# Age Estimation by Using Face Images

Hao-Hsin Shih (hs2762), Ming-Ying Chung (mc3808)

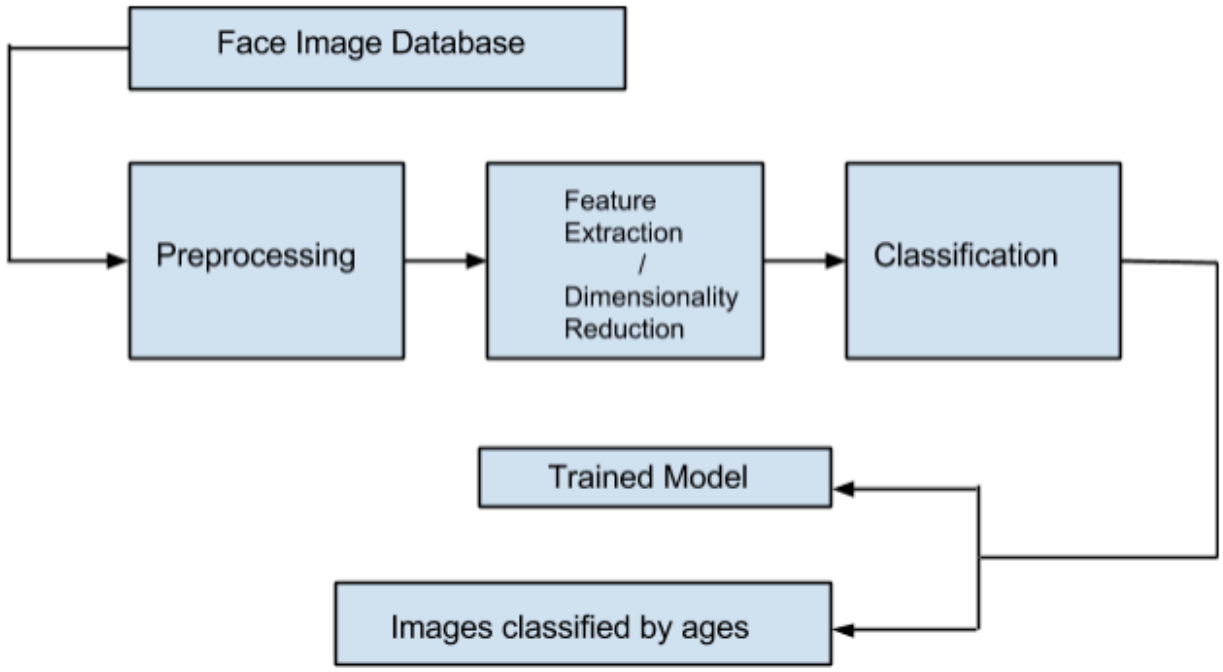
## 1. Introduction

Faces are one of the most important traits of human beings. We identify people almost by face, which includes tons of information like genders, races, expressions, and ages. Researchers pay lots of attention in this topic recently due to its significance and rising role in biometrics, commercial online systems, and social network area. However, it is no doubt that it is hard to reach an acceptable accuracy of predicting an age from face image so far. Age estimation by using face images is a complex problem due to several reasons. First, different genes, genders and even growing background leads to totally different appearances even in the same age. It is not unusual one looks like 5 to 10 years older than his/her actual age should look like. Besides, age estimation is also difficult for human. We use empirical experience to identify age groups, but we cannot precisely tell ages.

Age estimation can be divided into the process of modeling faces and age classification based on the models. One of the earliest researches on age estimation [8] takes advantage of empirical analysis on facial features ratios and wrinkle geography on faces to model input faces. Later, several methods proposed to model face representation, which can be categorized into generative and non-generative methods [12]. Each representation may have its limitations, and we are going to focus on techniques for extracting local features on faces.
Local binary patterns (LBP) and Gabor filter have been applied to this field combined with other classification methods. In Shan's research [4], they applied both basic LBP/Gabor and boosted LBP/Gabor with Adaboost and SVM to do age estimation, and achieve 55.9% accuracy in a dataset over 7 age groups from 0-66+. Another research [5] shows that modified LBP can achieve higher accuracy over the same age groups.

In this project, we develop an age estimation system using uniform LBP to model faces and support vector machines (SVM) as classifier on sampled MORPH database [1] and the Images of groups dataset [10]. We also develop an elegant Android client as a front end to query our age estimation system. We focus on people at 15 to 60+ years old and divided into six age groups: 15-19, 20-29, 30-39, 40-49, 50-59 and 60+.

## 2. Algorithm Pipeline




### 2.1 Face Image Database

To experiment on our age estimation system, we need a set of face images labeled with accurate age and evenly distributed over defined age groups. Also, there are several constraints on our dataset: (1) each image contains only one clear front face without any obstruction (2) we only focus on adult faces (defined later).

After careful research, we choose to use images from MORPH Database [1] and the Images of groups dataset [10] in our project. The dataset features a longitudinal face images collected in real-world conditions. There are tens of thousands of images in MORPH, where we sample 1200 face images, and every 400 of them in age group of 30-39, 40-49 and 50-59 respectively. In the groups dataset [10], each image contains multiple faces. So we have to extract faces by multiple phases of face detection. By locating multiple faces in a image (described later), we can splits them into different smaller image files. In addition, images may also include people out of the age regions we target. Therefore, cross-match is also needed, which ensures the data set follow our assumption of input. We collect the dataset for age group 15-19 and 20-29 from groups

