Image Processing Project Report

Team 1

Ahmed Khaled	S01 BN 03
Zeinab Rabie	S01 BN 26
Sara Maher	S01 BN 28
Abdelrahman Mohamed	S02 BN 03

Computer Engineering Department - Cairo University

Style Transfer

1 Idea

Transferring artistic style from one image to another is an interesting application of image processing: it'd be quite desirable to take a painting by Rembrandt and render a modern photograph in similar style. Aside from direct applications (like FaceApp, Instagram, Facebook, and others) such style transfer would also be very useful in video games and movies as a way of rendering scenes in a new light without much manual intervention. Style transfer can be seen as a good first step towards genuine computational creativity. We will implement [1] which is the current state of the art in image-processing based (*not* end-to-end) style transfer. The results are quite visually pleasing, see Figure 1 for an example.

2 Pipeline

The block diagram is included in Figure 2. Color transfer is usually done by histogram matching, Gaussian pyramids are constructed by successive approximations, segmentation masks are used to keep unimportant or unwanted details in the style reference from being carried over to the input image. The least squares minimization problem is solved iteratively. This pipeline is again adapted from [1].

3 Non-primitive functions

Functions for efficient and/or approximate nearest-neighbor search (using tree-based methods) might be needed. We'd use an open source library (like nanoflann in C++) or implement it ourselves if time permits. However, the image processing functions will be implemented manually if they are not in the scope of the course (like denoising).
In phase two, we got permission to use non-primitive functions for segmentation (Chan-Vese segmentation and graph cuts for face segmentation).

Figure 1: Input image - style image - output, adapted from [1]

