HIGHLY VERSATILE FACIAL LANDMARK DETECTION MODELS USING ENSEMBLE OF REGRESSION TREES WITH APPLICATIONS

Peshotan Irani
301401116
pia5@sfu.ca

Chidambaram Allada
301409991
callada@sfu.ca

Manju Malateshappa
301377437
manjum@sfu.ca

Mrinal Gosain
301393980
mgosain@sfu.ca

ABSTRACT

Our goal through this project was to train and create a highly versatile model that detects facial landmarks using the methodology outlined in the paper from Kazemi[1]. Our model was then compared to the industry-standard model used in dlib to detect the facial landmarks. Our technique uses pixel intensities differences to directly estimate the landmark positions, these positions (which are estimates) are then refined using a process called Cascade of Regressors (this is an iterative process). Every iteration of the process results in new landmark estimates which have a lower error on alignment than the previous one. We hyper-tuned five parameters to get a balance between computational speed and accuracy. Our final model had a test error (sum of squares) of 8.57 which was very close to the error of the industry-standard model used by dlib which has an error of 8.27. What we discovered through our project is that we can train highly versatile models that are significantly less complex than the dlib models (our model was 27.5mb vs 65.9mb for the dlib's model) and can be used in devices which have a lower processing power. Finally, for the application part of our project, we used our model to develop a facial swap software that uses the landmarks and then applies transformation and color correction to blend the two images. We later expanded our model to include live webcam swaps and finally to any videos. Further suggestions for the model include training our model to detect only specific body parts such as eyes, nose,etc. and then swapping only those parts. For the software improve we can develop a function that counters the mouth area in a better way and helps in having a better lip sync.

1 Introduction

Through our project we have attempted to present, using an ensemble of regression trees, an algorithm that can do face alignment quickly and at the same time achieve accuracy in comparison to the previously done state –of –the –art methods on the standard data set. We have tried to solve the face alignment problem using a cascade of regression functions. Each of the regression function in the cascade efficiently estimates the shape from an initial estimate as well as the intensities of the pixels that are indexed relative to this initial estimate. There are two key elements that we incorporate into our learnt regression functions.

The first key element is the very inception of this project which starts with the indexing of the pixel intensities relative to the current estimate of the shape. Depending on the illumination conditions, the extracted features which are the vector representation of the face image can vary greatly. This is the nuisance factors. The extracted features can also vary due to the shape deformation. The above-mentioned condition makes it difficult for accurate shape estimation using the extracted features. The dilemma is that we need reliable features to accurately predict the shape and the second one is that we need an accurate estimate of the shape to extract reliable features. To tackle this problem an iterative approach is followed where the image is transformed to a normalized coordinate system based on a current estimate of the shape and then the features are extracted to determine an update vector for the shape parameters. The second key element is the one where we consider how to combat the difficulty of the inference/prediction problem. During testing time, an alignment algorithm estimates the shape, which is a high dimensional vector that agrees with the image data and model of our shape.

2 Prior Work

Prior work includes the standard model used by dlib which is also based upon Kazemi's paper[1]. Dlib's model has 68 facial features which have an architecture similar to that shown in figure 1. Their model has a sum of squared error of 8.27, which is impressive but is very complex and is much larger in size (about 65.9mb) than our trained model which is almost half that size (27.5mb) and has a sum of square error of 8.57. This poses a difficulty of running the Dlib model on devices of low computational power.

Features	Points
Jaw	0 - 17
Left Eyebrow	17 - 21
Left Eye	36 - 41
Right Eyebrow	22 - 26
Right Eye	42 - 47
Nose	27 - 35
Mouth	48 - 60

Figure 1: Image and the Corresponding table showing the schematic of the final model

3 Approach

Taking a cue from the research paper [1] that we have used as a reference frame for our project, the algorithm to precisely estimate the position of facial landmarks has been given as follows. The starting point is the Face landmark utilization which is the process of extraction of set of key features from the given image. There are at present many algorithms for this purpose. For this project, we utilized a cascade of regressors. The methodology and training part has been explained in detail in further sections.

