Computer Vision: Final Project
Image Overlay

Satyen Singh, Erica Chou, Henry Tse, Abhin Gude
srs833, emc689, ht688, ag8556

Abstract

The creation and use of images that fit over a detected person's face, or filter images, have been becoming increasingly popular in this age of social media. These filters that were popularized by Snapchat and currently used in other social media apps such as Messenger, and Instagram leverage Computer Vision and Image Processing concepts in order to properly align a base filter image over a face. This report will explore these concepts, and using these techniques in Python to create a basic but original implementation of "Snapchat" filters which will be discussed in this report.

Introduction

The main goal of the project is to explore and learn about Computer Vision and Image Processing concepts in order to create an implementation to overlay filter images onto a face within an image. The workflow of the project and report can be separated into two different sections based on the concepts needed to understand the section: facial land marking through deep learning and filter image modification through Affine Transformations. The first part of the project required training and using a Deep Learning model to recognize a face and mark down important facial features with landmarks. The dataset to train the Deep Learning model used 96x96 images that came with a set of 15 landmark points, and the dataset was fed into a Convolution Neural Network (CNN) to train the model to give the model the ability to identify and map landmarks on a person's face. The second portion of the project focused on using Affine Transformations to morph a base filter image to match the part of the face that should properly overlay the filter image onto the face. The transformation was done by finding an optimal fit to transform three or more land marked points on the filter image onto the corresponding landmark points on the face image. The steps in development, results from the "Snapchat Filter" code, and discussions about the strengths and weaknesses of the implementation are included within the report.

Basic Methodologies

Affine Transformations

An Affine Transformation is defined as a linear transformation such as shearing, scaling, and rotation that can also be translated within a space. This type of transformation can be defined with this basic equation:

$$f(\bar{x}) = A\bar{x} + b$$

[Traa] Where $\bar{x}$ represents the points that will be transformed, A is the linear transformation matrix and b is the translation vector. The main property of Affine Transformations is that after the transformation, the transformed image will keep the structure of lines of the original image and will not distort the pixels within the images. To apply these transformations, the matrices were converted into homogeneous coordinates in order to keep the Affine Transformations and translations within the same matrix. Shown below is the structure of the transformation matrix in homogeneous coordinates for 2D images:

$$\bar{x} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad A = \begin{pmatrix} a & b & x_0 \\ c & d & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

Where a, b, c, d represent the variables associated with Linear Transformations and x_0 and y_0 represent the variables associated with translation. The variables a, b, c, d, x_0, y_0 are the variables that have to be solved when transforming the filter images onto a face.

Using Landmarks to Transform Images

Corresponding landmarks were used in order to align the filter images with certain parts of the face, where corresponding landmarks are distinct points within each image among a pair of two images that represent the position of the same feature. For example, Figure 1 shows how landmarks on this nose filter would map to distinct features on this woman's face in the test image. Using these corresponding points, the Affine Transformation Matrix that will transform the landmark positions from one image to best fit the landmark positions of the other image can be found. Because the Affine Transformation happens within 2D space, the transformation matrix has six unknowns; therefore, at least

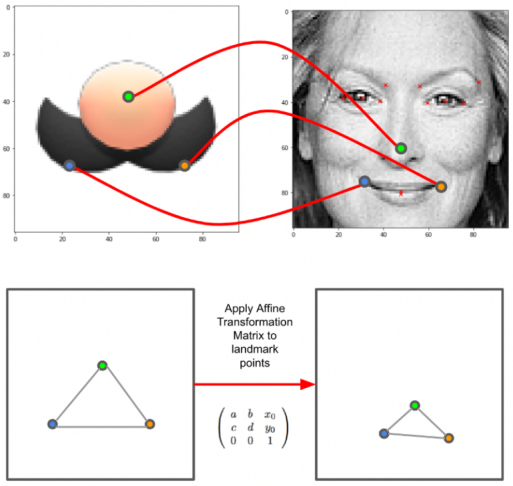


Figure 1: (Above) Image of corresponding landmark points between a filter image and landmark image. (Below) Example of what Affine Transformation is supposed to do to the landmarks on the filter

six points of correspondence, or three landmarks from each image, are needed to find the values of the transformation matrix. More than six points can be used to find the optimal transformation matrix, but this could lead to some of the landmarks not lining up perfectly with the image. However, for fitting filter images over facial landmarks, this is not a problem. In fact, using extra landmarks was necessary due to the nature of some of the filters such as the eyes filter, which needed four points from each image to ensure that the eyes were symmetrical on the person's face.

