Style Transfer

Final Project Report

Computer Imaging

CS 291A

Kruthika Koratti Sivakumar

Nikhil Bhardwaj Balaji

Appannacharya Kalyan Tej Javvadi

INTRODUCTION

Style transfer is the technique of recomposing images in the style of other images. The motivation for this project came from the paper, "Image Analogies". The paper attempts to solve the following problem.

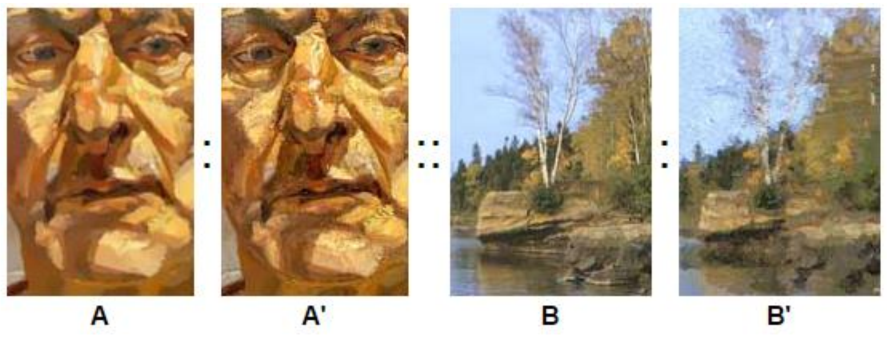

Fig 1. Image Analogies

Given *A* and *A'* (filtered version of *A*), the paper proposes a method to guess the filter which is responsible for the transformation. We then apply the filter to any input image *B*. This problem is illustrated in Fig 1.

Prisma (a mobile application), transfers the global style of a painting to any other image. Given that any filter when applied upon an image, has a global effect on the image too, we found the idea of Prisma to be similar and interesting. Fig 2 shows *an example of what Prisma does.

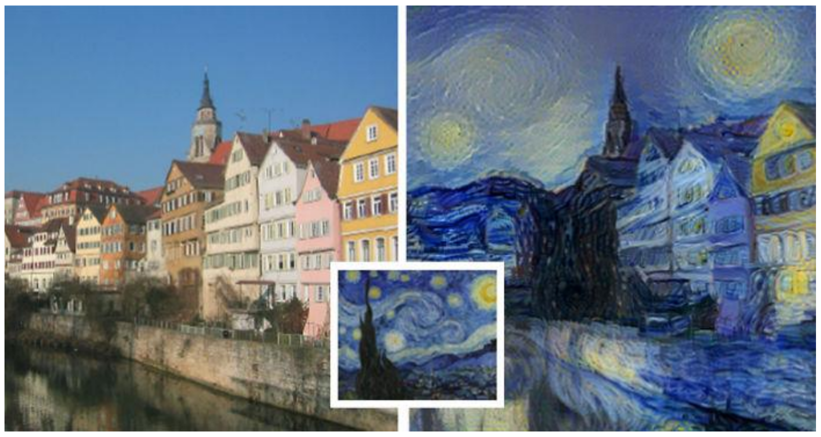

Fig 2. Prisma

This implementation of transferring the global style to any image, without taking 'before' and 'after' pictures of the painting was very intriguing. We saw that the algorithm used by the paper works on an inherent aspect of separating the style and the content of the image. Our idea was to take an input image (headshot portrait of a person without makeup) and another example image (headshot picture of a person with makeup) and using the same algorithm, produce an output image in which the style (makeup, in our case) of the example image is transferred to the input image.

GOALS

Our initial goals:

1. **Style Transfer:** Given a style image, transfer its style to any other image using a deep learning network.
2. **Filter Prediction:** Given an image which has already been passed through some unknown filter, transfer the style from this image to any other image.
3. **Makeup Transfer:** Transfer make-up from one person to another. We aimed at automating the entire algorithm, unlike the authors of the paper, who used some kind of user intervention in their algorithm.

We aimed at creating an android application combining all these aspects.

STYLE TRANSFER

In the first part of our project, we use the deep learning algorithm specified in [1].

Crux: When you forward-pass an image through a neural network which is trained for object recognition, some layers give out the characteristics representing the content of the image, while some other layers give out the characteristics representing the style of the image.

We implemented the algorithm in TensorFlow. Below is a detailed description of the algorithm.

Details of the neural network used:

1. Name of the network - VGG-NET (VGG stands for Visual Geometry Group which is their group name).
2. Number of layers – 19
3. Dataset Trained on – ImageNet database.
4. Layer at which content characteristics are obtained – (‘***relu4_2***’).
5. Layers at which style characteristics are obtained – ('***relu1_1***', '***relu2_1***', '***relu3_1***', '***relu4_1***', '***relu5_1***').

The implementation is as follows:

1. Input images:
    a. *A* – content image (image to be altered)
    b. *B* – style image (image from which style needs to be transferred)
2. Download the trained VGG-NET model (this will contain the weights of the trained result)
3. Forward-pass the content image through the neural network. The output of the layers at which the content characteristics are given out are stored in ‘***content_features***’
4. Forward-pass the style image through the neural network. The output of the layers at which style characteristics are given out are stored in ‘***style_features***’
5. Assign ***alpha*** and ***beta*** where alpha is the weight given to the content loss while beta is the weight given to the style loss.

