choose whether they want to edit a still image or a video, and offer NPR features accordingly. For the time being, we have both images and videos being inputted into the same pipeline. However, some NPR capabilities load at a faster frame-by-frame rate than others.

We also foresee additional functionality being made available for the machine learning part of our pipeline. Alongside considerations discussed in 2.4 Future directions - Models on the horizon, we would like to have more robust foreground/background NPR applications. For instance, although we can currently separate an image's foreground and background, our matting algorithm's compatibility with our interface only allows for tone mapping and filtering to be applied on the foreground. It would be of interest to allow users to have different NPR operators on the foreground and the background, and then potentially recombine these images to increase the artistic capabilities of our software since the foreground and background would have distinct appearances and textures. Furthermore, we could implement other deep-learning NPR models integrated with our user interface, such as ArtLine, a more robust pencil sketch module.

3.3. Usability

One of our major goals within the project was to create a system that was usable and flexible for the end user. While more UI/UX testing is required, we believe that our work accomplishes this goal due to three factors:

1. The ability to adjust the parameters on all filters and applicable tone mappings

2. The inclusion or exclusion of relevant menu buttons, depending on the input. This helps to avoid confusion concerning unimplemented features.

3. The simplistic layout, with no functionality hidden.

Future work would include testing with our stakeholders to determine any areas that require tweaking.

4. Acknowledgements

Thank you to the ES 143 staff for an incredibly educational and interactive semester developing our foundations of computer vision.

References

[1] *Convolutional Neural Network: How to Build One in Keras amp; PyTorch*. URL: https://missinglink.ai/guides/neural-network-concepts/convolutional-neural-network-build-one-keras-pytorch/.

[2] Eduardo S. L. Gastal and Manuel M. Oliveira. "Domain Transform for Edge-Aware Image and Video Processing". In: *ACM TOG* 30.4 (2011). Proceedings of SIGGRAPH 2011, 69:1–69:12.

[3] Ian J. Goodfellow et al. "Generative Adversarial Nets". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, 2014, pp. 2672–2680.

[4] *Introduction to Image Segmentation with K-Means clustering*. URL: https://www.kdnuggets.com/2019/08/introduction-image-segmentation-k-means-clustering.html.

[5] Susheelsk. *susheelsk/image-background-removal*. URL: https://github.com/susheelsk/image-background-removal.

[6] Richard Szeliski. *Computer Vision: Algorithms and Applications*. 1st. Berlin, Heidelberg: Springer-Verlag, 2010. ISBN: 1848829345.

[7] C. Tomasi and R. Manduchi. "Bilateral filtering for gray and color images". In: *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. 1998, pp. 839–846. DOI: 10.1109/ICCV.1998.710815.

[8] Holger Winnemoeller, Sven Olsen, and Bruce Gooch. "Real-time video abstraction". In: *ACM Trans. Graph.* 25 (July 2006), pp. 1221–1226. DOI: 10.1145/1179352.1142018.