To find the values for the Affine Transformation Matrix, the Least Squared Error Method was used to calculate the values for all the variables within the matrix.

Least Squares Error

The Least Squares Method is an approximation technique used to find an estimated solution for an equation given a "predicted" matrix and an "observed" matrix, which in this case are the facial landmarks on the face image and filter image respectively. The generalized form of the equation that solution needs to be found for is as follows:

$$Ax = b$$

[Dan] Where A is the observed matrix, b is the predicted matrix, and x is the matrix that has to be solved for. By manipulating the equation above, the value of x can be approximated. This equation shown here is the equation used to calculate the value of x:

$$x = (A^T A)^{-1} A^T b$$

Where, for the case of finding corresponding landmarks, x is the resulting values of the Affine Transformations, A

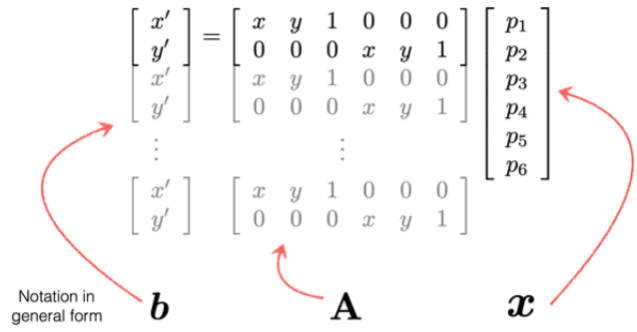


Figure 2: The values and formats of the x, A and b matrices used to find the Affine Transformation values [Kit17]

is the matrix that contains the filter x and y values, or the "source" points, and b is the matrix that contains the values of x' and y' of the face image, or the destination points.

The formatting and matrix sizes of these values that are solved when finding the values of the Affine Transformation are shown in Figure 2. The number of source points and destination points must be equal, and each array for source and destination points must have more than three landmark points.

Backwards Mapping

Backwards mapping, also known as inverse mapping, is a technique used to prevent the loss of pixel details when transforming an image that occurs with forward mapping. This is done by using an inverse transformation from each of the pixels in the new, transformed image to the original image, and if the transformation result does not end up exactly on one of the pixels, the intensity value of the pixel closest to the calculation result is used. Backwards mapping ensures that each pixel in the new image will be assigned a color value from the original image.

Overview of the Pipeline

Data Augmentation

The initial training dataset after getting rid of data that contained null points consisted of 2140 images along with their facial landmarks. This is a small amount of data for training a ResNet neural network. Therefore, we decided to use data augmentation for increasing the training images. We made five times the initial training data which is around 10000 training images. For every image in the training dataset we applied two random transformations of rotation, translation, and two random brightness and sharpness adjustments.
The main challenge of performing the Data Augmentation came from transforming both the image and landmarks. Though transformation of the landmarks was not necessary for intensity and sharpness adjustments, for the rotation and translation of images, the landmark points had to be individually transformed along with the image. For rotation and

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{xx} & a_{xy} & b_x \\ a_{yx} & a_{yy} & b_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Figure 3: [Cos12] Equation for transforming the landmarks during data augmentation

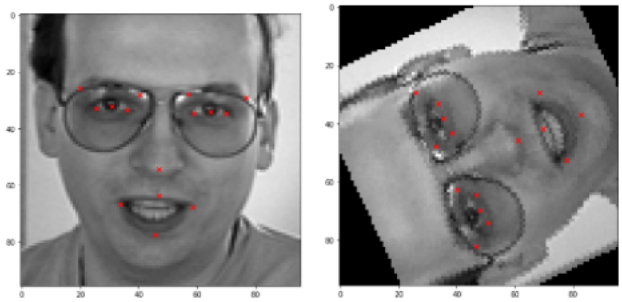


Figure 4: A side by side comparison of an image and its landmarks before and after data augmentation

translating images we used PyTorch transformation functions and landmarks were manually transformed with the help of transformation equations[Trab]. The equation used for transforming the landmarks is given as : Where a_{xx} is $cos\theta$, a_{xy} is $-sin\theta$, a_{yx} is $sin\theta$ and a_{yy} is $cos\theta$. b_x and b_y are the translation shifts in x and y direction respectively.