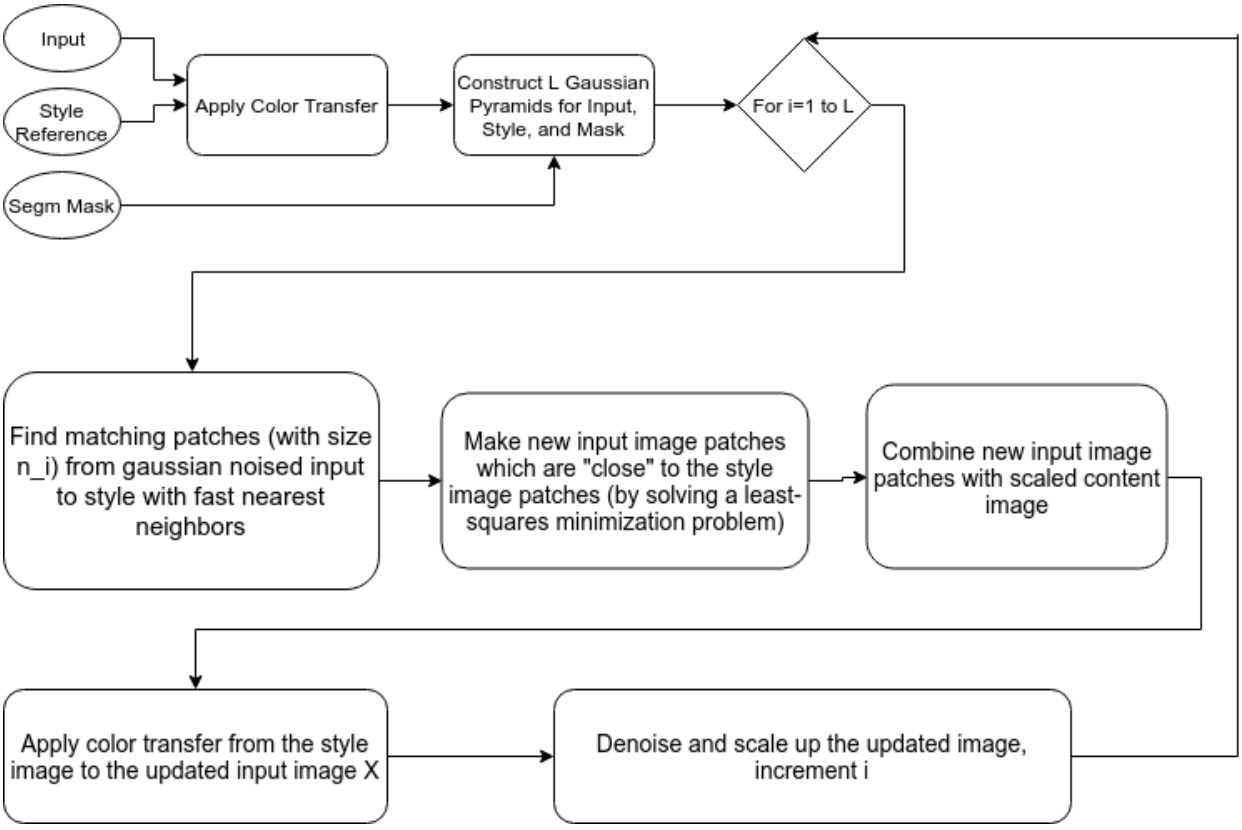

Figure 2: Block Diagram

4 Used Algorithms

4.1 Color Transfer

We tried these algorithms:

1. Color histogram matching
2. Luminance-only transfer using LAB space
3. Alternative Color histogram matching

Results:

1. The first algorithm gave visually unsatisfying results.
2. The second method preserves the color and changes the luminance only, based on the fact that the human eye is affected more with the change in luminance rather than change in color. It works on just one channel instead of three so it's somewhat faster than the other two.
3. We decided on using the third one, which gives a more appropriate result, a noticeably more color-transferred image.

4.2 Segmentation

4.2.1 Edge Segmentation

The first step of edge segmentation is edge detection. We tried the following methods:

1. Canny Edge Detection.
2. Edge Detection based on local gradient statistics.

Results: the edge detection through local gradient statistics was robust and efficient, therefore we used it.
Comment: The paper didn't mention the used thresholds so we adjust them until we reach a satisfying result.
After performing edge the paper specifies using a filling algorithm to fill between edges to get a filling mask that reserves important details in the content image. However, no specific filling algorithm is specified. A major problem with filling is that to get an optimal result edges must be closed which almost impossible to happen unless the objects in the image are very simple. We tried six different approaches.

1. Filling Holes
   A very simple algorithm that fills any closed shape. This algorithm is practically useless as edges are almost never closed.

2. Convex Hull
   Convex hull detection is getting a convex polygon that surrounds a given set of points. If we detect the convex hull of edges and fill it we are guaranteed to cover the object in the image. Two problems however, arise. First edges are noisy and doesn't necessarily cover just the main object of the image. This means that outliers may exists in the data which dramatically affects the performance of the algorithm. To solve this issue we threshold the edges with a high threshold value to remove as much noise as possible. After several trials threshold from 0.65 to 0.85 seems to get good results. The second issue is that convex hull may get a very bad result if the object is concave. We need to subtract the extra regions from it.

3. Watershed
   Watershed is a segmentation algorithm that segment based on distance between local maximas. First some markers are defined on local maximas then image is segmented into regions where every pixel falls into the nearest marker. Applying this technique on edges should lead to the object regions taking high gray levels and background taking low gray levels. Some problem arises as the algorithm doesn't detect background it tends to give big regions darker gray levels which in some cases could lead to a high error. Edges also must be dense and we need a lot of them to make sure that object regions are not open to background which may lead to parts of the objects considered as part of the background. This means that edges are to be binarized with low threshold to get more edges included range from 0.15 to 0.3 is found to get good results. Still the major problem that gray level thresholding changes the results dramatically with with small changes.

4. Mixing Convex Hull and Watershed
   From the previous section it's obvious that both Convex Hull and Watershed alone doesn't get good results. However, we be able to get better result of we combine both of them. Considering that Convex Hull takes extra regions but it doesn't include extra regions like watershed. Where watershed gets the shape of the object better but it might falsely take an extra region with it. So if we multiply the results of both them we may get an output that both doesn't get include extra regions and gets the shape of object correctly. This approach overall gets better results compared to the previous two approaches however it's still have a downside of being very susceptible to outliers.

