IdentiFace: FullContact Field Session 2015

Jess Boerner, Ross Bunker, Jacob Emmel, Michael Harvey
June 16, 2015
## CONTENTS

1. Introduction 1

2. Requirements 1
   2.1. Functional Requirements 1
   2.2. Non-Functional Requirements 1

3. System Architecture 2

4. Technical Design 2
   4.1. Object Detectors and Training 2
   4.2. Face Detection 5
   4.3. Gender and Age Estimation 5
   4.4. Post Processing 6

5. Design and Implementation Decisions 6

6. Results 9
   6.1. Future Work 9
   6.2. Performance Testing Results 10
   6.3. Proof of Usability 11
      6.3.1. API Calls 11
      6.3.2. JSON Responses 12

Appendix A. Precision, Recall, and F-Score Definitions 14

Appendix B. Object Detection Using a Histogram of Oriented Gradients 15

Appendix C. Structure and Explanation of JSON Output 16
1. INTRODUCTION

Our client, FullContact, is a Denver-based startup that is working to provide a cloud-based contact solution. Up to 25,000 contacts from three Gmail accounts and various other services can be unified with no duplicates in FullContact’s database. These contacts are synced across all linked accounts every 20 minutes, and are intelligently enhanced with other information about the individual represented by the contact. There are also iPhone and iPad apps, so the information can be accessed on-the-go. FullContact’s service focuses on contact management, so that their users don’t have to.

FullContact enhances a contact by drawing in as much information about the individual as possible, and refining what the people who have linked to the contact see. If multiple photos of an individual have been found, the system has the capability to remove any duplicates, but still lacks the ability to do anything further with the set of photos. The goal of this field session was to implement machine-learning driven analysis of additional photos. This analysis includes detection of faces, estimation of age and gender, and collection of other metrics such as how much of the photo area is taken up by the face. This data can be utilized twofold: the vestiges of an algorithmic photo ranking system can be put in place based on the number and size of faces, and profiles can be enriched by the automatic addition of gender and age information. Our solution utilizes several levels of analysis in order to detect faces and estimate and refine guesses about age and gender of individuals in single-face pictures.

2. REQUIREMENTS

2.1. FUNCTIONAL REQUIREMENTS

• The service should act as a "black box," accepting one image and returning information about it.
  – The service should be accessible by FullContact, but not outside users.
  – The service should be accessible by HTTP GET or POST requests.
  – The user should be able to supply an image’s URL or byte data to be evaluated.
  – The service should return data in JSON format.

• The service should determine the number of and locations of faces present in the image, as well as the percentage of the image’s area occupied by each face.

• If only one person is present in an image, the service should estimate that person’s gender and age.
  – Ages will be estimated within seven buckets defined by FullContact.

2.2. NON-FUNCTIONAL REQUIREMENTS

• The service should be installed on an Amazon web server running Ubuntu.
• The number of external libraries used by the service should be minimized, at the discretion of the Field Session team.

• FullContact should be able to use values of precision and recall, along with $F_{0.5}$-scores, to compare the service created by the Field Session team to commercial solutions for similar problems. See Appendix A for descriptions of these values.

3. SYSTEM ARCHITECTURE

4. TECHNICAL DESIGN

4.1. OBJECT DETECTORS AND TRAINING

The face detection module, as well as the gender and age estimation module, use object detectors from dlib, a general-use C++ library, which features a Python API. Dlib implements a Histogram of Oriented Gradients ("HOG") approach to object detection, and includes utilities for easy training. See Appendix B for additional information on HOG-based detection.
Because of the large sizes of the subsets, which included sets up to 6,064 images in size, and the memory used by the training program, we were unable to train the support vector machine files using our personal computers; instead, we used an Amazon Web Server for training, running scripts to create each SVM file in succession without our direct input.

Dlib's object detector trainer accepts three parameters that alter the training: $C$, $\epsilon$, and a boolean value for mirroring images. If images are mirrored by the trainer, both the original image and its horizontal reflection are used in training; this can be used to double the size of training sets that feature symmetric objects. As faces are generally symmetric, we enabled this feature. $C$ is a constant value for the support vector machine, where larger $C$ values lead the SVM to seek closer fits to its training data; mathematically, $C$ must be greater than zero. The value of $\epsilon$ is used by the trainer to determine when training should end, with smaller values leader to more training; $\epsilon$ must be greater than zero, and typical values are less than one.