6. Take an image from which the algorithm must start training for the style transfer. There are three options for this ‘***initial_image***’ –
    a. Random noise
    b. A (content image)
    c. B (style image)
7. The following steps are iterated for over 1000 times to get the final image with the content from A and style from B.
    a. Pass the ‘initial_image’ through the neural network.
    b. Take the output from the content layers. Take l2_norm of the difference of the output from the content layers and the ‘***content_features***’. Call this ‘***content_loss***’.
    c. Take the output from the style layers. Take the l2_norm of the difference of the output from the style layers and the ‘***style_features***’. Call this ‘***style_loss***’.
    d. There will be a regularization term which is computed based on the image size and the shape. Call this ‘***reg_loss***’.
    e. Compute “***total_loss***” = ***alpha **** ‘***content_loss***’ + ***beta **** ‘***style_loss***’ + ‘***reg_loss***’.
    f. ‘***total_loss***’ is minimized with the ‘***Adam***’ optimizer and the ‘***initial_image***’ is updated with this back-propagation.

Results

If we take the ‘***initial_image***’ as random noise image, we get different set of images each time we run the above described algorithm for 1000 iterations. This takes around 33 minutes. Even for 1000 iterations we wouldn’t get a decent image.

When the ‘***initial_image***’ is the style image, the content from the content image wasn’t getting stored in the output. Even after 1000 iterations, the output remains very similar to the style image.

When the ‘***initial_image***’ is the content image, we get decent results (with a blend of content and style) 200 – 300 iterations which takes, roughly, 6 minutes. Figure 3 shows these results.

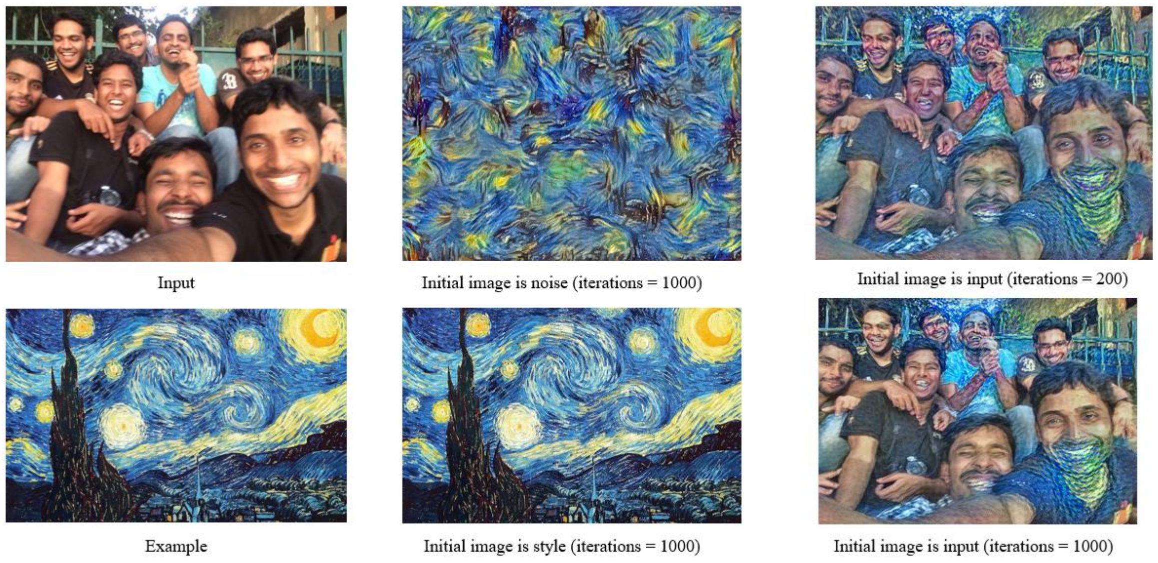
Input
Initial image is noise (iterations = 1000)
Initial image is input (iterations = 200)
Example
Initial image is style (iterations = 1000)
Initial image is input (iterations = 1000)

Fig 3: Results of style transfer

FILTER PREDICTION

Style transfer with simultaneous filter estimation

Given a non-photorealistic image (***NP_I***, which is the filtered output of some image ***I***, because of an unknown filter contributing a style ***S***, or some unknown style), our aim is to estimate the style and transfer it to any other image. From [1], we knew for a fact that the style layers and content layers of ***NP_I*** will contain the style from ***S*** and the content from ***I***, respectively. Hence, all we had to do to estimate the style from ***NP_I*** was to get the output from the style layers of the object-recognition-trained neural network. Thus, for solving this problem, there was no change in the algorithm mentioned above, except for fine tuning a few parameters (the weights for style loss and content loss, the number of iterations).