Training the Deep Learning Model

To predict the landmarks on the face, we used a Residual Network. More specifically, we used a ResNet50 Model. The reasoning for this was that it is a highly popular model for computer vision tasks and it is significantly faster to train on than a VGG model.

One of the biggest advantages of a ResNet model is that it solves the problem of the vanishing gradient. As models train and get deeper, they get to a point where the learning rate degrades and the loss that is back propagated is extremely small or 0. This leads to the model not being able to update itself as it gets deeper. The way that ResNets deal with this problem is by introducing the identity shortcut connection. This identity shortcut layer skips one or more layers and connects directly to the output, increasing the amount of noise in the network.

Besides solving the problem of the vanishing gradient, ResNets also allow for features to be reused which makes them more efficient than other models.

Description of Loss Function We evaluate the performance of our model using the loss function. The loss function, broadly speaking, will tell us how much the predictions from our model vary from the true values. The higher the loss, the less accurately our model is able to predict. Our goal is to reduce the loss function as that means our model is reaching peak performance. For our purposes, the loss function that we used is the Mean Squared Error (MSE). MSE is a highly popular metric for regression problems. Since we are not doing classification and trying to predict a set of points, it is suitable to use MSE. It calculates the squared difference between the true points and the predicted points on the images and averages over all of the points.

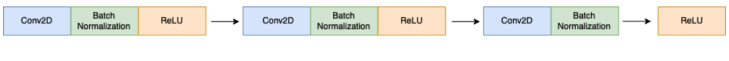
$$MSE = \frac{1}{n}\sum(y - \hat{y})^2$$

Using our loss function, our model is able to update itself in a way that it is able to reduce the overall loss at each iteration. This is done through the use of the optimizer. The purpose of the optimizer is to take the loss generated and use it to update the model weights and improve the accuracy. There are many different kinds of optimizers, but the one that we used was the ADAM (Adaptive Moment Estimation) optimizer. It is an extension of another kind of optimizer known as Stochastic Gradient Descent (SGD) optimizer. It uses the concept of momentum which accelerates the SGD optimizer to reduce loss aggressively in the initial stages of training. ADAM is highly useful for dealing with large datasets or very deep networks.

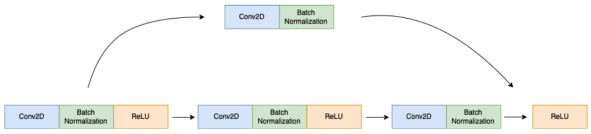
Description of Optimizer Using our loss function, our model is able to update itself in a way that it is able to reduce the overall loss at each iteration. This is done through the use of the optimizer. The purpose of the optimizer is to take the loss generated and use it to update the model weights and improve the accuracy. There are many different kinds of optimizers, but the one that we used was the ADAM (Adaptive Moment Estimation) optimizer. It is an extension of another kind of optimizer known as Stochastic Gradient Descent (SGD) optimizer. It uses the concept of momentum which accelerates the SGD optimizer to reduce loss aggressively in the initial stages of training. ADAM is highly useful for dealing with large datasets or very deep networks. Our result is similar to a snapchat filter where we locate and predict landmarks to add an overlay image onto the face. We worked with still images to look for landmarks such as the forehead, mouth, and eyes. The result is an altered image with an overlay image onto the face at the specified learned landmarks. The facial landmarks were properly learned with high validation and testing accuracy. We were able to achieve our goal and produce good results.