3.1 Cascaded Regression Method

Cascaded regression methods for face alignment are regression-based methods that work in a stage-by-stage cascade, iteratively improving an initial shape estimate in a coarse-to-fine manner. Beginning with the face landmark localization method, which is the process of extrapolating the set of key points from a given face image. Let x_i denote the x,y coordinates of the i^{th} facial landmark in an image I, where $x_i \in \mathbb{R}^2$. Then the shape vector S will be the coordinates of all the p facial landmarks in I. It can be denoted as $S = (x_1^T, x_2^T, \ldots, x_p^T)^T \in \mathbb{R}^{2p}$

Let $\hat{S}^{(t)}$ denote the current state estimate of vector S. Let r_t denote the current state regressor in the cascade which predicts an update vector from the image I and $\hat{S}^{(t)}$, that is added to the current shape estimate $\hat{S}^{(t)}$ to improve the estimate.

$$\hat{S}^{(t+1)} = \hat{S}^{(t)} + r_t(I, \hat{S}^{(t)}) \tag{1}$$

The regressor r_t makes it's prediction based on features, such as pixel intensity values, computed from I and indexed relative to the current shape estimate $\hat{S}^{(t)}$. As mentioned above, since the predictions are based on the features, some form of geometric invariance is maintained in the process and as the stages of regression(cascade) proceed we can be more certain that a precise semantic location on the face is being indexed.

The range of outputs expanded by the ensemble lie in a linear subspace of training data if the initial estimate $\hat{S}^{(0)}$belongs to this space. This initial shape $\hat{S}^{(0)}$can be chosen as the mean shape of the training data , centered and scaled according to the face rectangle(bounding box output) of a generic face detector. To train each r_t we use the gradient tree boosting algorithm with a sum of squared error loss. The regressors produce a new estimate from the previous one, trying to reduce the alignment error of the estimated points at each iteration.

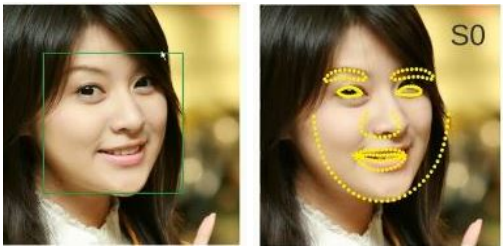

Figure 2: Mapping the mean shape to face rectangle[1]

3.2 Regressor Learning Method in the Cascade

Assuming that we have our training data as $(I_1, S_1), \ldots, (I_n, S_n)$ where I_i is the face image and S_i is its shape vector. To learn the first regression function r_0 in the cascade, we create triplets of face image, initial shape estimator and a target update step from our training data. i.e $(I_{\pi_i}, \hat{S}_i^{(0)}, \Delta S_i^{(0)})$ where

$$\pi_i \in \{1, \ldots, n\} \tag{2}$$

$$\hat{S}_i^{(0)} \in \{S_1, S_2, \ldots, S_n\}/S_{\pi_i} \tag{3}$$

$$\Delta S_i^{(0)} = S_{\pi_i} - \hat{S}_i^{(0)} \tag{4}$$

for i = 1,. . . ,N. The total number of triplets that we need to create will be N=nR where R is the number of initializations used per image I_i. Each initial shape estimate for the image is sampled uniformly from $\{S_1, \ldots, S_n\}$ without replacement. The set of training triplets is then updated to provide the training data,$(I_{\pi_i}, \hat{S}_i^{(1)}, \Delta S_i^{(1)})$ for the next regressor r_1 in the cascade by

$$\hat{S}_i^{(t+1)} = \hat{S}_i^{(t)} + r_t(I_{\pi_i}, \hat{S}_i^{(t)}) \tag{5}$$

$$\Delta S_i^{(t+1)} = S_{\pi_i} - \hat{S}_i^{(t+1)} \tag{6}$$

The process is iterated till a cascade of T regressors $r_0, r_1, \ldots, r_{T-1}$ are learnt which when combined give a sufficient level of accuracy.

Each regressor is here learnt using the gradient boosting tree algorithm, so the square error loss associated is used and the residuals computed in the innermost loop corresponds to the gradient of the loss function evaluated at each training sample. We also have a learning rate parameter ν where 0<ν<1, also known as the shrinkage factor. This learning rate parameter helps in combatting over-fitting.