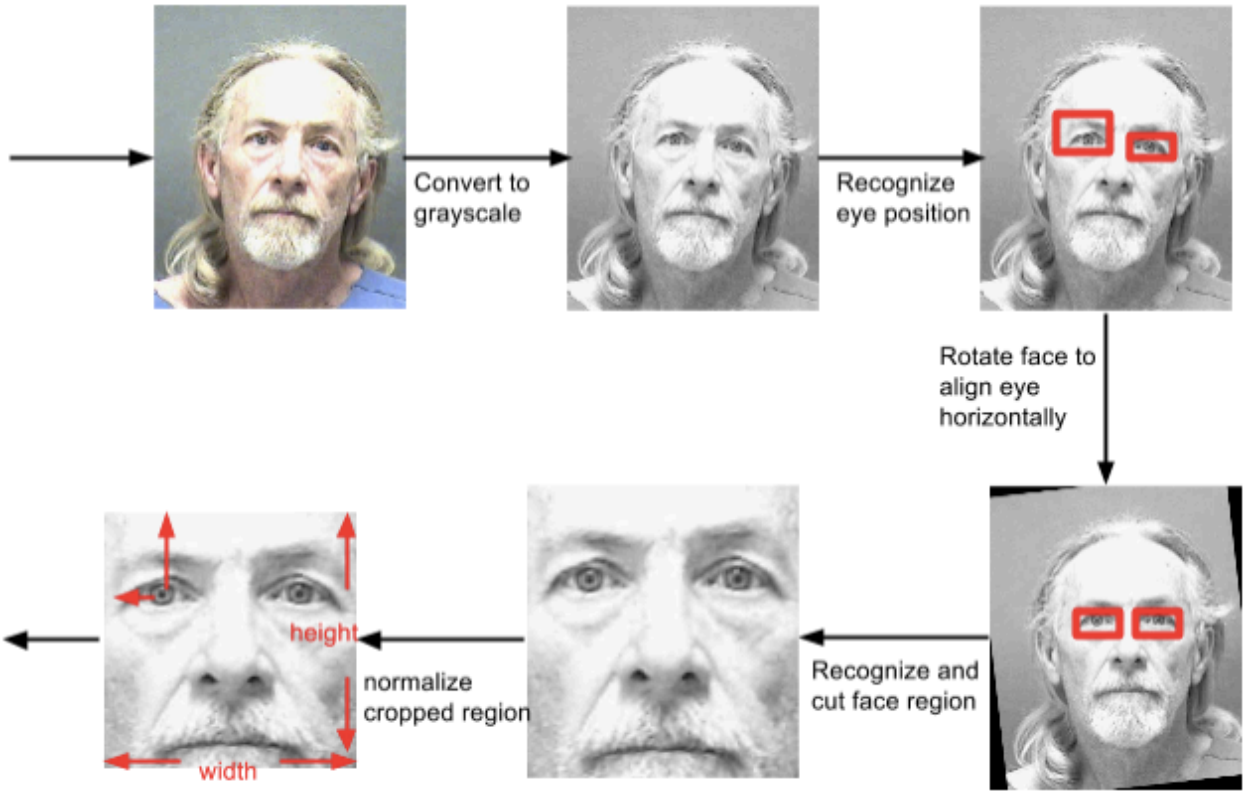
data set by the way mentioned above. The face images in age group 60+ come from both MORPH dataset and groups dataset.

There are several reasons for the choice of age grouping. Previous research [4][5] shows that the accuracy of estimating the age of people under 2 years old and above 66 years old are relatively higher than other age groups. We try to raise the accuracy of age estimation in adult age groups, which have relatively lower accuracy.

After we reached the accuracy 67.62% in cross-validation for age group 30-39, 40-49 and 50-59 in prototype with 1200 images, we decided to include more age group in our classifier, and train it with 2240 images, which are almost the double of the amount of images in our prototype version.

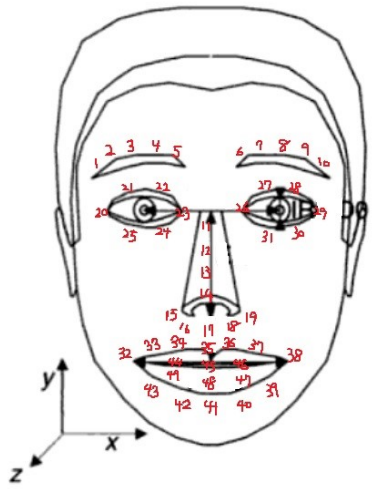
Finally, we reach an accuracy of 59.0889% for 6 categories classification defined above and 10-fold cross-validation on 2240 image in different size and format (grayscale or RGB). Since the purpose of this project is to build an app for mobile user to take a photo and upload the picture to server for analyzing the age by face and return the result to user, the diversity of possible kinds of pictures is considered. Thus, we allow images in different sizes and format in our dataset for training a model we used to predict the age by SVM.

## 2.2 Preprocessing




Before we start to do age estimation using human faces, the first problem we have to conquer is how to extract the human face from a image as precisely as possible. Our goal is to extract the face with front head in gray-scale.
At the beginning, we think of OpenCV, which is a famous open source tool for many real world applications with efficiency. Although it is a powerful tool for image processing and object detection, it still difficult to satisfy our need in mapping the face precisely and extract in from target image.

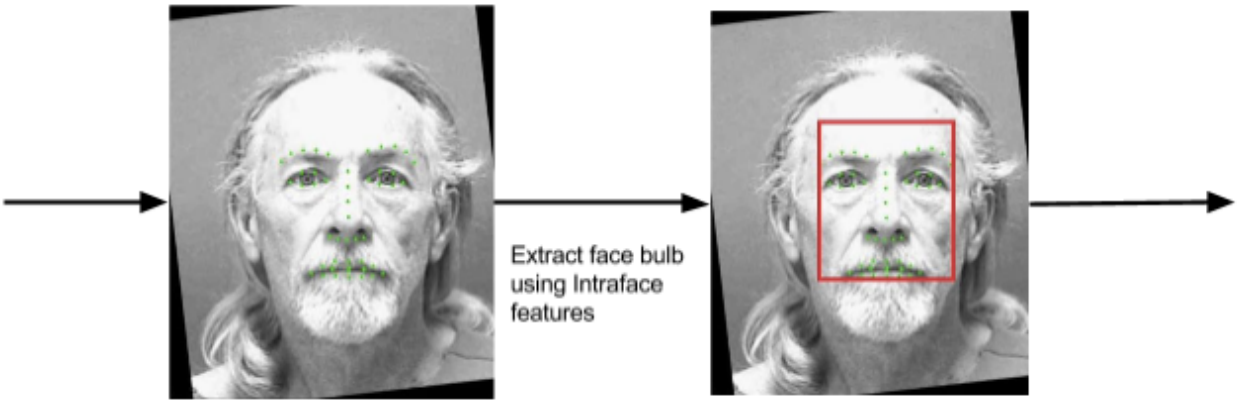
The problem is we hope to get the region of face with:



```
% original point (x, y) = (0, 0) is on upper left
left bound = min( position[1]['x'], position[20]['x'])
right bound = max( position[10]['x'], position[29]['x'])
bottom = max(position[39]['y'],...,position[43]['y'])

% Top is by estimation:
Top = min(left[y], right[y])
left = position[3] ['y'] - (position[25]['y' ] -position[3]['y'])
right = position[8]['y'] - (position[8]['y'] - position[30]['y'])
```

This is where Intraface [7] comes to our system. It helps to map the face as the way shown above. After rotating face to align eyes horizontally, we extract face by the bounds mentioned above.