Dlib's example training program uses the values $C = 5$ and $\epsilon = 0.01$ for training, so we initially used the same values. Once we gained access to the server, we created a new script to train a new group of object detectors for the age range 35-44 with varying $C$ and $\epsilon$ values; this bucket was selected because it featured a large subset of our training images, but not a majority. After training twenty-five new SVM files, we ran a testing script to determine the effects of modifying the values on the precision, recall, and $F_{0.5}$-score of our age estimator.

As shown in figure 4.1, our initial values of $C = 5$ and $\epsilon = 0.01$ produced a precision "hotspot." Additionally, $C = 7$ and $\epsilon = 0.015$ yielded a high precision result.

Recall values were highest for $C$ values greater than or equal to seven, with a high value at $\epsilon$ of nine; results for recall are shown below, in figure 4.2.
To observe the overall results of altering $C$ and $\epsilon$, we also computed the $F_{0.5}$ scores of each test. The results are shown in figure 4.3.

This training yielded the values $C = 7$ and $\epsilon = 0.015$ as optimal values, so we trained the
full set of nine SVM files with these values; however, the results were mixed, with no clear improvements, so we decided not to use the new files in our final system.

4.2. FACE DETECTION

The face detection module implements dlib's default face detector, an object detector trained on approximately 3000 images taken from *Labeled Faces in the Wild*, a face photo database created to aid in the study of face recognition algorithms. When provided an image, the module first determines the number of times it should be upsampled by the face detector. If the image's area exceeds a maximum threshold value—equivalent to the area of a 1300px by 1300px image—it is not upsampled, in order to shorten the execution time of detection. Smaller images are upsampled at least once; if either their width or height falls below 160px—two times the detection window's size—they are upsampled a second time, allowing for the detection of even smaller faces.

The face detector returns a list of bounding boxes for each face it found, containing location and size information about the faces. This is used by the face detection module to calculate the percentage of the image's area occupied by each face, which is returned to the control module alongside the box data.

4.3. GENDER AND AGE ESTIMATION

If an image is determined to only contain one face, it is passed to the gender and age estimation module. Unlike the face detection module, which uses a single detector provided by dlib, this module implements a series of custom-trained detectors, optimized for the detection of faces from particular groups of people. To train these detectors, we used the Adience image set from the Open University of Israel, a collection of approximately 10,000 images of faces with accompanying age and gender information. One copy of the set was split into male and female directories, which were used to train male-focused and female-focused detectors, respectively. A second copy of the image set was divided into seven directories, based on the seven age buckets of interest to FullContact: 18-20, 21-24, 25-34, 35-44, 45-54, 55-64, and 65+. Seven corresponding object detectors were then trained for use in age estimation.

For each piece of demographic information desired, the module runs all of the relevant object detectors on an image, upampling using the same scheme as the face detection module. For example, in age estimation, the module applies each of the seven age bucket-focused detectors to the image. Each detector returns a "score" value, corresponding to how closely the face it found matches its HOG filter; if no face is found, the value -1 is set as the score. These scores are positive numbers, but do not have a maximum possible value, although they typically fall within the range (0, 3).

Naively, the categories with the highest score can be chosen as the estimations for the demographic data; the accuracy of this approach is described in the Results section. However, by passing the raw values to another module for postprocessing, we hoped to be able to improve the outputs.
4.4. Post Processing

In an effort to make age and gender estimation ‘smarter,’ raw score values are piped into a post-processing module which implements a much lower level of machine learning. The goal is for our black box to ‘learn’ its shortcomings in an intuitive way. First, a model must be trained for age and gender raw scores that maps into the correct age bucket and gender choices. Once we have a model, we can analyze any image using the raw scores from age or gender estimation and predict its classifications. In order to maintain a uniform data set across all solutions we used 362 images in training post-processing models for age and gender. This allowed us to test performance of the post-processing models against the same 100 image set used to test all commercial solutions. To achieve a low level of machine learning processing, we chose libSVM with its ‘default’ settings for linear classification. libSVM includes a few tools to ensure the models are not over-fitted and that parameters are picked optimally with respect to the data. Unfortunately the data size we are dealing with is not large enough to train a respectable model for post-processing. Precision scores with post-processing were lower than those without. Recall scores improved for some age buckets, but the significance is low since we have prioritized precision over recall.