ResNet Architecture

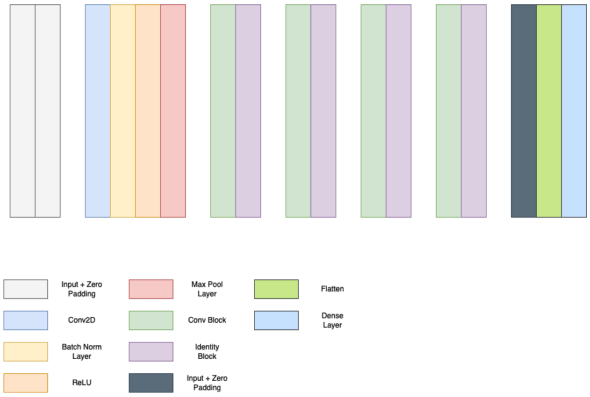
Identity Block



Convolutional Block



Full Network



Preparing and Applying Filters on Face Images using Landmarks

Preparation of the filters and overlaying filters onto the images of faces properly was an essential part of creating these Snapchat filters. The first step of this process was reading and loading in the images as RGBA arrays using the cv2 library, where the RGB and the transparency values, the alpha, were used. The alpha value is especially important for the overlaying step, which will be discussed later. After reading in the images, landmark points were assigned to the filter image through a function that takes in the "type" of image, and the width and height of the original image. The function then returned two arrays, an x values and y values array that keeps the x and y values of the landmark points. These landmark points were manually determined for each filter that can be used based on what facial landmarks would work best to position the filter and where perceptually the facial landmarks would line up on the filter image. The landmark points were also assigned within the function based on which "type" of filter among a set list of filters is called upon, and the function that creates these landmarks is able to scale these landmark points for different image sizes.

Once the landmarks were set, the filter were transformed using Affine Transformations in order to be place correctly on the person's face. To do this, the Affine Transformation matrix between the filter landmark points and the face landmark points were found. This was done by creating a function that took both the filter and the relevant face image landmark points. Using the two sets of landmark points, we used the Least Squares Error method to find the values of the Affine Transformation Matrix that would result in the filter landmark points being as closely fit to the face image

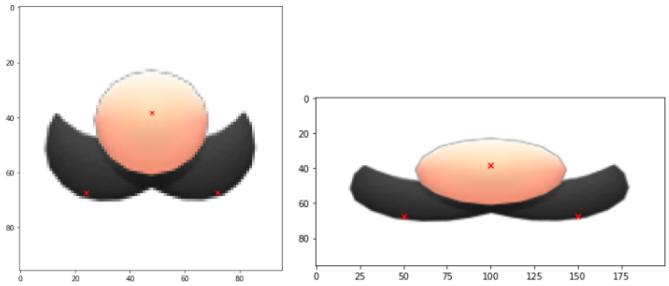


Figure 6: Two Images depicting the Mario "Nose" Filter and the corresponding landmarks associated with the filter. Notice that the landmark positions stay relatively in the same places on the nose and mustache despite the differences in widths of the filter

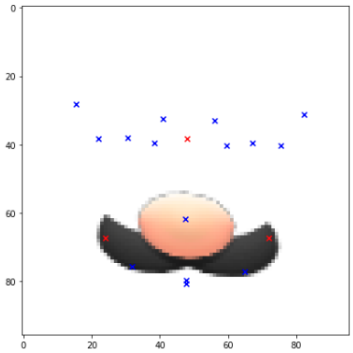


Figure 7: Image of Mario "Nose" Filter after the Affine transformation matrix was found and applied onto the original nose filter image. The red points represent the original landmark positions of the filter while the blue points represent the facial landmarks of a test image

landmark points as possible.

After the values for the Affine Transformation Matrix was found, the filter image, transformation matrix, and the size of the output image is fed into an OpenCV function named cv2.getAffineTransformation(). This function applied the transformation matrix on the source image in this case, the filter image. Using the inverse mapping technique it returns a new image that shows the transformed image. Figure 8 below shows an example of the nose filter shown in Figure 7 after Affine transformations were performed to fit certain landmark points given by a test image's face.

Overlaying the Filter Image onto the Face Image

Overlaying an image involves adding an additional layer to the original test image. The additional layer includes the filter images which align with the landmarks learned on each test image. However, the images cannot be easily combined due to the difference in image contrast. The

background of the filter image is white (low intensity), directly combining the filter image and test image will result in an increase of contrast throughout the entire image called over compositing. To combat this issue, a method called alpha compositing was used. Alpha compositing or alpha blending is the process of combining images with a background to create full transparency.

In order to start the alpha composition process, both the background and overlay images must be of the same shape and color channel. To insure this sentiment we converted our background dataset shape and color channel to the same as our foreground images. We first reshaped the face image into a 96x96 array and an uint8 data type. The uint8 (unsigned integer of 8 bits) data type consists of all whole numbers between 0 to 255. All unsigned integers are non-negative numbers. Converting to the uint8 data type allows for an additional conversion into the RGBA channel used for alpha compositing.