Results

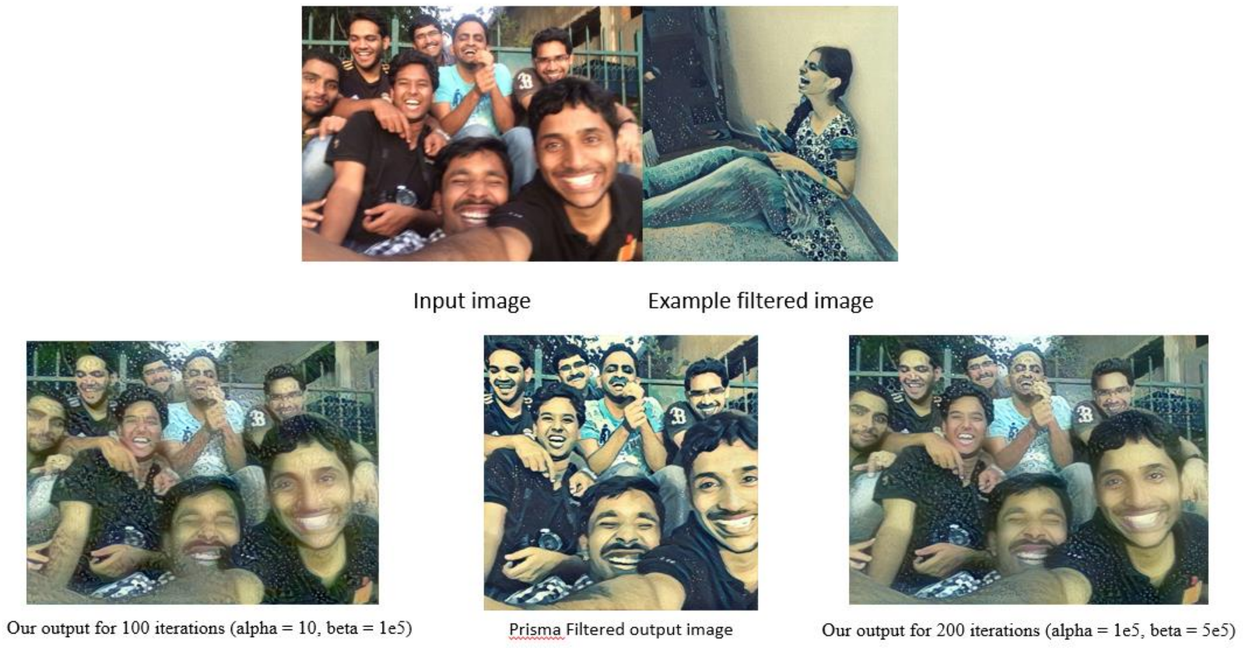
Input image
Example filtered image
Our output for 100 iterations (alpha = 10, beta = 1e5)
Prisma Filtered output image
Our output for 200 iterations (alpha = 1e5, beta = 5e5)

Fig 4. Results of Filter Prediction

Observations

Roughly, our implementation replicates the style of the filtered image. It does not achieve high contrast but, as stated before, it achieves the style of the filtered image (please see Fig 4).

Make-Up Transfer

The third part of our project was to extend the MIT style paper for transferring makeup. Below (Fig 5) is a brief illustration of what this paper aims at achieving. Given an input image (a casual headshot picture) and example image with a target style, the algorithm produces an image with the face from the input and the style from the headshot portrait.

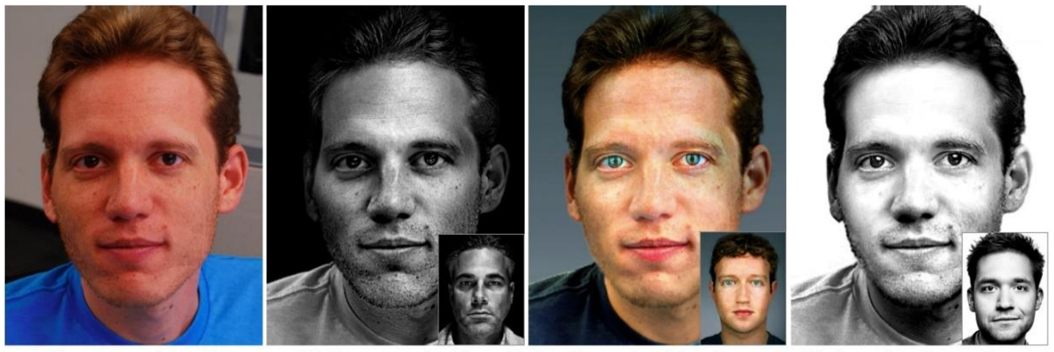
(a) Input: a casual face photo
(b) Outputs: new headshots with the styles transferred from the examples. The insets show the examples.

Fig 5. Illustration of style transfer for different example images

The algorithm is as follows:

1. *Dense Correspondence* – Facial landmarks are used to align the example image to the input image. SIFT Flow is also run to get finer alignment.
2. *Multiscale Transfer* –
    a. *Multiscale Decomposition* – Laplacian stack of input and example is created without scaling down the images. The variance of the Gaussian is increased by a power of 2 for every level.
    b. *Local Energy* – Power maps at each level of the Laplacian stacks are computed.
    c. *Robust Gain Transfer* – Gain transfer from the morphed example to input is carried at each level of the Laplacian stack forming the output Laplacian stack given by

    $$L_l[Output] = L_l[Input]\ x \sqrt{\frac{S_l[Warped\ Example]}{S_l[Input] + \epsilon}}$$

    where ϵ is a small value (0.01) to avoid division by 0. Also, this gain (the expression under root) is smoothed with a Gaussian and clamped to a max value of 2.8 and a min value of 0.9.
3. *Additional Post-processing* – For transferring the eye highlights and the background which we did not focus on.

Preprocessing required:

1. Creating Masks – this step is to separate the foreground (face) from the background. The author used the masks obtained from alpha-matting.
2. Obtaining facial landmarks – this involves running a face detection algorithm which gives out the important facial landmarks (66) in total. These facial landmarks are used to morph the target style image to the input image which forms a core part of the algorithm.