5. Design and Implementation Decisions

Our first design decision regarded whether we would pursue an implementation of an open source library, or work with a pre-built commercial solution. To determine which path was best, all possible commercial and open-source solutions were researched. There were seven commercial solutions and three open alternatives:

- **Commercial Solutions**
  - AlchemyAPI
  - emoVU
  - Face++
  - InSight API
  - KeyLemon
  - ReKognition API
  - SkyBiometry

- **Open Source Solutions**
  - dlib
  - OpenBR
  - OpenCV

Once all of the possibilities were known, each solution was explored and implemented to the point where test data on its performance could be collected. A standardized set of test images was used to determine the precision and recall of all of the solutions, commercial and open
source, and a set of graphs visualizing the data was created. One commercial solution, emoVU, was immediately discarded due to the lack of resolution in its age estimation buckets. OpenBR was eliminated due to the fact that it is an implementation of OpenCV, with some additional GUI work tied on top. OpenCV is already a black box, with a few exposed control parameters to work with its machine learning algorithms, and OpenBR just added an additional layer of abstraction on top of this. OpenBR would have been ideal, if an open solution was desired but there was not time to train classifiers or deal with smaller details. However, one of the open source solutions, dlib, proved to be competitive with commercial solutions; a comparison is provided in figure 5.1. Even without training on the full image set, its results approached those of the commercial solutions.

Figure 5.1: Comparison of our initial dlib-based solution's $F_{0.5}$-scores to those of commercial solutions.

Additionally, dlib proved to be better than OpenCV in many ways; an example of the results of its default face detector compared to OpenCV's is shown in figure 5.2 below.
Due to its reasonable precision and recall, and low operating cost relative to commercial solutions, dlib was chosen as the main library with which to move forward.

While working with the preliminary implementations, the team all learned Python independently (no one had spent more than a few hours with it prior to field session). Many of the libraries we needed had either Python wrappers over a C++ library, in the case of dlib, or were fully written in Python. This, in conjunction with how easy Python is to use, lead the team to code the remainder of the project in Python. There were a few bash scripts created for testing and to automate training of multiple files, but all of the actual functional code was written in Python.

Due to the requirements of the client, we broke the structure of our program down into several modules. We needed a web interface, in order that the functionality of the program could easily be added to FullContact’s existing backend. For this, Tornado (a Python web server) was chosen, due to its extensive usage in other projects and comprehensive documentation. We also had two main functions required of the program: to be able to return detection data on the number and location of faces in an image, and to be able to return gender and age estimations on a single face. Despite the fact that both of these functions rely on dlib, it was decided that we should separate them into two separate modules, in the interest of increasing the orthogonality of the system.

We also needed to determine how information should be passed between different modules, once we had laid out the function that each would perform. We became quite fond of the ‘dict’ structure built into Python as a means to transport complex groups of values. Unlike an array, where values can be retrieved by index but have no guarantee of being the desired value, a dict requires keys, and will throw a KeyError exception if an invalid key is referenced. We also used ‘native’ types as much as possible, like the ‘ndarray’ structure in numPy. Both dlib and skimage, another image-processing library we used, natively utilize the ndarray type when dealing with images (i.e., no conversion is necessary). By loading all images into ndarrays, and catching all potential exceptions at the loading stage, we could ensure the correct function of modules downstream from that point, as all other sources of error had already been eliminated or preempted by guaranteeing the type.
6. Results

6.1. Future Work

While our system's performance is competitive when compared to paid, commercial solutions, there are many areas in which it can be improved. It could be trained with larger data sets than were used. The sets used were chosen because of their size and availability; no larger and properly-curated (containing age and gender information for each subject) sets were available. The training process could be improved, however, if larger sets are collected and curated by hand. Alternately, a system could be implemented whereby the age and gender information filled out by users for their contacts could be utilized as new training points for the age and gender estimation patterns. This would enable crowd-sourced training of the machine-learning algorithms, and eliminate the necessity for having to curate large data sets by hand. There are also several parameters involved with the construction of SVMs, that were only briefly explored. A deeper understanding of machine learning would facilitate the correct utilization of the parameters to train most effectively off the data. The throughput of the system could also be improved with additional time. Currently, the system checks how closely a given image matches patterns for each of the genders and all seven of the age buckets. Algorithmic optimization may be possible, where certain patterns are not checked against, based on the responses from checks against other patterns (i.e., if 65+ returns a strong response, 55-64 a marginal response, and 45-54 a weak response, it is probably not necessary to check against the 18-20 pattern). The code is also not parallelized, beyond some built-in multicore functionality during the training module (which is never utilized, except in setting up the system). 'Internal' and 'external' parallelization could be implemented, where 'internal' checks against multiple patterns simultaneously for a single image, and 'external' will allow for the testing of multiple images simultaneously. Furthermore, a system will need to be implemented to actually rank an individual's pictures, based on the data we produce. A naive algorithm would be to first bucket pictures based on whether they contain one face, more than one face, or no faces, and then sort the photos in each bucket by the percentage of the total area that the face(s) take up. FullContact discussed the importance of collecting the particular metrics during the development phase, but since our focus was image analysis, they did not wish for us to actually implement the ranking, as that is more fit for deployment elsewhere in their stack. As a final note, there are many evaluations that can be made from the raw data we return (such as confidence levels, etc.), but all depend on arbitrary thresholds. As such, the usefulness of the numbers that would be computed is dependent on the threshold values, which are far harder to determine than it is to write the code utilizing them. Due to this, FullContact will implement these in the future, if they have the necessity for doing so.
6.2. Performance Testing Results