5. Chan Vese
   Chan Vese is a segmentation algorithm designed to segment objects without clear boundaries which makes a very good approach for our case. The algorithm have a set parameters. That represents different variables. After tuning the parameter the following values seems to get the best results. $\mu = 0.1$ $\lambda_1 = 0.06$, $\lambda_2 = 1$, tol= 10^{-3}, init level set=checkerboard or edges, $dt = 0.52$ and num interation = 2000. This output while not always completely filled it's object is generally obvious without extra regions, easy to make better by applying dilation or filling holes if needed (in our case that didn't help instead it included more background in the mask). A dilation could also be applied on the edges to before applying the algorithm. Still one little problem occurred which is that the parameters are tuned manually on a small dataset which

mean they might lead to bad results on different input. Solving such a problem would require using machine learning on a big data set to learn these parameters. However, such a data is not available and would need to created by hand as the paper does use a custom edge detection method. Creating and labeling such data would be unfeasible as it need to be drawn and filled manually. The algorithm also takes a lot of iterations to get good results which makes it slow.

6. Morphological Chan Vese
   This algorithm is an extension to Chan Vese that segments objects with ill-defined boundaries given that their inside in different in lighting than the outside on average. Trying this algorithm with the gray scale transformation of the image. Using edges as initial level leads to very good results. It's still have some parameters like lambda1 and lambda2 however, it still gives good results using the default values and with low number of iterations (less than 50) which makes it really fast.

Functions used in filling:

1. skimage.draw.polygon(): Constructing polygon in convex hull.
2. skimage.filters.gaussian(): Removing noise from morphological chan vese output.
3. skimage.segmentation.chan_vese(): Perform chan vese segmentation.
4. skimage.segmentation.morphological_chan_vese(): Perform morphological chan vese segmentation.
5. skimage.filters.rank(): Getting local maximas for watershed markers.
6. from skimage.morphology.watershed(): Performing watershed segmentation.

Comments: After trying all previous algorithms it's obvious that morphological chan-vese is the best so far as it gets good results quickly and doesn't affected much with outliers. It's also worth noting that thresholding seems to be affected by the object in the image as different threshold values suits different objects which did happen in algorithms while tuning them. Some edges may be irrelevant to the main object of the image however, they very strong that they might lead to the inclusion of unwanted regions.

4.2.2 Face Segmentation

To do face segmentation, we first detect faces using Haar Cascade classifier.
We have also attempted to re-implement GrabCut in two phases:

1. We use 2 Gaussian Mixture Models (GMMs) to get weights, means, and covariances of foreground and background pixels, using number of components (k)= 5.
2. We then use the extracted probabilities from the GMMs to minimize the negative log likelihood energy using a mincut algorithm. This presently takes too much time, and therefore we use GrabCut from OpenCV directly.

Functions used:

1. sklearn.mixture.GaussianMixture: to construct the GMMs, considered a primitive since Gaussian Mixtures were discussed in optimal segmentation.
2. pymaxflow.maxflow: to get the minimum cut and do the segmentation.
3. cv2.GrabCut: currently used to implement GrabCut, irrespective of our implementation.

Comments:

1. We observed that the detector is not very good, and that using a more sophisticated system would probably give better results. The paper does not state what face detector they use.

4.3 Fast Nearest Neighbours

The paper we used specified using PCA algorithm to reduce dimensions of of the patches data. However, it didn't specify what the data which could lead to a lot possible choices like:

- Performing PCA on every channel separately
- Performing PCA on gray scale of the patches
- Performing PCA on flattened patches where all channels are used together

First approach leads to a problem that PCA may result in different dimensions for different channels. Second approach leads to a very bad accuracy and isn't quite meaningful as style transfer is based on optimizing energy difference between both style and output and content and output with respect to all three channels so discarding colors wouldn't be a good idea. Finally the last approach is slower however leads to the best results so far.
Used Functions:

1. numpy.cov(): Computes covariance matrix.
2. numpy.linalg.eig(): Computes eigenvalues and corresponding eigenvectors for a given matrix.