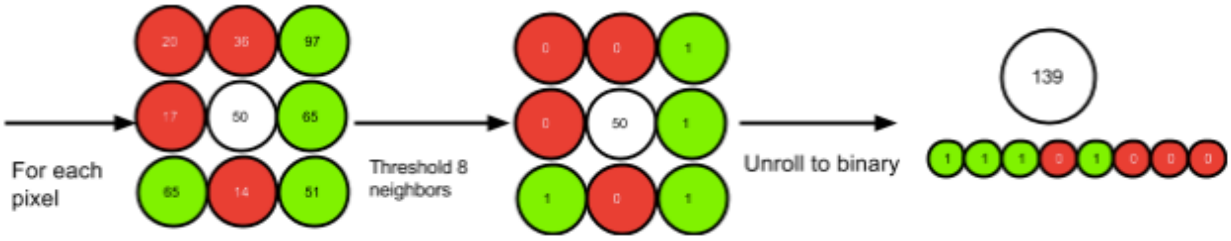

We perform this processing on all images in our database, and then continue to do feature extraction on them.

## 2.3 Feature extraction

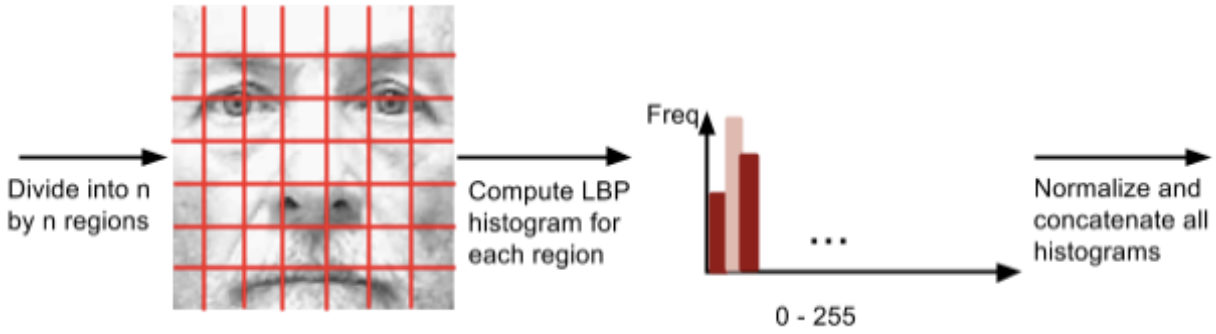
Now that we have cropped faces, we then need to figure out a way to extract features for age estimation.

### 2.3.1 Local binary patterns

Proposed in 1990s and later introduced to the field of computer vision, local binary patterns (LBP) has been widely used as a type of feature expressions in many applications, such as texture classification, object detection and facial analysis. It is a computationally efficient texture descriptor, yet robust enough to sense monotonic changes in illumination.




In our system, an LBP feature vector is created in the following step. For each pixel, we identify its 8 neighbors and threshold based on the center pixel. Then we can obtain a decimal value by unrolling threshold neighbors into an 8 digits binary.




Each input cropped face image is divided into n-by-n regions, where n can be 3, 5, or 7. In our experiment, a 7-by-7 division achieves the highest accuracy. For each region, we compute a decimal for every pixel using the above method, and then accumulate those values to obtain a LBP histogram (LBPH). The normalized LBPHs of every region are then concatenated and thus we obtain a feature vector of 7*7*256 elements for the input face.

### 2.3.2 Dimensionality reduction

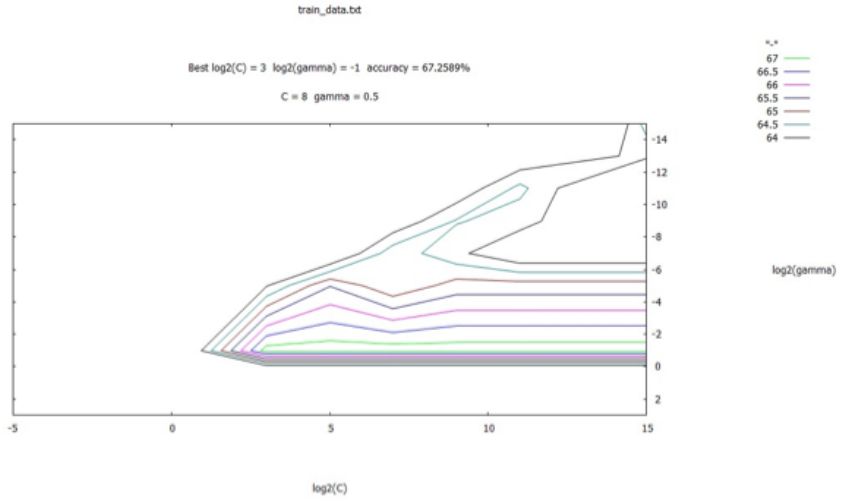
The above method generates a 256 dimensions feature in each patch, which is overwhelming for classification. Here, uniform local binary patterns are introduced. A local binary pattern is called uniform if the circular binary pattern contains at most two bitwise transitions from 0 to 1 or 1 to 0. For example, 10000001 is a uniform LBP because it has two transitions, but 1010101, which has 6 transitions, is not a uniform LBP. Past research [3] shows that uniform LBPs account for almost 90% of all patterns in texture images if LBP is calculated from 8 neighbors, which is useful enough to our system.

We map all non-uniform patterns into 1 bin, and there are 58 uniform patterns possible in a 8-bit binary. Hence, we reduce to a 59 dimensions feature vector for each patch, which is 77% smaller.