3.3 Selecting Node Splits

To train the regression tree, we randomly generate a set of splits which is θ's, at each node. We choose the θ* that minimizes the sum of square error. If Q is the set of the indices of the training examples at a node and θ is the corresponding split, the error is given by Once

$$E(Q, \theta) = \sum_{s \in \{l,r\}} \sum_{i \in Q_{\theta,s}} ||r_i - \mu_{\theta,s}||^2 \tag{7}$$

where $Q_{\theta,l}$ is the indices of the examples that are sent to the left node depending on the θ. r_i is vector of all the residuals computed for the image i in the gradient boosting algorithm.

$$\mu_{\theta,s} = \frac{1}{|Q_{\theta,s}|} \sum_{i \in Q_{\theta,s}} r_i \quad \text{for} \quad s \in \{l, r\} \tag{8}$$

The optimal split that minimizes the loss can be found efficiently by

$$\arg\min_{\theta} E(Q, \theta) = \arg\max_{\theta} \sum_{s \in \{l,r\}} \mid Q_{\theta,s} \mid \mu_{\theta,s}^T \mu_{\theta,s} \tag{9}$$

We only need to compute the$\mu_{\theta,l}$when evaluating different values of θ, as $\mu_{\theta,r}$ can be calculated from the average of the targets at the parent node by

$$\mu_{\theta,r} = \frac{|Q|\mu - |Q_{\theta,l}|\mu_{\theta,l}}{Q_{\theta,r}} \tag{10}$$

4 Experiments - Feature Selection and Parameter Tuning

As discussed earlier the decision at each node is based on threshold the difference of intensity values at a pair of pixels. However there is some caveat associated to it. The drawback of using pixel intensities differences is that number of potential split , i.e θ , is quadratic in the number of pixels in the mean image. Thus determining the θ's that minimize the sum of square error becomes difficult.

However this limiting factor can be eased by taking the structure of image data into account with the introduction of exponential prior P over the distance between the pixels used in a split so that the closer pixels are chosen.

$$P(\mathbf{u}, \mathbf{v}) \propto e^{-\lambda \|\mathbf{u}-\mathbf{v}\|} \tag{11}$$

This introduction of prior also reduces the prediction error on the number of face dataset as analyzed in the research paper. Our model was trained in the process similar to dlib shape predictor model (landmark_68). The dataset we used was the iBug_300 dataset. This dataset was about 1.8Gb in size and contained around 2500 images. During our training process we had the option to finetune about 5 hyper-parameters to achieve the maximum accuracy for the shape predictor in the dlib library. These were as follows:

4.1 Regularization Constant (Nu)

The regularization constant basically dictates how generalized the model will be. A value close to 1 would not be generalized at all and would cause in the result of fixed-overfitted model. Whereas a value of close to zero will require a lot more training samples to train the model. This increases the computational load and thus requires more resources to train. After experimenting will different values (0.3, 0.2, 0.15, 0.125 ,0.1), we decided to go with the value of 0.125.

4.2 Depth of the tree

This basically determines how big our model would be. We experimented with values of 5,4,3 and 2. When we used a depth of 5 it was taking way longer to execute and hence we had to abort our program midway. According to us the best fit between accuracy and speed was a value of 3.

4.3 Size of the Feature Pool

This specifies the number of pixels used to generate the features for the random trees at each cascade. A value of around 400 is ideal trade-off between accuracy and execution speed. A higher value leads the algorithm to be more accurate but slows the down the execution. We chose a value of 300 because, because it is noticeably faster and leads to fairly accurate modeling.

4.4 Oversampling Amount

This parameter is needed when the dataset is small, what it does is it applies a random deformation to the training samples, thus generating more training samples and hence increasing the size of the training set. In our case the dataset was large enough (about 2500 images) and over 1.8 Gb of data, hence we didn't need any oversampling.

4.5 Cascade

As the name suggest this parameter decides the depth of the cascades. Again this affects the accuracy of the model and comes at a cost of execution time and resources. We tried values in range of 10-18 and found out that a value of around 13 was the best fit. A value over 16 was very computationally expensive. Value of 10 worked as well but the accuracy was comprised.