After reducing the dimensions we the paper specifies using a clustering tree to fast finding nearest neighbour they. However it didn't specify exactly which algorithm is used. We found two options which are available in sklearn. First algorithm is KD tree which operates by dividing space on different k-dimensions till it ends with a node of neighbour patches. Second algorithm is ball tree which similar to KD tree however it performs better on higher dimensional data. There is metrics for choice first the accuracy of finding nearest neighbour as both are approximate techniques and second the speed of each one.
In our experiments Both both algorithms performed almost the same. However, sklearn function had a auto mode which chooses between algorithms based on data size and number dimensions. After some trials auto mode got the best results so decided to use it at the end.
Used Functions:

1. sklearn.neighbors.NearestNeighbors(): Takes data set and finds returns nearest neighbour for any given query vector.

2. Sklearn.feature_extraction.image.extract_patches(): Takes image and returns numpy array of patches of a given shape.

Comments: A major issue is the very long time taken in nearest neighbour search in small patch sizes as number patches increases dramatically increasing the data size. PCA is also slow on large dimensions especially that number dimensions is multiplied by 3 for the three channels. After several trials we found that using PCA only on small patches sizes leads to increasing the speed of nearest neighbour search. This results comes from the fact that PCA reduces the dimensions of the data dramatically some times even up to 0.2 or less of the original data. So we settled down on using PCA only on bathes less than or equal to 21*21. Bigger batches are take very long time to perform PCA on them compared to using nearest neighbour search directly.

4.4 Style Transfer

Algorithms used:

1. Gaussian Pyramid Construction
   The content, style, and segmentation masks are scaled and smoothed so that the image can be processes over multiple scales.

2. Multi-scale Optimization and Dynamic Padding
   Operating on the Gaussian pyramid and with variable patch sizes, we find the nearest neighbors for a collection of patches from the input image X in the style image. To avoid side artifacts, we pad the image with appropriate padding such that the current patch size is a multiple of the total. We tried padding the whole image first but the results were not visually pleasing.

3. Iterative Recursive Least Squares
   Solving a weighted least squares energy optimization problem is equivalent to aggregating patches from the style image in the result image based on the weights.

4. Content Fusion and Denoising
   The content image is restored selectively in areas determined by the segmentation mask then denoising is applied. The entire algorithm is iterative and is applied many times before convergence.

Comments: The quality of style transfer is severely limited by how well the segmentation mask captures the original image, and because of the many parameters the user might be confused.

5 Denoising

We use the domain transform for edge-aware image and video processing. We first apply the domain transform to the image, then use the discrete Recursive filter for smoothing.

1. Use finite difference to get partial derivatives w.r.t x, y.

2. I'_k = sum over x, y.

3. Get horizontal and vertical derivatives to transform the domain according to the equation: $\text{ct}(u) = 1 + \frac{\sigma_s}{\sigma_r} \times I'k$, σ_r and σ_s are parameters entered by user to set the level of smoothing where σ_r represents variance over the signal's range and σ_s represents variance over the domain.

4. 2 passes on 1-d filter are done. First a horizontal pass then transposing then a vertical pass then transposing again, done in one iteration to prevent visual artifacts (stripes).

5. Recursive filter is used for 3 iterations. Done through the following equation:

$$J(n) = I(n) + \text{var}(J(n-1) - I(n))$$

   Where $a = d^{\frac{-\sqrt{2}}{h}}$, var = power$(a, d)$, Abs_der is the distance between neighbor samples in the transformed domain. A symmetric response is achieved by applying the filter twice: left-to-right and then right-to-left.