## 2.4 Classification

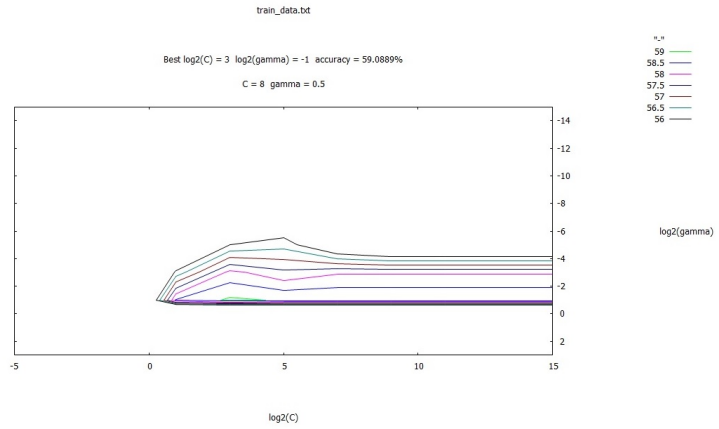
Support Vector Machine (SVM) is a popular supervised learning method to do classification and data segmentation. It is powerful to make good separation of data with multiple categories, or labels, by adjusting a hyperplane of set of hyperplanes where the dataset is in.

We use LIBSVM to train our face models, or LBPHs, and do the cross-validation to reach a acceptable precision of prediction. It is a well-known tool frequently used in machine learning, and it comes with several kernel functions, where we can adjust parameters to generate the optimal model for our data. Before training, we have to vectorize and normalize extracted features.



LIBSVM also provides some integrated function for our convenience; we could train all data with whole kernel function and parameter by brute force loop. And we could use the parameters with best precision to do the data training. The figure above shows the result of the parameters we obtain in prototype of this project by running the brute force loop by C-type kernel function with gamma = 0.5 and cost = 8, which reaches an accuracy= 67.2589% with 10 fold cross-validation on 1200 images uniformly distributed in 3 age regions 30-39, 40-49 and 50-59.

To complete this project we extend the image dataset from 1200 pictures to 2240 pictures with total 6 classes on different age groups. We use SVM to run brute force loop by C-type kernel function with gamma = 0.5 and cost = 8, which reaches an accuracy= 59.0889% with 10 fold cross-validation. The parameters are the same as the ones in prototype.  It is quite reasonable that the accuracy drop from 67.2589% to 59.0889% because the size of dataset and classes are both double. An 8.17% deduction on accuracy for such an extension on diversity of the dataset is quite acceptable. Now, the classes (age group) in our project cover all the possible age except 1-14, in which faces do not follow our definition of adults.




# 3. Experimental Results

## 3.1 Improvement by normalization and parameters adjustment

| n: n-by-n divided regions | faces w/o alignment | faces w/ eye alignment and normalization |
|---|---|---|
| 3 | 59.69 | 59.85 |
| 5 | | 63.23 |
| 7 | | 67.26 |

Our project goal is to raise the accuracy of age estimation in adult age groups. In prototype we achieve a 67.2589% precision in cross-validation by SVM with C-type kernel function with gamma = 0.5 and cost = 8. It is the best result after several trials on parameters adjustment in preprocessing.

Previous research [4][5] shows that it is relatively difficult to correctly classify adult face images, where faces with age from 37-65 only correctly classified in 41% accuracy. However, we succeed to tell the difference in only this age group. After extending the dataset to 2240 images with classes (age group), we still could achieve 59.0889% precision in the same experiment, which means the combinations of methods we use to do age estimation is quite reasonable so that it can be applied to more diverse dataset.

## 3.2 Performance measurement using confusion matrix

| | **Predicted class** | **20-29** | **30-39** | **40-49** | **50-59** | **60+** | **15-19** |
|---|---|---|---|---|---|---|---|
| **Actual class** | | | | | | | |
| **20-29** | | 252 | 0 | 0 | 9 | 3 | 167 |
| **30-39** | | 3 | 241 | 112 | 30 | 1 | 8 |
| **40-49** | | 8 | 117 | 130 | 122 | 15 | 5 |
| **50-59** | | 63 | 35 | 113 | 208 | 47 | 6 |
| **60+** | | 57 | 1 | 23 | 93 | 19 | 1 |
| **15-19** | | 180 | 2 | 0 | 1 | 9 | 168 |

The confusion matrix above is one of the results to do 10-fold cross-validation on 2240 images uniformly distributed on 6 different age groups.

### Age group 20-29

It shows a quite good result in this group. In the matrix, most images in group 20-29 are classified by correct label, but there are still over ⅓ of them classified as image in age group 15-19. Actually, it is not a big deal, because group 15-19 are neighbor of region 20-29 and they should be the most related groups. Images for this group are extracted from group dataset which contains picture with multiple people from Flickr. In real work, most make-up makes people looks younger, so it is a reasonable result.

**Age group 30-39**

Unlike the situation in group 20-29, in this case most images classified with wrong label are in the other group (40-49), but both group are still neighbors. Possible reason for this phenomenon may cause by the source of images in this group. Most pictures in the group are selected from MORPH dataset with face images from prisoners. So it may imply that prisoners usually look older than the age he or she is off make-up or lack of skin care and nutrient.

**Age group 40-49**

Images in this group are also from MORPH dataset. The result for this group is interesting because the images in it are mostly classified as its two neighbor and itself with almost equal probability. Moreover, the images classified as in age group 40-49 also distributed uniformly in its two neighbor and itself. We could say that face images in our dataset in group 40-49 have no significant difference from groups 30-39 and 50-59 for our classifier. So it has a relatively low precision in this region.

**Age group 50-59**

Something weird in this group is that large (about 14%) part of them are classified as 20-29 age, which is the furthest group from region 50-59, and it's totally wrong. For this part we could do more experiment to check what's the main issue that cause this problem, maybe is the methods we use or images we select to train the model.

**Age group 60+**

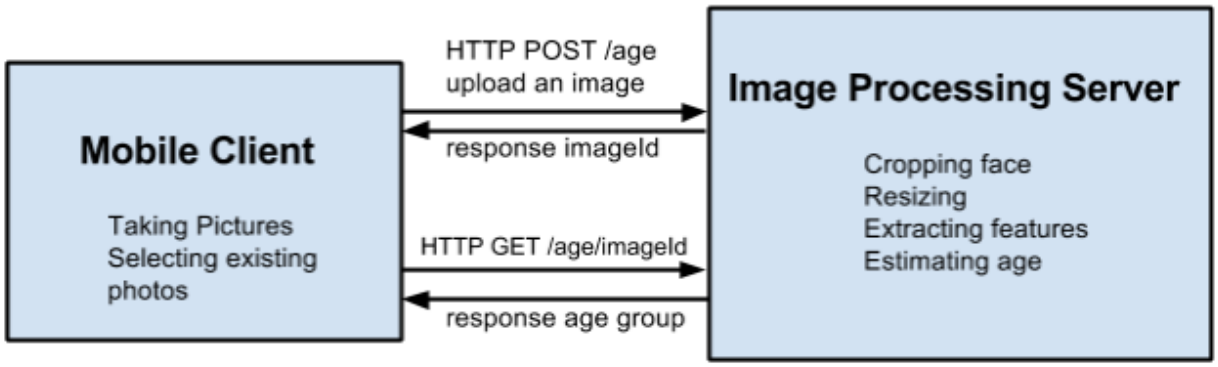
In previous research [4][5], group 60+ should have a relatively high accuracy for classification. It is said to relatively easy to classified, but it is not in the same situation in our project. Images in age group 60+ are classified as age 50-59 and age 20-29. This is also a part we fail to correct. To improve our project for classifying this age group we could just use the way implemented in previous research with higher accuracy to improve this part in the future.