3. The face should be approximately at the center of the image.

Implementation

1. *Crop and Resize* – Given any input/example image, they must be cropped such that the face is at the center of the picture. This is achieved as a Python script which can be called during runtime using MATLAB's system command.
    a. Running dlib on the image gives us the facial landmarks. The y-coordinate of the chin will give us the height of the target image.
    b. Keeping track of the landmarks related to the nose would give us the mid_x and mid_y coordinates of the face in the image.
    c. Finding contours on the image using cv2.findContours. By iterating over the coordinates of the contours one can find the coordinates at the left ear, right ear and the upper outline of the forehead in the image.
    d. We utilized the data downloaded from [2]'s website. This data has all the faces centered. So, we ran a python script which computes the ratios, $\frac{mid_x - leftear_x}{mid_x}$ and $\frac{rightear_x - mid_x}{width_{image} - mid_x}$. Average of these ratios were computed for 52 images from [2]'s dataset. These were found to be … Then, the image is cropped accordingly and saved.
    e. Control returns to MATLAB and the images are read and resized, resizing the image with the largest number of elements (found by numel in MATLAB) to the size of the image containing lesser number of elements.
    f. These outputs are also saved.
2. *Creating Masks* – Getting output from alpha matting involves two main methods:

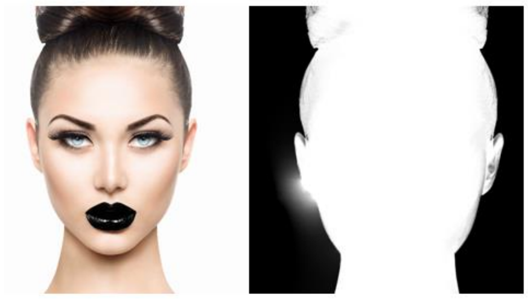
Fig 6: Alpha matte using user scribbles

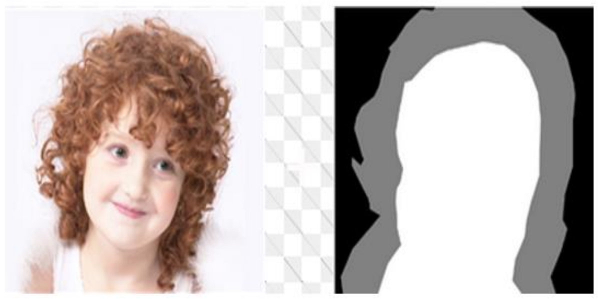
Fig 7: Example of trimap

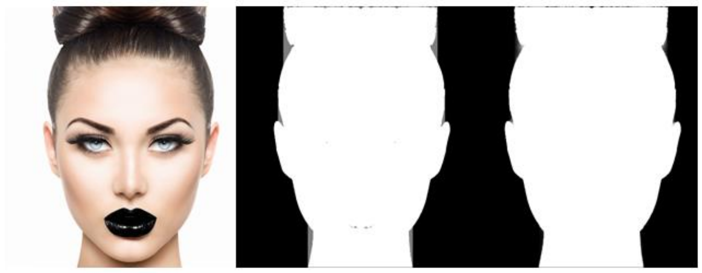
Fig 8: *(Left to right):* Input, Generated trimap, generated alpha matte

a. User Scribbles (Fig 6): The user must map mark the regions he is sure of being foreground and background. We found a MATLAB implementation for marking these points and generating a matt.
b. Using trimaps (Fig 7): A trimap maps three regions – definite foreground (filled in white, 255s), unknown (filled in grey, 127s) and definite background (filled in black, 0s).

Generally, the output mattes are better when they are computed with the help of respective trimaps. If observed carefully, it is quite difficult to generate the trimap, even with user intervention. However, we still tried automating using the below described steps:

a. The facial landmarks are computed. From them, min_x, min_y, max_x and max_y coordinates are computed.
b. The contours for the image are computed and min_x, min_y, max_x and max_y are computed/updated.
c. These 4 coordinates are used to initialize a rectangle in a format OpenCV recognizes. Then, Grabcut is performed over the image using this rectangle, which gives an approximate foreground estimate. All elements in the foreground estimate are made white (filled with 255s).
d. The contours computed in *b* are looped over forming two dictionaries, dict_x and dict_y. Dict_x (dict_y) has the x (y) coordinates of the contours as keys and the list of y(x) coordinates forming contours with that x-coordinate (y-coordinate) as value.
e. For each x (y) coordinate in dict_x (dict_y), the corresponding pixel range varying from minimum to maximum in the list associated to x are filled with 127s. The regions which are already 255s from the grabcut algorithm are left as such.
f. The generated trimap is fed into the matting algorithm, which is modified to read 255s as foreground and 0s as background, giving out an alpha matte.
g. Fig 8 depicts these techniques.

3. *Facial Landmarks* – MATLAB's system calls a python script which writes the facial landmarks to a file.
4. *Rest of the algorithm* – Code available on the author's website was used for the algorithm. However, better results for warping were obtained when the interpolation function was changed to 'spline' instead of linear.

Initials Results

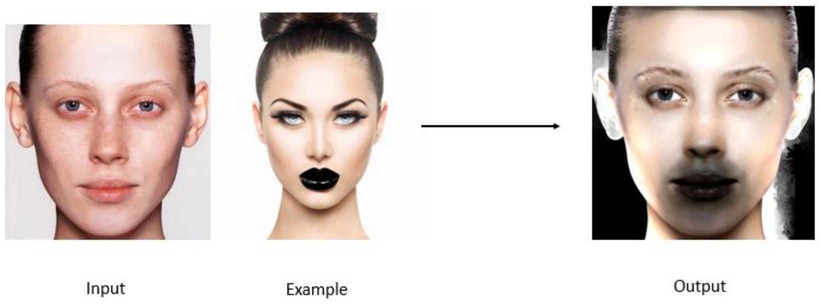

Fig 9: Output (MIT Style Transfer Paper)

What went wrong?