Comments: The quality of denoising is heavily dependent on the parameters. If σ_r or σ_s are too large, all the details are lost. If they are too small, the resulting image lacks coherence.

	Strengths	Weaknesses
Color Transfer	Fast. Requires no parameter tweaking.	
Edge Segmentation	Morphological Chan Vese is accurate and does not require much user tweaking	Other segmentation algorithms requires significant user tweaking. Chan vese is slow
Face Segmentation	Relatively accurate with frontal selfies (most common)	Detection is very fuzzy for faces that are not direct and frontal.
Nearest Neighbors	Approximate search works well esp. using PCA.	Using a more performant library in a language such as C++ could improve performance drastically.
Style Transfer	Relatively quick, requires no pretraining, user has many 'knobs' to turn until desired result.	The quality of style transfer is severely limited by how well the segmentation mask is done, in constrast, in CNN approaches no user interaction is required.
Denoising	Fast, induces prior without need for slow bilateral filter.	Denoising quality heavily dependent on parameters that user may not enjoy tuning.

6 Experimental Results and Analysis

6.1 Test Cases

We include faces, natural scenes, and artificial houses as content images. See Figure 3 and Figure 4. For style images, see Figure 5 and Figure 6. Successful transfer cases are shown in Figures 7, 8 and 9. Failure cases due to various reasons are shown in 10 and 11.

7 Work Division

Ahmed Khaled	IRLS and Patch Aggregation, Optimization, Mincut
Zeinab Rabie	Color Transfer, Edge Segmentation
Sara Maher	Denoising, Gaussian Mixture Models
Abdelrahman Mohamed	Edge Filling, PCA, GUI

References

[1] M. Elad and P. Milanfar. Style Transfer Via Texture Synthesis. *IEEE Transactions on Image Processing*, 26:2338–2351, May 2017.