5 Observations

We used a variety of combinations of the parameters for training our data. We started with the following initial combination : Depth of Tree = 3, Size of Feature Pool = 300, Oversampling Amount = 5, Cascades = 13.

The first parameter we varied was the Regularization Constant (Nu) we varied it from 0.4 to 0.1 and got the following sum of square error:

Nu	Training Error	Testing Error
0.4	15.24	27.57
0.3	16.87	20.91
0.2	17.38	16.45
0.15	18.78	12.23
0.125	22.89	15.89
0.1	24.78	15.45

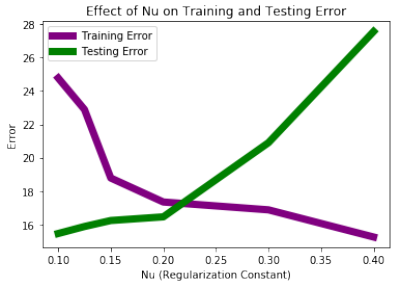

Figure 3: Effect of Nu on Train and Test Error

Next we trained the model varying number of cascades, we varied our model from cascades from 10 to 15. The result was as we expected when we increased the number of cascades both training and testing error went down, but it began computationally expensive.

# Cascades	Training Error	Testing Error
10	21.24	26.24
11	19.87	24.87
12	17.34	22.34
13	14.98	20.23
14	14.32	19.25
15	13.91	18.91

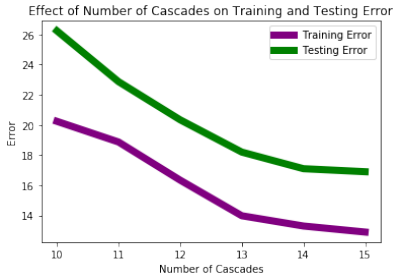

Figure 4: Effect of Number of Cascades on Train and Test Error

The last parameter we varied was the depth of the tree. As expected the higher the depth of the tree the smaller was the error, but again increasing the depth of the tree was computationally expensive.

Tree-Depth	Training Error	Testing Error
2	17.87	20.87
3	13.34	16.23
4	11.41	14.98
5	10.32	13.25

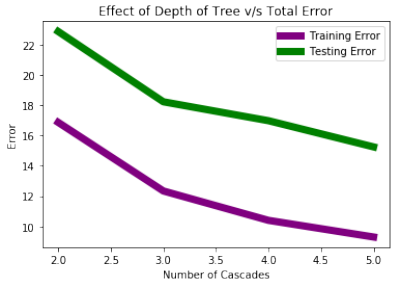

Figure 5: Effect of Tree Depth on Train and Test Error

The final combination we selected was Nu = 0.125, Depth of the tree = 3, cascade = 13, feature pool size = 300, num test split = 125, oversampling = 1.

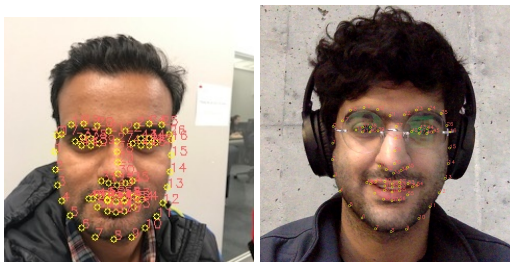

Figure 6: Left image has landmarks from our model and the image to the right has landmarks from dlib

6 Application - Software (Static and Video Swaps)

Once our model was trained we used it to develop a facial swap software. For the development of this software we referred to Matt Ed's blog on face swap. The first part of the software is to use the trained model to locate landmarks on both faces. We now have the landmark matrices of the 2 landmarks, the points in these matrices reference to the same feature, for example, 47^{th} row will give the coordinates of the left eye. We then apply transformations (rotation, translation and scaling) to our matrices using Procrustes analysis, this will help us overlaying the images on top of each other [2]. Mathematically this can be represented by the following equation

$$\sum_{i=0}^{67} \parallel sRp_i^T + T - q_i^T \parallel^2 \tag{12}$$

We seek to find the R, T, s in this equation by minimizing the above equation, where R is an orthogonal 2x2 matrix, s is a scalar, T is a 2-vector, and p_i and q_i are the rows of the landmark matrices calculated above.