**Age group 15-19**

Images in this group mostly are either classified as age 20-29 or exact in the right group. The images classified as age 15-19 are either in age group 20-29 or 15-19. So it is quite a good result because they gather intensely in the closed groups and one of them

is exact the right group. So the result for classifying images in this group actually is great if we could find some significant features to separate this two groups more precisely, then the classification in this part would reach a excellent accuracy.

# 4. System Implementation




Our system implementation is based on a series of open source stacks, from web front-end to mobile client.

## 4.1 Image Processing Server

We develop an HTTP server for age estimation based on the previous pipeline. The server is written by Node.js [11], a desktop version Javascript runtime environment. There are several advantages to develop web application in Node.js. First of all, Javascript is a dynamic typing language, which means the overhead to experiment new ideas is relatively little comparing to using Java/C++. Also, the language has an infamous built-in feature called asynchronous callback, which is suitable for handling multiple requests in real-time. Last, the prosperous ecosystem [12] of Node.js brings developers numerable solutions to every kinds of web-related problems.

We use Express.js framework [13] as a backend, which is one of the most common used MVC framework in Node.js. And our front-end is beautified by Bootstrap [14], a responsive CSS framework from Twitter.

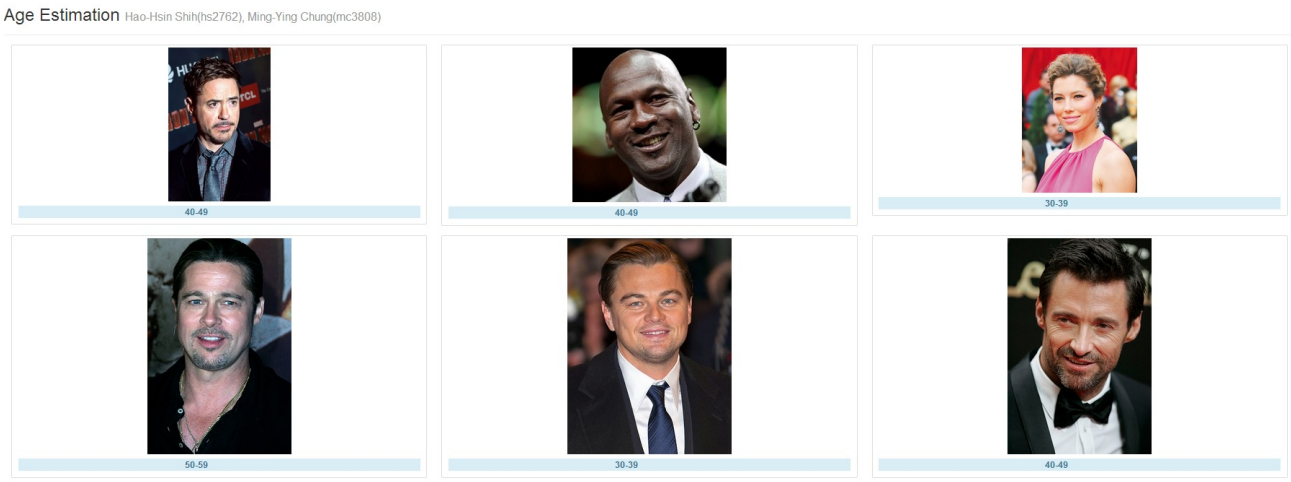
Our server keeps listening to port 3000 while being alive. Upon receiving an HTTP POST request to /age, the server will store the payload (image) of the request to a file using a unique signature by recording a timestamp. It then sends back the signature to

user. In the mean time, the server launches a child process to call our image preprocessing programs, written by Matlab, and waits for the termination. The image preprocessor extracts feature vector and then saves to a specific location. After that, the server then launches svm_predict from libsvm along with the feature vector to do age estimation.

Upon receiving an HTTP GET request to /age/imageId, our server checks if imageId is valid. If so, it responses the estimated age group to sender.

We also develop a simple frontend to easily view uploaded and estimated images.