When we observed the results obtained from the above described implementation, we noticed that the results were not satisfactory. We have included one of our results in Fig 9. We noticed that in the final output image, the lipstick is smeared.

The reason why this algorithm does not work:

Creating a Laplacian stack is an integral part of the algorithm. A Laplacian is created by smoothing out the image with a Gaussian of variance (which acts as the spreading factor) depending on the level of the image. Gain maps are formed with these images. Since at each level, we use a different Gaussian function (that varies in standard deviation by a power of 2), the final output, which is formed by summing up images at all levels, has a spread of standard deviation from all the levels, because of which we have the smearing effect in the region of interest.

We started looking for ways to fix this issue and came across this state-of-the-art makeup transfer [3]. Their algorithm decomposes the input and the example images into the LAB color space as done by [2] and treats each channel separately. Their logical reasoning for treating each channel separately, made a lot of sense in solving the problem at hand.

Their algorithm is explained below (Fig 10):

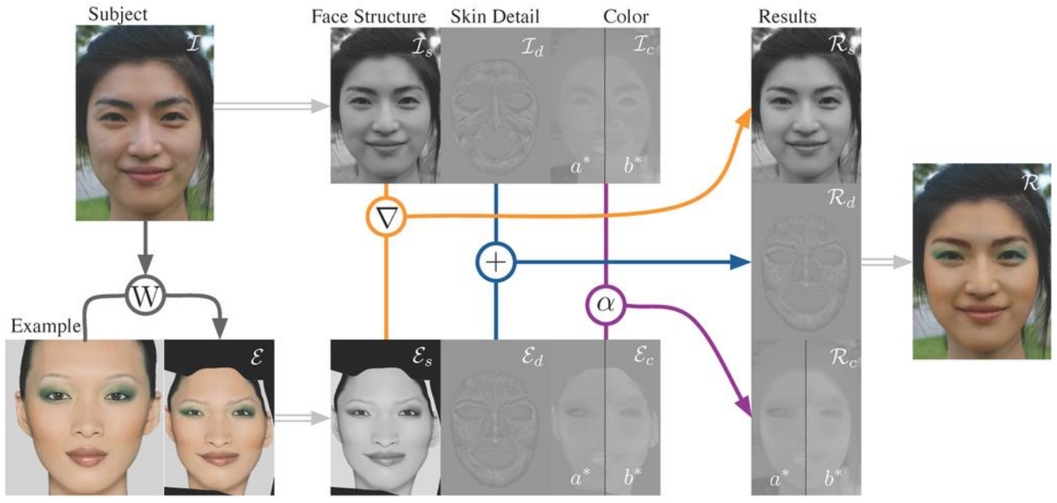

Fig 10: Algorithm of [3]

1. *Create face masks* – β maps for input and example are created as,
$$\beta(p) = \min_q \left(1 - k(q).e^{\frac{-(q-p)^2}{2\sigma^2}}\right)$$
where q indexes the pixel over the image, k(q) is 0.7 for eyebrows, 0 for skin area and 1 for other facial components.
2. *Warp* - Facial landmarks are collected for both images. Then, example image is warped onto the subject image.
3. *Layer decomposition* - input and example images are resolved into LAB color space. For each image, its *L* layer is divided into structure layer and skin detail layer.

   a. Face Structure is obtained by performing bilateral filtering on the *L* channel. This layer will contain the highlight and the shading effects of the make-up.
   b. Skin detail is obtained by subtracting the Face Structure from *L*. This contains the skin color of each image.
4. *Color Transfer* - result's color layers (*a* and *b*) are formed by alpha blending the input and the example image's *a* and *b* layers.
5. *Skin Detail Transfer* - result's skin detail layer is formed, ideally, by a weighted sum of the input skin detail layer and example skin detail layer. Since we need the skin detail to be like the example image, we make the weight of input skin detail layer, 0.
6. *Highlight and shading transfer* - The Result's structure layer is computed by Poisson blending the structure layers of input and example, respectively.
7. Finally, the result image is converted from LAB space to RGB space.

Implementation

1. *Creating face masks* – Implemented in python and the script is called during run-time using MATLAB's system command.
   a. Extracted 77 facial landmarks using STASM library.
   b. Formed OpenCV contour structures for facial components (ex: eyes, nose, etc.).
   c. The skin region is filled with 255, eyebrows were filled with 127 and facial components were filled with 0 using cv2.drawContours function. This mask is saved.
   d. Control goes back to MATLAB. The saved mask is read. *K(q)* is formed by making all 255s, 0 and scanned element-by-element, row-wise. The code takes a neighborhood of 5x5 pixels around the current pixel and forms the beta map using the equation specified above.
2. *Warp* – Implemented in python. Called in runtime using system command of MATLAB.
   a. Extract 77 facial landmarks using STASM library and store them in a file.
   b. Control goes back to MATLAB. Landmarks are read and the example is warped using 'spline' interpolation technique.
3. *Layer Decomposition* – achieved with rgb2lab of MATLAB. The structure layers are obtained using a bilateral filter code available online on MATLAB CENTRAL.
4. *Color Transfer* – alpha-blended the input and example color channels.
5. *Skin Detail Transfer* – obtained the example's skin detail.
6. *Highlight and shading transfer* – the idea was to edit the layer in the gradient domain and transfer the texture in the Laplacian stack which is a gradient domain [2]. So, we used their implementation to solve for the structure layer.
7. The result is formed by lab2rgb function in MATLAB.