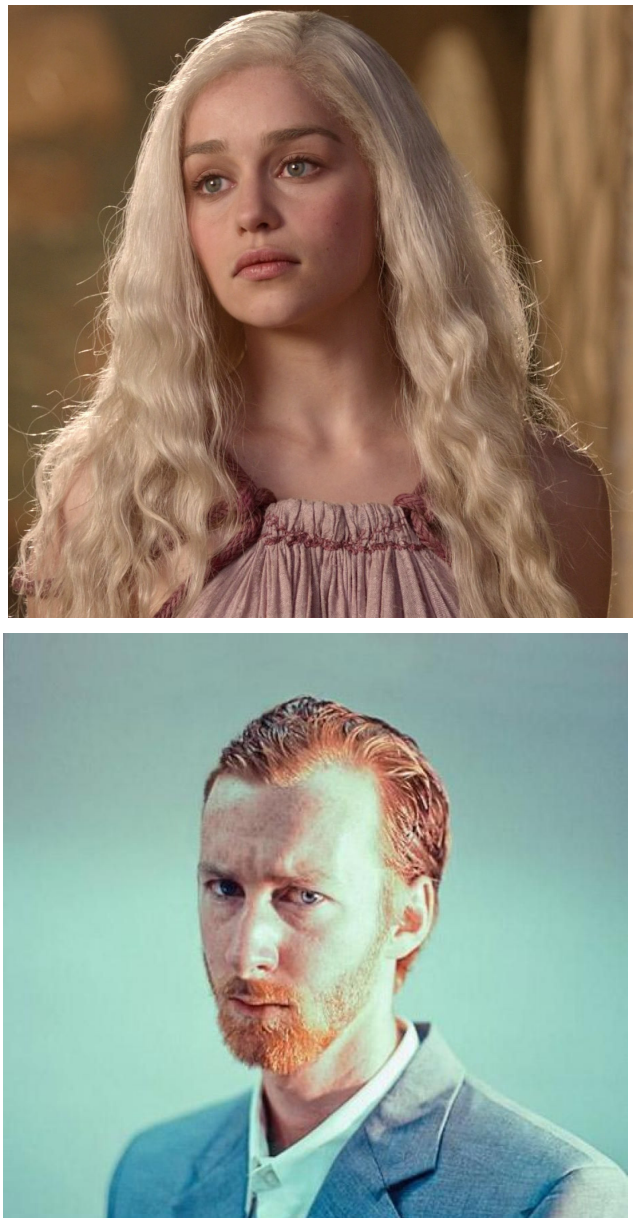

Figure 3: Some Content Images

Figure 4: More Content Images

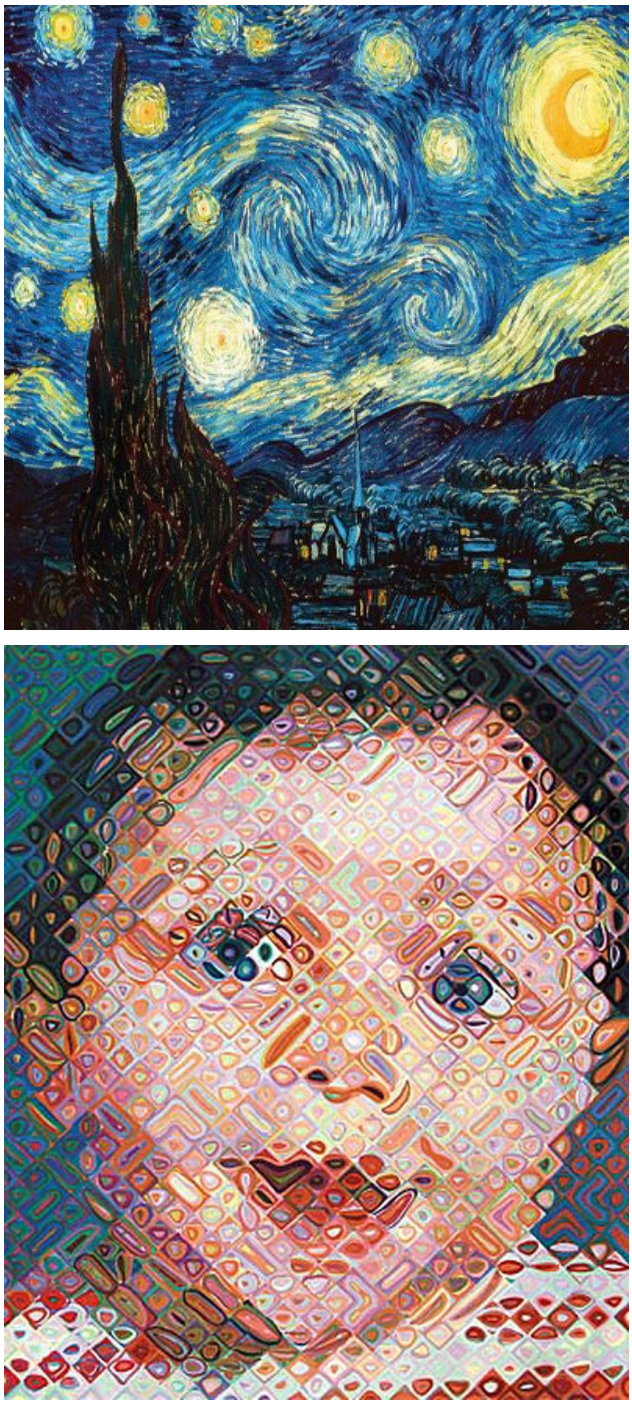

Figure 5: Some Style Images

squares.jpg 