Alpha compositing is the process of associating matte for each pixel along with maintaining its color. The matte layer contains information such as the geometric shape, which allows us to distinguish between the image and the blank background. 2D images have a color combination stored at every pixel. During alpha compositing, an additional array value called alpha channel is stored. This value ranges from 0 to 1, a value of 1 meaning that the pixel is fully opaque, and 0 meaning the pixel is fully transparent as the color beneath the pixel will bleed right through.

Alpha composition utilizes an algorithm called Painter's Algorithm:

$$\alpha_o = \alpha_a + \alpha_b(1\text{-}\alpha_a)$$

$$C_o = (C_a\alpha_a + C_b\alpha_b(1 - \alpha_a))/\alpha_o$$

[Smi] C_o , C_a , and C_b represent the color component at each pixel applied at each color channel on the background and foreground images. Whereas α_o ,α_a and α_b represent the alpha component at each pixel. In Figure 9, we

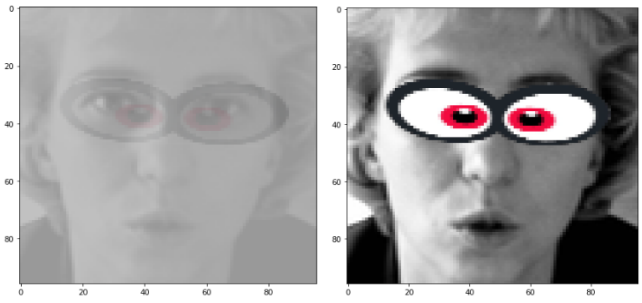


Figure 9: (Left) Image Before Alpha Composition (Right) Image After Alpha Composition

can see through the left image that the contrast in the image was greatly decreased due to the white background in the filter image. Applying alpha composition results in the right image, which shows the true colors of the combined image.

Results

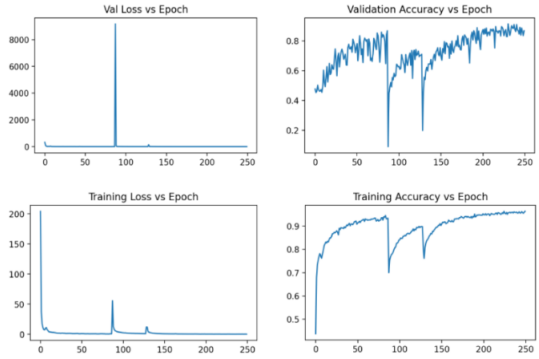


Figure 10: These four graphs show the performance of our model over 250 epochs. (Top) The validation loss and accuracy (Bottom)The training loss and accuracy

As you can see from the graphs, the overall loss sharply decreases over the course of training. Due to the the identity shortcut block in the ResNet architecture, the loss sharply increases at points where the identity block directly connects to the ouput. At those points the loss increases by a lot and the accuracy decreases by a lot as well. However, the model continues to learn over time because of this added noise and we were able to reach 0.95 training accuracy and 0.90 validation accuracy after 250 epochs. This proved our model's ability to predict the set of features on a given face. We decided to run for 250 epochs as running it for any longer yielded in worse predictions even though the training and validation accuracy continued to increase.

Our result is similar to a Snapchat filter where we locate and predict landmarks to add an overlay image onto the face. Figure 8 shows the images throughout the pipeline of the project. We can see how in the first row the original, colored images that were taken of various people were modified and properly landmarked by our Deep Learning model in the second row. In each of the images in the second row, we can see how the model was able to correctly look for landmarks such as the forehead, mouth, and eyes. We can then see how these images and their matching landmarks in the second row were properly overlaid with filter images using Affine Transformations and Image Overlay techniques. Each of the images shown use different types of filters and highlights how these different combinations of filters worked on different types of faces. Looking at the results from the Deep Learning model's landmarking accuracy and the resulting pictures, we were able to achieve our goal and produce good results.