RESULTS

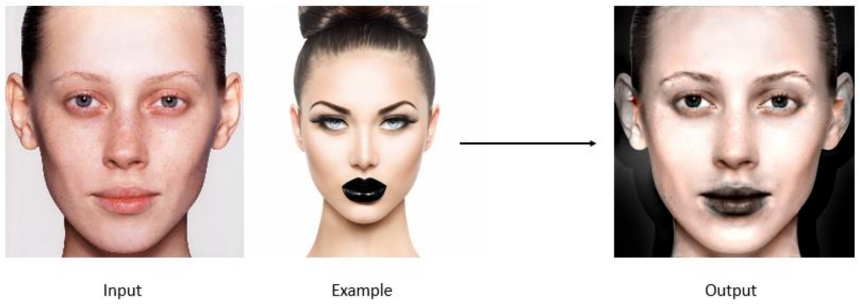

Fig 11: Output of Makeup Transfer (with user intervention)

Results of Makeup Transfer (without user intervention)

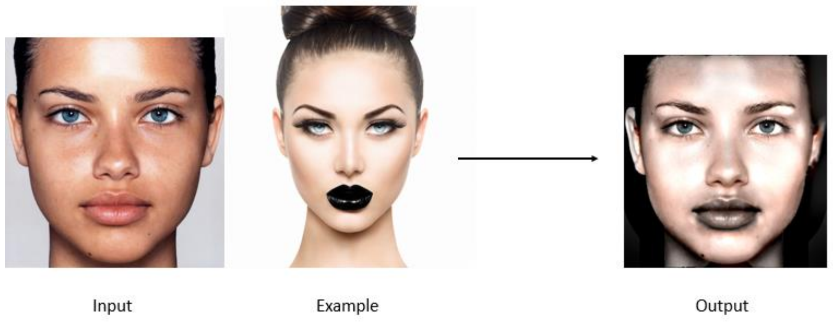

Fig 12(a)

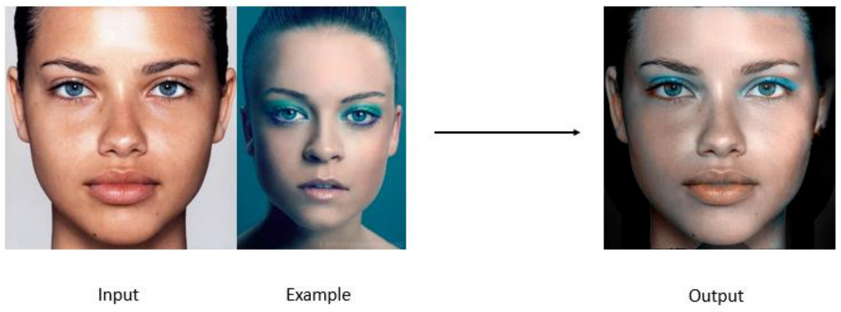

Fig 12(b)

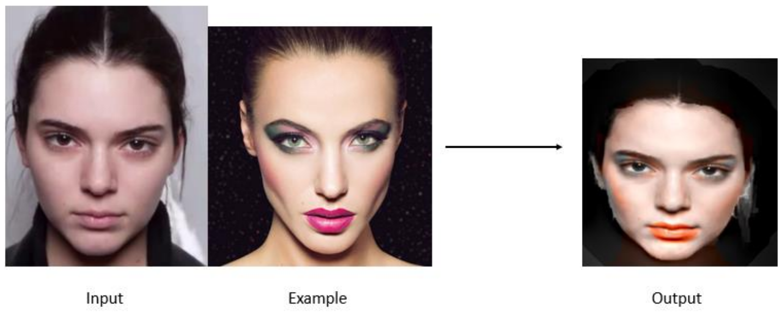

Fig 12(c)

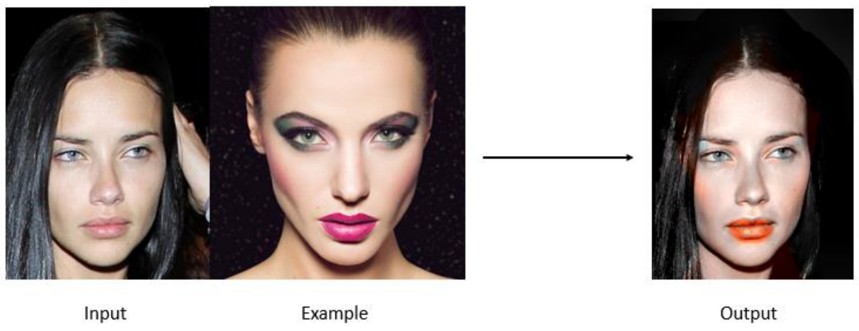

Fig 12(d)

The most common artifact is the color of lips in the output image. It does not match that of the example image. This is because, the output's color channel is not formed directly from the example but as a weighted sum of the input and example is taken. Also, the way we solve for the structure layer in the output is different from that proposed by [3].

Another common artifact is that the lip stick is, sometimes, not aligned exactly on the lips of the face in the output image. This is due to a warping artifact.

Another artifact is that sometimes (as is the case in 12(d)) the blush gets copied on to the hair of the output. This is due to warping. The facial landmarks at the cheekbones are under the hair. Thus, the makeup which has to get applied onto the cheekbones finds itself on the hair.