Figure 6: Some Style Images 2

Figure 7: Some Successful Transfer 1

Figure 8: Some Successful Transfer 2

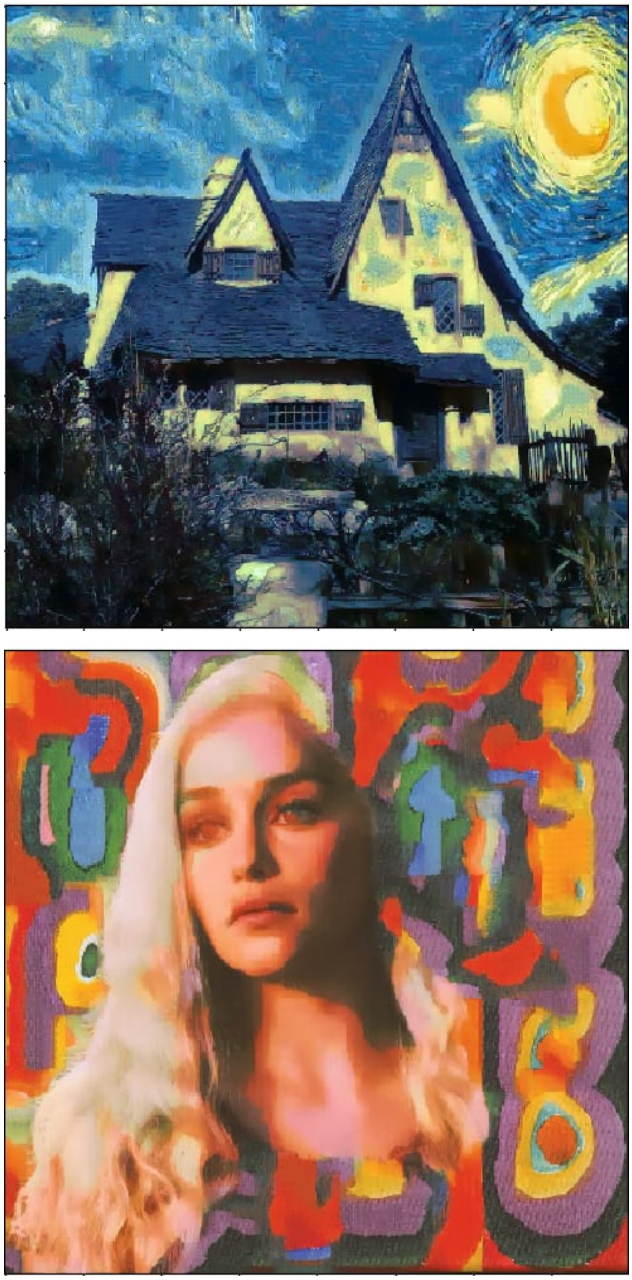

Figure 9: Some Successful Transfers 3