Figure 11: (Top) Original Face Image samples taken by each group member (Middle) Grayscale images with landmarks mapped to each face (Bottom) Showcase of the different filters being overlaid on each of the faces

Difficulty and Challenges

During the data augmentation the images were transformed using PyTorch transforms. However, the landmarks that went with the images had to be transformed manually. When we transformed the landmarks using the equations mentioned above, there were negative value coordinates. This caused the landmarks to no longer be aligned with the face because the PyTorch library transforms the coordinates to positive values even if the value calculated was negative. This problem was solved by writing a rotation helper function to check for all the angles and their corresponding transformations such that the landmark transformations fall into the positive coordinates and overlay on the face points accurately.

There were some challenges when it came to creating and overlaying the filters. Because the landmark points defined by the dataset do not go over facial features like the outline of the head, the cheeks, or the hairline, certain filters either had to be scrapped or have less accurate landmarks associated with them. There were also some minor difficulties overlaying images due to over compositing. Altered images had higher intensities due to the combination of the white background in the overlay images. This was able to be resolved using the Painter's Algorithm. Images needed to be in the same alpha channel to be altered which required a conversion from uint8 to RGBA.

Strengths and Weaknesses

The strength of our model is that it is able to accurately identify landmarks on faces that are up close and properly overlay the images without alternating its intensity. We can also easily add new overlay images into the system since it does not take much time to add a new image and assign new landmark points to the image. The weaknesses of our model is that it is unable to recognize faces from a further distance. This is due to many factors such as the noticeable

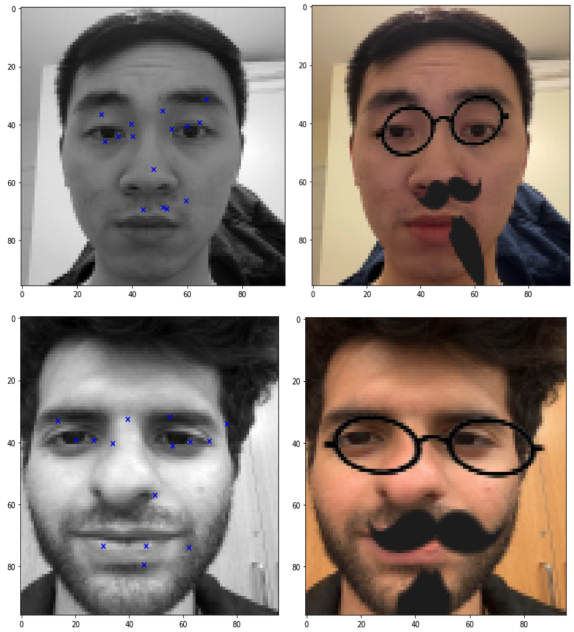


Figure 12: (Left) Landmarks at a further distance and not centered (Right) Image Overlay at a further distance and not centered

difference between the size of our training dataset and model's trainable parameters and the quality of our training data where the images only included very up close images of faces. These flaws also led to the model not being as accurate on images with heavy shadows,certain types of hair, and background objects. The model is also only able to define landmarks on 96x96 images, due to the fact that we only trained our model with 96x96 images. Lastly there is a small amount of landmark features which causes more approximation and slightly less accurate results. For future refinements due to time constraints we would modify the model and training data set to allow the model to recognize faces from further away. Currently the model is only able to recognize facial features that are up close. We would have to do further research on different network models to allow closer images. Using higher resolution filters/face images or facial meshes would also allow better facial feature recognition and create more facial landmarks. Lastly, we could apply our filters to video rather than to just images which would require a more complex CNN model which could detect the movement of the face and apply the filter accordingly.