ANDROID USER-INTERFACE FOR MAKEUP TRANSFER

We started out with the User Interface required to implement our idea as an Android App. We defined two activities:

- MainActivity: An activity to select input and example images using buttons that open either gallery or the camera, and to display selected/captured images.
- ResultView: An activity to display the final makeup transferred output image on the screen.

Fig 13 is a series of screenshots of our app.

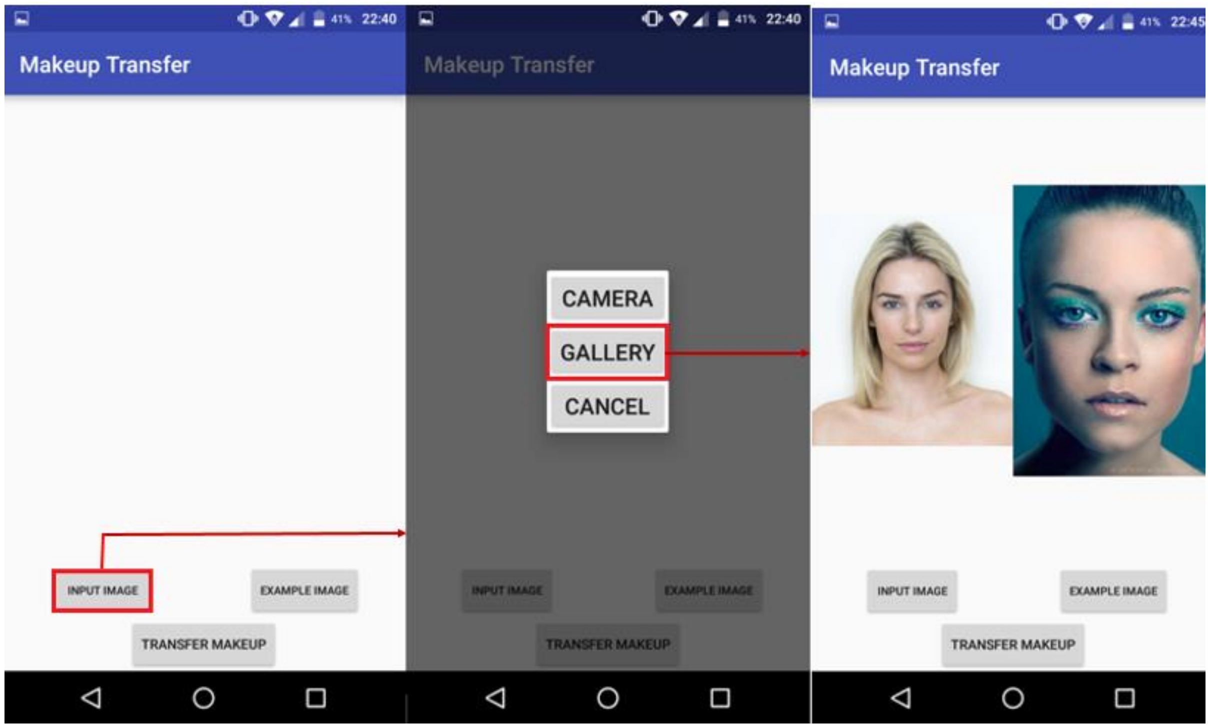

Fig 13: Screenshot of the user-interface of our app

Communication with server:

We first set up Apache2 and PHP5. We first check if the connection is working by entering open localhost in a browser on the server. We then write the PHP file for uploading pictures onto the server. Once the "Transfer Makeup" button is touched, android connects to the server (the PHP file is executed) and the input and example files are uploaded to a specific folder on the server. In the PHP file, we call a MATLAB loop function. This function keeps checking if a particular folder called "check" exists in the directory. When the PHP file is executed and the two input pictures are uploaded, a new empty folder called "check" is created and the MATLAB loop function starts executing. In this loop function, we call another MATLAB function which implements the core algorithm (makeup transfer) using VLFeat. Once this function is executed, the output picture is downloaded back to the android device and displayed in the ResultView activity. Fig 14 is a flowchart illustrating this process.

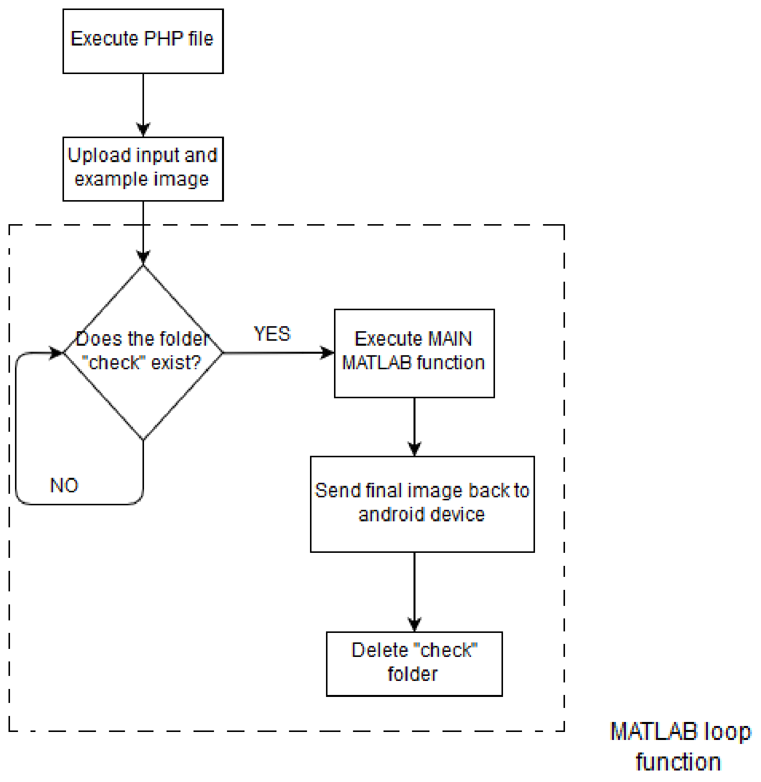
Execute PHP file
Upload input and example image
Does the folder "check" exist?
YES
NO
Execute MAIN MATLAB function
Send final image back to android device
Delete "check" folder
MATLAB loop function

Fig 14: Flowchart illustrating communication with server

We could upload images and execute the MATLAB file. But we were facing issues with downloading the image back to the android device.