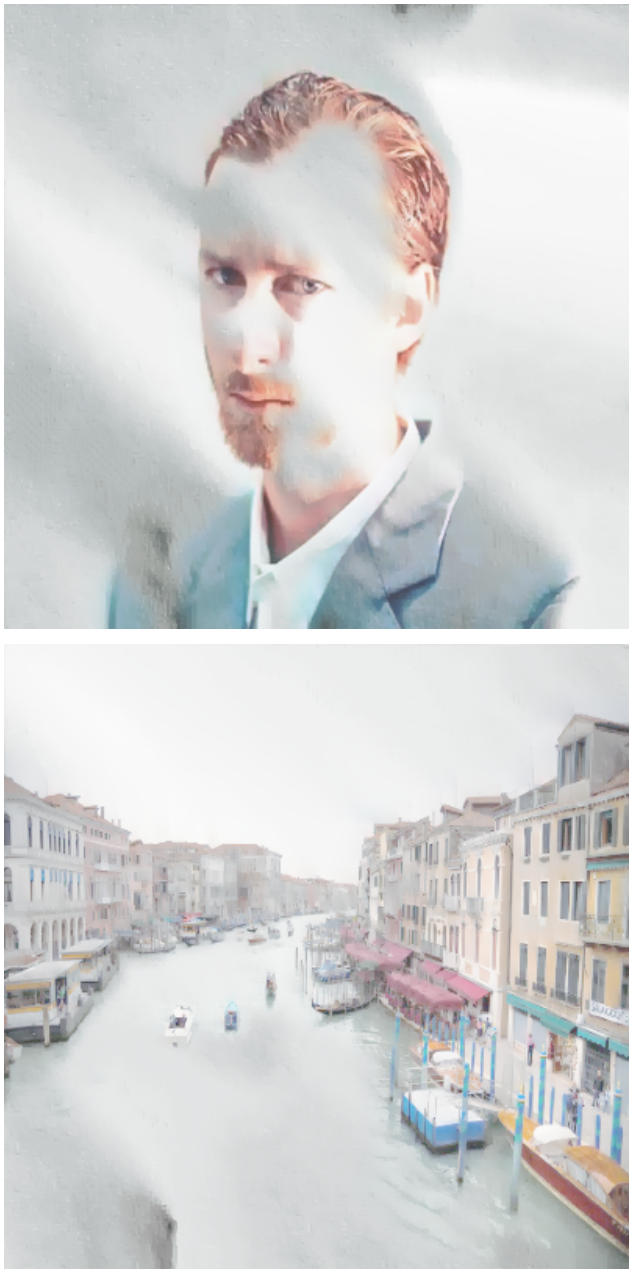

Figure 10: Some Failed Transfers Due to Poor Match

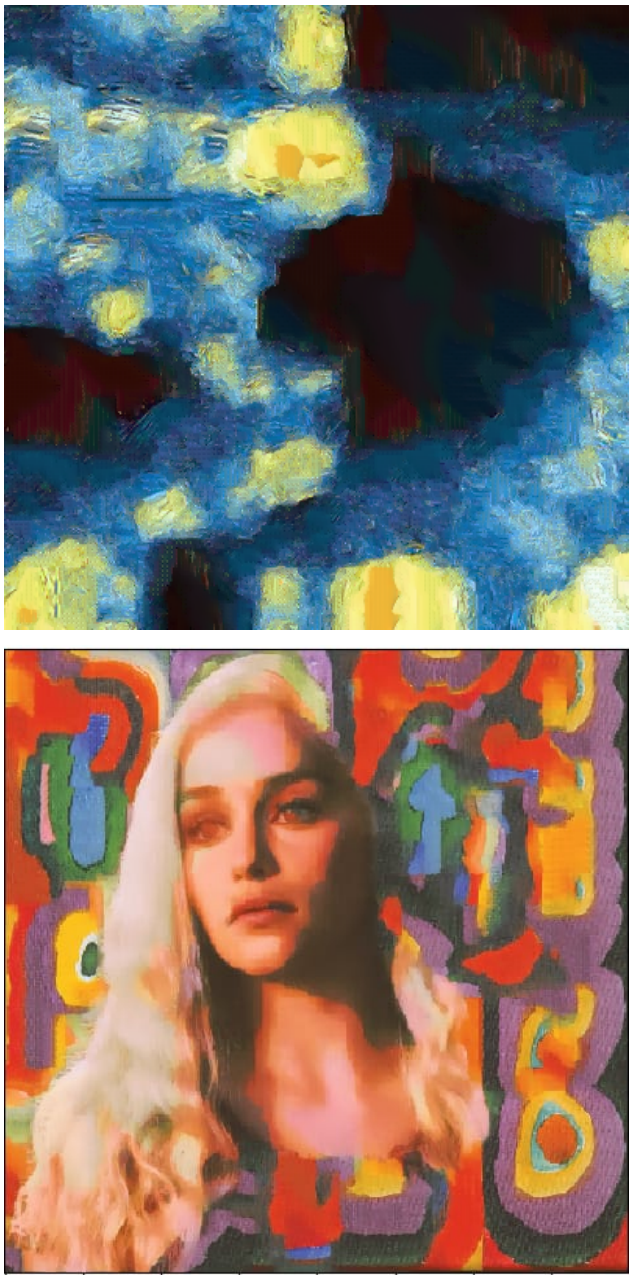

Figure 11: Some Failed Transfers Due to Bad Segmentation