The next step involves correcting skin tone colors between the two images. For this, we make use of a function that attempts to match the two colors by dividing the image1 by the gaussian blue of image2 and then multiply it by gaussian blue of image1. This also helps in the lighting conditions that might differ in the two images. The key here to have an appropriate size for the gaussian kernel. If the blue is too then along with the skin tone the features from one image will transfer to the other, if we use a number which too large then there will discoloration and the blending will not be smooth [2].

Finally, the swap is made using a face mask. We define a function to generate a mask for the landmark matrix. It then applied a polygon around the nose, eyes and mouth. This creates a bounded region for the features, this is created for both the images. Finally, a last function is used to swap the mask of image1 onto image2 and vice-a-versa, this is done by taking the element-wise maximum and combining the two masks such that features from image1 are covered by features from image2 and vice-a-versa. The following images show the facial swaps on two images.

Figure 7: Static swap of two images

After the static swap, we decided to apply this to live video, because live videos are also made up of individual frames which can be treated as individual images and we can then apply the same methodology to each frame. The result of swapping the face of a YouTuber with Kim Kardashian's face is below.

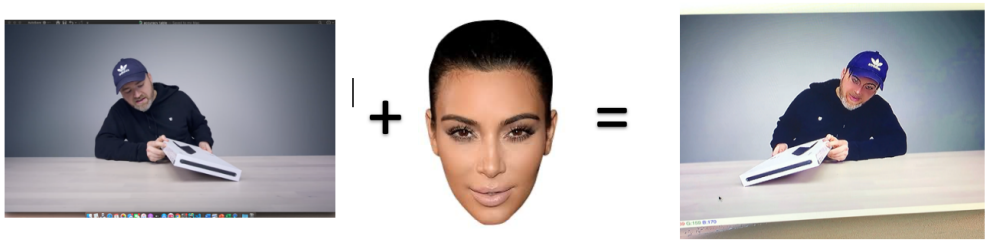

Figure 8: Superimposing an image on a live video

6.1 Conclusion, Limitations and Further expansion

Through training our model and developing the application for it, we have concluded that Kazemi's algorithm can be used to train models that can vary vastly in size but can be reasonably accurate. The benefit of this that we can run our model in environments of lower computational power such as mobile phones, etc. in which Dlib's model fails. For further expansion we that this swapping of features can be done for any feature (nose, eyes, hair, etc.). To apply this we have to train our facial landmark model for only those sets of features, e.g. we can train our model to only locate the landmarks associated with the eyes. Then we can use functions similar to the full facial swaps and make the necessary swaps. The image below shows landmark recognition for only eyes.

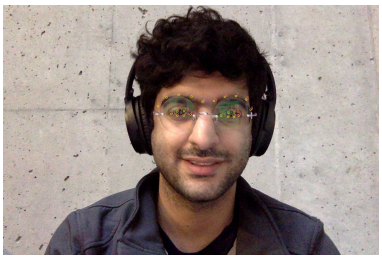

Figure 9: Training our model to detect only specific parts of the face

The limitation we found with our model was that when we applied it to the application, the mouth sync was not working as intended this can be addressed by writing a function that detects the spacing between the landmark position of the mouth and then syncing it which the image.

7 Contribution - Team

Each member of the team contributed equally **(25% each)** to the success of the project. We all worked on almost all the sections together so it was really hard to pinpoint what member did what.
Peshotan Irani - Hyper-parameter training, Literature reading, Report Compilation, Training Model
Chidambaram Allada - Literature Reading, Training Model, Expansion of Application, Report Compilation
Manju Malateshappa - Literature Reading, Report Compilation, Hyper-parameter Tuning, Training Model
Mrinal Gosain - Literature Reading, Expansion of Application, Hyper-parameter Training, Report Compilation

References

[1] Kazemi, Vahid, and Josephine Sullivan One millisecond face alignment with an ensemble of regression trees. *the IEEE conference on computer vision and pattern recognition. 2014.*

[2] Mathew Earl. Swapping Facial Features. *https://matthewearl.github.io/.*