CHALLENGES

1. The toughest part of the project was to integrate all modules into one single Android App. We could upload pictures onto the server but could not get the download part working. It was mainly because we had to integrate the PHP, MATLAB and Python, all together into one working model. Since all three of us were beginners in Android, we could not figure out a solution within the given timeframe.
2. When the makeup transfer algorithm was giving us bad results with the initial implementation, we had to make changes to our alpha matting algorithm. Instead of taking the input image and directly generating the alpha matte, we first generated a trimap and used both the input image and the trimap to compute alpha matte. This took some time for us to implement.
3. In the original implementation of the MIT style transfer paper, a lot of preprocessing was done manually (getting alpha matte and facial landmarks were generated independently for every image). We had to automate the entire process right from taking input images.

ASSESSMENT

We documented our results for style transfer, filter prediction and make-up transfer along with a set of qualitative questions about our output images and sent it to a group of people. We have listed out the questions below.

1. Describe in detail the "style" that you notice, is getting transferred (if any).
2. Do you see anything unusual or out of place in the final output?
3. Do you think the output is the same as the input image in the style of the example image?
4. How natural/artificial does the output look?
5. How would you rate this work on a scale of 1 to 10?

Our subjects:

1. Feedback from 10 people
2. 9 business majors and 1 computer vision major
3. Age group: 20 – 25 years
4. All tech savvy

Feedback:

1. They all noticed that the style of the example image was getting transferred onto the input image. They were satisfied with the Prisma filter estimation and style transfer. They thought the makeup transfer results were reasonable and convincing, but also pointed out that there were a few artifacts (they thought "the pictures looked odd at a few places").
2. Every one of them noticed artifacts and thought the final pictures could do with some refinement.
3. All: Yes
4. They thought that the style transfer and Prisma results looked extremely good. For makeup transfer, they commented that the final picture looked artificial and distorted in some regions. They also noticed how the background of the input image is lost in the process.
5. Computer vision major – 10.
   3 others – 5
   3 others – 7
   1 other – 8
   2 others – 10
   Average score: 7.4/10

CONCLUSION

This project implements style transfer and makeup transfer (as an extension of style transfer, as described in [2]). We also tried to reproduce the working of the popular image editing app, Prisma and extended it to the case of filter estimation, given an input image and a filtered image. We tried to do this in an Android environment and to be honest, it was a gamble since none of us were experts in Android. But still, we ended up being partially successful as we got the process of uploading the input image to the local server working but ran into problems when it came to downloading the output image from the server onto the phone and unfortunately, we couldn’t resolve all of these within the limited time frame. This, on the other hand, presents an opportunity for future work, namely trying to build a fully functional Android app that implements the two algorithms that form the basis of our work.

There were also less than satisfactory aspects in the results we obtained with respect to the imaging part of our work, like loss of background information in the output and artifacts in some cases. Again, this presents us an opportunity for fine tuning, only if more time was available.

Overall, this project was an immensely educating and informative endeavor for all of us and taught us a lot about the applications of the two algorithms discussed. In the process, we also got an opportunity to learn about relevant research and share the information with other people in the class.

REFERENCES

1. L. Gatys, A. Ecker, and M. Bethge. A neural algorithm of artistic style. Nature Communications, 2015.
2. Shih, Y., Paris, S., Durand, F., and Freeman, W. T.2013. Data-driven hallucination for different times of day from a single outdoor photo. ACM Transactions on graphics (Proc. SIG-GRAPH Asia).
3. Guo, D., and Sim, T. 2009. Digital face makeup by example. In IEEE Conf. Computer Vision and Pattern Recognition.
4. Hacohen, Y., Shechtman, E., Goldman, D. B., and Lischinski, D. 2011. Non-rigid dense correspondence with applications for image enhancement. In ACM Trans. Graphics, vol. 30.
5. Bae, S., Paris, S., and Durand, F. 2006. Two-scale tone management for photographic look. In ACM Trans. Graphics.
6. Reinhard, E., Adhikhmin, M., Gooch, B., and Shirley, P. 2001. Color transfer between images. IEEE Computer Graphics and Applications 21, 5, 34–41.
7. Cohen-or, D., Sorkine, O., Gal, R., Leyvand, T., and Xu, Y.-Q. 2006. Color harmonization. ACM Trans. Graphics 25.
8. Joshi, N., Matusik, W., Adelson, E. H., and Kriegman, D. J. 2010. Personal photo enhancement using example images. ACM Transaction on Graphics (TOG) 29, 2, 1–15.
9. Tong, W.-S., Tang, C.-K., Brown, M. S., and Xu, Y.-Q. 2007. Example-based cosmetic transfer. In IEEE Pacific Graphics.
10. https://github.com/anishathalye/neural-style