---

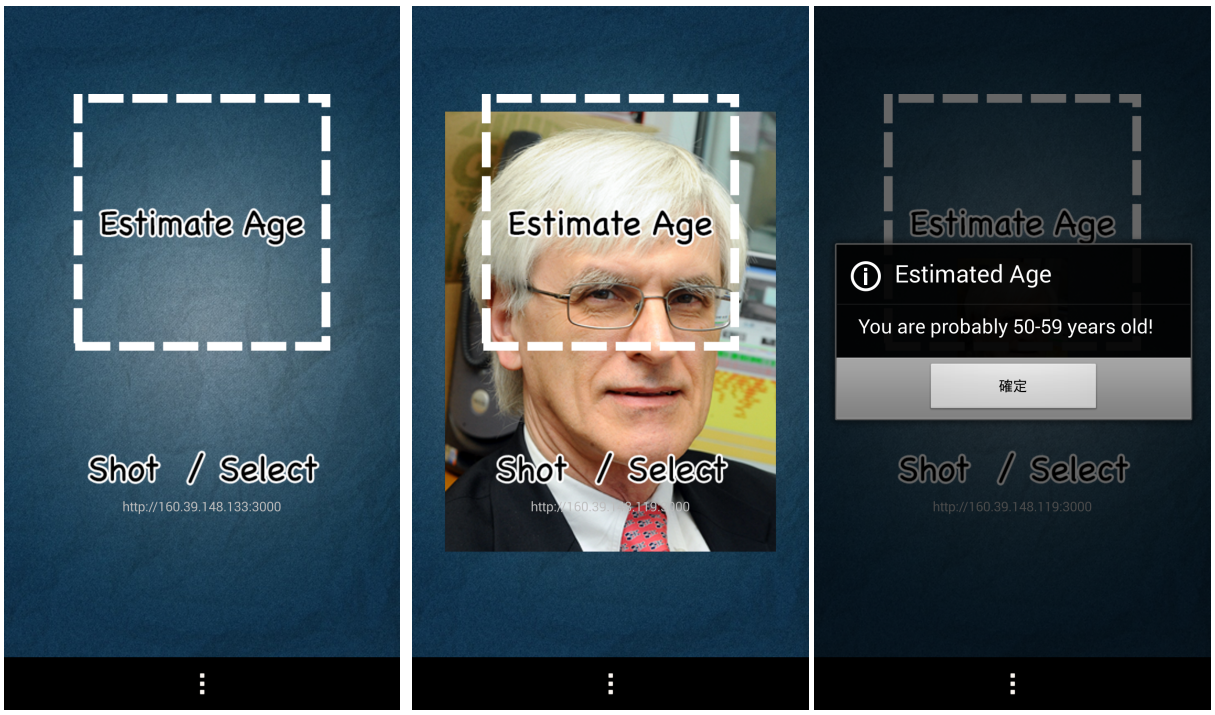



## 4.2 Mobile Client

Our choice of mobile platform is Android [15] since there are lots of well-documented examples and tutorial on the Internet. We also have access to 2 HTC devices to test on.

The application enables users to select or shot one picture at one time. It then communicates with remote server to estimate the age of a face in that image.

The user interface is simple yet clear. We have two buttons to shot and to take pictures and one region to hint the most suitable location of head for estimation. We also allow users to dynamically set up another server address.




## 5. Future Work

We have experimented on uniform LBP + SVM and achieved a relatively good precision in adult images. The next step is to experiment on using other feature representation and classifier. Also, there is a room for designing a way to predict accurate age, not just cluster sources into age groups.

Now we leave all images processing work on server. The bottleneck of our system heavily depends on matlab execution to recognizing features, cropping faces and extracting features. To reach a better efficiency and make it more extensible, our future work is to rewrite the algorithm pipeline in C++. Also, some less expensive computing works, such as image cropping and resizing, should be done by mobile clients before uploading to server.

To achieve a popular downloads on Google play, not only functions but also user interface should are considered important. We need to design a more interesting UI to increase the interaction between user and our application. Besides, it is a good idea to make users specifying face regions themselves, which will lead to a more accurate prediction.

To collect more data for further research, it is also important to design a feedback

function in our system. By doing so, our mobile client can thus become a way to collecting useful face images with ages.

## 6. Reference


[1] Ricanek, K.; Tesafaye, T., **MORPH: a longitudinal image database of normal adult age-progression**, *Automatic Face and Gesture Recognition, 2006. FGR 2006. 7th International Conference on* , vol., no., pp.341,345, 2-6 April 2006.

[2] Chih-Chung Chang and Chih-Jen Lin. 2011. **LIBSVM: A library for support vector machines**. ACM Trans. Intell. Syst. Technol. 2, 3, Article 27 (May 2011), 27 pages.

[3] Timo Ojala, Matti Pietikäinen, and Topi Mäenpää. 2002. **Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns**. IEEE Trans. Pattern Anal. Mach. Intell. 24, 7 (July 2002), 971-987.

[4] Caifeng Shan. 2010. **Learning local features for age estimation on real-life faces**. In Proceedings of the 1st ACM international workshop on Multimodal pervasive video analysis (MPVA '10). ACM, New York, NY, USA, 23-28.

[5] Ylioinas, J.; Hadid, A.; Pietikainen, M., **Age Classification in Unconstrained Conditions Using LBP Variants**, Pattern Recognition (ICPR), 2012 21st International Conference on , vol., no., pp.1257,1260, 11-15 Nov. 2012.

[6] Young H. Kwon and Niels da Vitoria Lobo. 1999. **Age classification from facial images**. Comput. Vis. Image Underst. 74, 1 (April 1999), 1-21.

[7] Xuehan Xiong; De la Torre, F., **Supervised Descent Method and Its Applications to Face Alignment**, Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on , vol., no., pp.532,539, 23-28 June 2013.

[8] Young H. Kwon and Niels da Vitoria Lobo. 1999. **Age classification from facial images**. Comput. Vis. Image Underst. 74, 1 (April 1999), 1-21.

[9] Bilgin Esme. 2011. **On Age Estimation by Using Still Face Images**.

[10] A. Gallagher, T. Chen, **Understanding Groups of Images of People**, IEEE Conference on *Computer Vision and Pattern Recognition,* 2009*.*

[11] **node.js**, http://nodejs.org/

[12] **npm - Node Package Modules**, https://www.npmjs.org/

[13] **Express - node.js web application framework**, http://expressjs.com/

[14] **Bootstrap**, http://getbootstrap.com/

[15] **Android SDK | Android Developers**, http://developer.android.com/sdk/index.html

# Appendix. Source Code Review

## 1. Algorithm Pipeline for Age Estimation

### (1) age_estimation

Input the file name of image and file name of output .txt file, it extract the features of theimage and vectorize it then output to the .txt file.

```
function output = age_estimation(pic_name, target_name)

   filename = target_name; %'age_vector.txt'
   fid = fopen(filename, 'w');

     rotate_face = eye_distance(pic_name);
     if rotate_face == 0
         output = 0;
         return;
     end
     %2:gray; 3:rgb
     if length(size(rotate_face)) == 3  %rpg
       img = rgb2gray(rotate_face);
     else %2, gray
       img = rotate_face;
     end

     face_pic = facecut2(img);
     if face_pic == 0
       output = 0;
       return;
     end

     fprintf(fid, '%s ', '0');

     features = extract_features(face_pic); %size: 531 * 1, length: 531
     for j = 1:length(features)
       fprintf(fid, '%d:%f ', j, features(j,1));
     end
     fprintf(fid, '\n');

   fclose(fid);

   output = 1;
   return;
end
```