Future Refinements

For future refinements due to time constraints we would modify the model and training dataset to allow the model to recognize faces from further away. Currently the model is only able to recognize facial features that are up close. We would have to do further research on different network

```
Data columns (total 31 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   left_eye_center_x          7039 non-null   float64
 1   left_eye_center_y          7039 non-null   float64
 2   right_eye_center_x         7036 non-null   float64
 3   right_eye_center_y         7036 non-null   float64
 4   left_eye_inner_corner_x    2271 non-null   float64
 5   left_eye_inner_corner_y    2271 non-null   float64
 6   left_eye_outer_corner_x    2267 non-null   float64
 7   left_eye_outer_corner_y    2267 non-null   float64
 8   right_eye_inner_corner_x   2268 non-null   float64
 9   right_eye_inner_corner_y   2268 non-null   float64
 10  right_eye_outer_corner_x   2268 non-null   float64
 11  right_eye_outer_corner_y   2268 non-null   float64
 12  left_eyebrow_inner_end_x   2270 non-null   float64
 13  left_eyebrow_inner_end_y   2270 non-null   float64
 14  left_eyebrow_outer_end_x   2225 non-null   float64
 15  left_eyebrow_outer_end_y   2225 non-null   float64
 16  right_eyebrow_inner_end_x  2270 non-null   float64
 17  right_eyebrow_inner_end_y  2270 non-null   float64
 18  right_eyebrow_outer_end_x  2236 non-null   float64
 19  right_eyebrow_outer_end_y  2236 non-null   float64
 20  nose_tip_x                 7049 non-null   float64
 21  nose_tip_y                 7049 non-null   float64
 22  mouth_left_corner_x        2269 non-null   float64
 23  mouth_left_corner_y        2269 non-null   float64
 24  mouth_right_corner_x       2270 non-null   float64
 25  mouth_right_corner_y       2270 non-null   float64
 26  mouth_center_top_lip_x     2275 non-null   float64
 27  mouth_center_top_lip_y     2275 non-null   float64
 28  mouth_center_bottom_lip_x  7016 non-null   float64
 29  mouth_center_bottom_lip_y  7016 non-null   float64
 30  Image                      7049 non-null   object
```

Figure 11: A table containing the data column labels of the landmarks and other information such as the image array used within the dataset

models to allow closer images. Using higher resolution filters/face images or facial meshes would also allow better facial feature recognition and create more facial landmarks.

Lastly, we could apply our filters to video rather than to just images which would require a more complex CNN model which could detect the movement of the face and apply the filter accordingly.

Appendix

Data

The dataset used for training contained 8832 total human face images, and each image has 15 landmarks corresponding to some of the prominent features that could be seen on a face such as the eyes, ears, nose, lips and mouth. This dataset also has a testing.csv file which includes a list of 7049 images, each row contains coordinates for 15 facial landmarks and image data as row ordered pixels. Also includes a list of 1783 test images which contains image id and data as row ordered pixels. There is also a sample dataset of a list of 27,124 facial landmarks to predict. The following image shows the landmark coordinates and labels in the training data.

Libraries Used

- **OpenCV**: Used mainly to read in images from our system or Google Drive and apply the Affine transformations onto the filter images. Also used to preserve the alpha values of our images for proper overlaying of filter images onto face images
- **Matplotlib**: Used to plot and show the various images used within our code
- **NumPy**: Used mainly to manage or do arithmetic operations on arrays
- **Pandas**: Used for reading and displaying in the facial landmarking dataset
- **Tdqm**: Displayed time and epochs when training Deep Learning model and opening training dataset
- **PyTorch**: Used for creating Affine transforms on images in data augmentation
- **TensorFlow**: Used for creating Affine transforms on images in data augmentation
- **Random**: Used to randomize brightness, sharpness, rotation angle and translation in data augmentation
- **Pillow (PIL)**: Used for converting numpy arrays to images and resizing images
- **ZipFile**: Used to extract data from zip files

References

[Cos12] Arthur Coste. "Affine Transformation, Landmarks Registration, Non Linear Warping". In: (2012). DOI: http://www.sci.utah.edu/~acoste/uou/Image/project3/ArthurCOSTE_Project3.pdf.

[Dan] Joseph Rabinoff Dan Margalit. "Interactive Linear Algebra". In: (). DOI: https://textbooks.math.gatech.edu/ila/least-squares.html.

[Kit17] Kris Kitani. "2D Alignment: Linear Least Squares". In: (2017). DOI: http://www.cs.cmu.edu/~16385/s17/Slides/10.1_2D_Alignment__LLS.pdf.

[Smi] Alvy Ray Smith. "Alpha and the history of digital compositing". In: (). DOI: http://www.alvyray.com/Memos/7_alpha.pdf.

[Traa] Affine Transformations. URL: https://docs.opencv.org/3.4/d4/d61/tutorial_warp_affine.html.

[Trab] PyTorch Transforms. URL: https://pytorch.org/vision/stable/%20auto_examples/plot_transforms.html#%20sphx-glr-auto-examples-plot-transforms-py.

Contributions

Affine Transformations & Image Overlay
Erica Chou, Henry Tse
Deep Learning & Facial Landmarking
Satyen Singh, Abhin Gude