## (2) eye_distance

Input the file name of image, then it get the coordinate of eyes of the face in that image for calling the function rotate face to rotate the image for normalization then return the rotated image.

```
function im = eye_distance(input)
 img=imread(input);

 % load model and parameters, type 'help xx_initialize' for more details
 [DM,TM,option] = xx_initialize;
   % perform face alignment in one image.
   faces =
DM{1}.fd_h.detect(img,'MinNeighbors',option.min_neighbors,'ScaleFactor',1.2,'MinSize',option.min_
face_size);
      if isempty(faces)
      im = 0;
   else
         output = xx_track_detect(DM,TM,img,faces{1},option);
         if ~isempty(output.pred)
            left_cen_x = double(output.pred(20,1) + output.pred(23,1))/2;
            left_cen_y = double(output.pred(20,2) + output.pred(23,2))/2;
            right_cen_x = double(output.pred(26,1) + output.pred(29,1))/2;
            right_cen_y = double(output.pred(26,2) + output.pred(29,2))/2;

            im = rotateface(img, [left_cen_x;left_cen_y],[right_cen_x;right_cen_y]);
                     else
            im = 0;
         end
    end
end
```

## (3) rotateface

Rotate the image by the coordinates of eyes to align the eyes (normalization)

```
function Cm = rotateface(Im, eye_left, eye_right)
   eye_dir = eye_right - eye_left;
   % translate left_eye to center
   center = floor([size(Im, 2); size(Im, 1)] / 2.0);
   translate_offset = center - eye_left;
   T = maketform('affine', [1 0 0; 0 1 0; translate_offset' 1]);
   Im = imtransform(Im, T, 'XData', [1-translate_offset(1) size(Im, 2)+translate_offset(1)],
'YData', [1-translate_offset(2) size(Im, 1)+translate_offset(2)]);
   % rotate original image around the left eye
   rotate_angle = atan2(eye_dir(2), eye_dir(1)) * 180/pi;
   Im = imrotate(Im, rotate_angle, 'bilinear', 'crop');
   % translate back to left_eye
   T = maketform('affine', [1 0 0; 0 1 0; -translate_offset' 1]);
   Cm = imtransform(Im, T, 'XData', [1+translate_offset(1) size(Im, 2)-translate_offset(1)],
'YData', [1+translate_offset(2) size(Im, 1)-translate_offset(2)]);
end
```

## (4) facecut2

Get the coordinates of eyes to call function cropface2 to cut the face image with front head and return it.

```
function face_img = facecut2(im, pic_name)

 % load model and parameters, type 'help xx_initialize' for more details
 [DM,TM,option] = xx_initialize;
   % perform face alignment in one image.
   faces =
DM{1}.fd_h.detect(im,'MinNeighbors',option.min_neighbors,'ScaleFactor',1.2,'MinSize',option.min_f
ace_size);
     if isempty(faces) %no face is detected
       face_img = 0;
       disp('no face detected');
             return ;
   else
     output = xx_track_detect(DM,TM,im,faces{1},option);
     if ~isempty(output.pred)
       left_bound = int16(min([output.pred(1,1), output.pred(20,1), output.pred(38,1)]));
       right_bound = int16(max([output.pred(10,1), output.pred(29,1), output.pred(38,1)]));
       low_bound = int16(max([output.pred(39,2), output.pred(40,2), output.pred(41,2),
output.pred(42,2), output.pred(43,2)]));
       %guess
       up_bound = int16(min([(output.pred(3,2) - (output.pred(25,2) - output.pred(3,2))),
(output.pred(8,2) - (output.pred(30,2) - output.pred(8,2)))]));
       %disp(right_bound);
       if up_bound < 1
           up_bound = 1;
       end
       left_cen_x = double(output.pred(20,1) + output.pred(23,1))/2 - left_bound;
       left_cen_y = double(output.pred(20,2) + output.pred(23,2))/2 - up_bound;
       right_cen_x = double(output.pred(26,1) + output.pred(29,1))/2 - left_bound;
       right_cen_y = double(output.pred(26,2) + output.pred(29,2))/2 - up_bound;
     else
         disp('output.pred is empty');
         face_img = 0;
        return;
     end
   end
   pic = im(up_bound:low_bound,left_bound:right_bound);
    face_img = cropface2(pic, [double(left_cen_x) ;double(left_cen_y)], [double(right_cen_x)
;double(right_cen_y)], [0.2;0.2], [210 ;210]);

end
```

## (5) cropface2

Cut the face image with front head and return it

```
function Cm = cropface2(Im, eye_left, eye_right, offset_pct, dest_sz)
   offset = floor(offset_pct .* dest_sz);
   eye_dir = eye_right - eye_left;
   eye_dist = sqrt(sum(eye_dir.^2));
   eye_ref_width = dest_sz(1) - 2.0*offset(1);
   eye_scale = eye_dist / eye_ref_width;

   % crop the rotated image
   crop_xy = -floor(eye_left - eye_scale*offset);
   crop_sz = floor(dest_sz*eye_scale);
   crop_rat = eye_scale*offset ./ eye_left;
   crop_size = floor([crop_rat(2) crop_rat(1)] .* size(Im));
   Im = imresize(Im, crop_size);

   %Im = Im(crop_xy(2):final_sz(2), crop_xy(1):final_sz(1));
   Cm = imresize(Im, dest_sz');
end
```

## (6) extract_features

Extract the feature of image by calling function LBP and LBP_histoc then return the vectors it gets from the two functions.

```
function F = extract_features(Im)
   % divide image into NxN regions
   N = 7;

   % bin of Uniform patterns: 58
   % bin of non-uniform patterns: 1
   BIN = 59;

   L = LBP(Im, 8, 1);
   F = zeros(N*N, BIN);
   row_step = size(L, 1) / N;
   col_step = size(L, 2) / N;
   for i = 1:N
       i1 = floor((i-1) * row_step) + 1;
       i2 = floor(i * row_step);
       for j = 1:N
           j1 = floor((j-1) * col_step) + 1;
           j2 = floor(j * col_step);
           patch = reshape(L(i1:i2, j1:j2), (i2-i1+1)*(j2-j1+1), 1);
           F((i-1)*N+j, :) = LBP_histoc(patch);
       end
   end

   F = reshape(F', N*N*BIN, 1);
end
```

## (7) LBP

Extract the feature on image by LBP method.

```
function LBPs = LBP(Im, P, R)


% transform into grayscale if input Im is colorful
if size(Im, 3) == 3
   Im = rgb2gray(Im);
end;

% size of each LBP sample
L = 2*R + 1;
% center of LBP sample
C = round(L/2);

%Im = uint8(Im);
nrows = size(Im, 1) - L + 1;
ncols = size(Im, 2) - L + 1;
LBPs = zeros(nrows, ncols);
for i = 1:nrows
   for j = 1:ncols
       A = Im(i:i+L-1, j:j+L-1);
       A = A-A(C,C);
       A(A>0) = 1;
       A(A<=0) = 0;
       % transform from binary code into decimal
       LBPs(i,j) = A(C,L) + A(L,L)*2 + A(L,C)*4 + A(L,1)*8 + A(C,1)*16 + A(1,1)*32 + A(1,C)*64 +
A(1,L)*128;
   end;
end;
```

## (8) LBP_histoc

Reduce the dimension of value get from function LBP

```
function H = LBP_histoc(x)
   % {0-255} -> {0-59}
   % ind from 1-256
   TABLE = [1 2 3 4 5 0 6 7 8 0 0 0 9 0 10 11 12 0 0 0 0 0 0 0 13 0 0 0 14 0 15 16 17 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 18 0 0 0 0 0 0 0 19 0 0 0 20 0 21 22 23 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 24 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 25 0 0 0 0 0 0 0 26 0 0 0 27 0 28 29 30 31
0 32 0 0 0 33 0 0 0 0 0 0 0 34 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 35 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 36 37 38 0 39 0 0 0 40 0 0 0 0 0 0 0 41 0 0 0 0 0 0 0 0 0 0 0 0 0 0
42 43 44 0 45 0 0 0 46 0 0 0 0 0 0 0 47 48 49 0 50 0 0 0 51 52 53 0 54 55 56 57 58];

   % 59 bins
   %xp = (xp - mean(xp)) / std(xp);
   H = histc(TABLE(x+1), 0:58);
   H = H / sum(H);
end
```

## 2. HTTP Server

Our HTTP server is stateless. It is responsible for displaying uploaded images and calling image processor and classifier to obtain estimated ages. The server only responses to 3 types of HTTP requests:

**(1) HTTP GET /**

**Response: a web page containing all uploaded images and each with a classified age group**

```
var AGE_GROUP = {
   0: '20-29',
   1: '30-39',
   2: '40-49',
   3: '50-59',
   4: '60+',
   5: '15-19'
};
app.get('/', function(req, res) {
   var files = fs.readdirSync('./public/img/');
   var images = [];
   // loop through image folders
   for (var i = 0; i < files.length; i++) {
       if (mime.lookup(files[i]).split('/')[0] != 'image')
           continue;
       var label = getLabelFromImagePath(files[i]);
       if (parseInt(label, 10) < 0) continue;
       var group = AGE_GROUP[label];
       images.push({
           path: 'img/'+files[i],
           group: group
       });
   }
   // render those images
   res.render('index', {images: images});
});
```

**(2) HTTP GET /age/<imageId>**

**Response: a JSON Object {“ageGroup”: <tag>} telling the classified result of the image given imageId. If the classification is not done yet, <tag> is null.**

```
app.get('/age/:id', function(req, res) {
   var imageId = req.params.id;
   if (imageId == null) return res.status(404);

   console.log('[DEBUG] Looking for image: ' + imageId);

   var files = fs.readdirSync('./public/img/');
   var label = '-3';
   //files.map(function(v) { return 'img/'+v; });
   for (var i = 0; i < files.length; i++) {
       if (mime.lookup(files[i]).split('/')[0] != 'image')
           continue;

       var name = files[i].split('/').slice(-1);
       var id = getIdFromImagePath(files[i]);
       console.log('[DEBUG] Image: ' + id + ' @ ' + files[i]);
       if (id != null && id === imageId) {
           label = getLabelFromImagePath(files[i]);
           break;
       }
   }
   if (parseInt(label, 10) < 0) return res.status(404);

   // send predicted age group
   res.send({ageGroup: label});
});
```

**(3) HTTP POST /age with payloads containing images**
**Response a unique <imageId> corresponding to the uploaded image**

```
app.post('/age', function(req, res) {

   //console.log('[Debug] files: ' + JSON.stringify(req.files));
   if (req.files && req.files.image) {
       var ts = +new Date();
       var tmpPath = req.files.image.path;
       var ext = tmpPath.split('.').slice(-1);

       // (1) send back image id
       res.send({imageId: ts});

       // (2) do age estimation
       var label = '-4';
       ageEstimation(ts, tmpPath, function (e, _label) {
           if (e) throw e;
           console.log('[DEBUG] Age Estimation completed! ' + _label);
           label = _label;
       });

       setTimeout(function () {
           if (parseInt(label, 10) < 0) {
               return console.log('[Error] No label: ' + label);
           }

           var group = AGE_GROUP[label];
           console.log('[DEBUG] classified label: ' + label);
           console.log('[DEBUG] Estimated age: ' + group);

           var path = './public/img/' + ts + '-' + label + '.' + ext;
           console.log('[DEBUG] Image is saved to ' + path);
           fs.renameSync(tmpPath, path);

       }, 15000); // wait for 10 seconds
   }
   else
       res.status(400);
});
```

## 3. Android Client

Our client is only for sending images and querying results from the server. The key point is the logic to handshake with the server.
The following code segment launch a post request uploading an image to server. After obtain the imageId for that image, it wait in background threads and retry for the actual result done.

```
AgestimatorRestClient.post("/age", params, new JsonHttpResponseHandler() {
	@Override
	public void onSuccess(JSONObject res) {
		try {
			String imageId = res.getString("imageId");
			// wait and get
			new RequestAgeEstimationTask().execute(imageId);
		} catch (JSONException e) {
			Log.e(TAG, "Error in parsing response json: " +
				e.toString());
		}

	}
});

private class RequestAgeEstimationTask extends
			AsyncTask<String, Void, String> {
	protected void onPostExecute(String imageId) {
		AgestimatorRestClient.get("/age/" + imageId, null,
					new JsonHttpResponseHandler() {
			@Override
			public void onSuccess(JSONObject res) {
				try {
					String label = res.getString("ageGroup");
					vibrate();
					updateAgeInfo(label);
				} catch (JSONException e) {
				Log.e(TAG, "Error in parsing response json: " +
						e.toString());
				}
			}
		});
	}

	@Override
	protected String doInBackground(String... strs) {
	for (int i = 0; i < 20; i++) {
		try {
				// keep waiting
			Thread.sleep(1000);
			} catch (InterruptedException e) {
			Thread.interrupted();
			}
		}
		String imageId = strs[0];
		return imageId;
	}
}
```