Figure 6.1: Comparison of our final solution's F$_{0.5}$-scores to those of commercial solutions.

Figure 6.2: Comparison of dlib F$_{0.5}$-scores to those of commercial solutions.
Figure 6.3: Comparison of $F_{0.5}$-scores for each stage of our dlib-based solution.

N/A score explanation: When calculating composite precision and recall scores, we came into the issue of representing any NaN or N/A results in a uniform way. In these instances we weighted the composite score on *only the data that was valid, not the whole data set including NaN results.*

6.3. PROOF OF USABILITY

Since this service was designed to mimic the function of commercial facial-recognition API’s, sample inputs and outputs from a commercial API will be compared side-by-side with ours as proof of usability.

6.3.1. API CALLS

Call to ReKognition API: http://rekognition.com/func/api/?api_key=KFracWMfLOVINm1N&api_secret=AjKE5STTInnkLE59&urls=http://www.dyslexiaassociation.ca/gallery/famous/AlbertEinstein.jpg

Call to IdentiFace API: http://localhost:8888/?url=http://www.dyslexiaassociation.ca/gallery/famous/AlbertEinstein.jpg

Note: Once productionized, there will only be internal access to the API, for now it is most easily accessed by SSH’ing in to our Amazon server with local port forwarding.
6.3.2. JSON RESPONSES

Return from ReKognition API:

```json
{
    "url": "http://www.dyslexiaassociation.ca/gallery/famous/AlbertEinstein.jpg",
    "face_detection": [
        {
            "boundingbox": {
                "tl": {
                    "x": 102.31,
                    "y": 122.31
                },
                "size": {
                    "width": 227.69,
                    "height": 227.69
                }
            },
            "confidence": 1,
            "pose": {
                "roll": 1.01,
                "yaw": 7.38,
                "pitch": -9.96
            },
            "quality": {
                "brn": 0.59,
                "shn": 2.1
            },
            "age": 77.56,
            "sex": 1
        }
    ],
    "ori_img_size": {
        "width": 416,
        "height": 504
    },
    "usage": {
        "quota": 19864,
        "status": "Succeed."
    }
}
```
Return from IdentiFace API:

{  
    "faceCount": 1,  
    "faceData": [
      {
        "boundingBox": {
          "topLeft": {
            "y": 157,
            "x": 115
          },
          "size": {
            "width": 187,
            "height": 186
          }
        },
        "gender": "M",
        "age": "65+",
        "percentArea": 16.59,
        "rawData": {
          "maleScore": 1.26247,
          "55-64Score": 0.19381,
          "65+Score": 1.20573,
          "femaleScore": 0.96292,
          "18-20Score": 0.74663,
          "25-34Score": 1.15673,
          "genderConfidence": 0.5673028098445666,
          "ageConfidence": 0.20242698153405514,
          "35-44Score": 0.97116,
          "21-24Score": 0.79010,
          "45-54Score": 0.89221
        }
      }
    ],
    "imageDims": {
      "width": 416,
      "height": 504
    }
}
A. Precision, Recall, and F-Score Definitions

0 Precision and recall are numerical values used to evaluate the performance of a classifier that returns either a positive or negative value. Precision measures how relevant the returned results are, using (A.1); it is not defined when the classifier returns no positives.

\[
\text{Precision} = \frac{\# \text{True Positives}}{\# \text{True Positives} + \# \text{False Positives}} \tag{A.1}
\]

Recall measures how many of the actual relevant items were returned, using (A.2); it is not defined for scenarios where there are no potential positive results.

\[
\text{Recall} = \frac{\# \text{True Positives}}{\# \text{True Positives} + \# \text{False Negatives}} \tag{A.2}
\]

F-scores, or F-measures, are numerical values that combine precision and recall information into a single number. The \(F_\beta\)-score is used to measure the performance of a classifier by a user that values recall \(\beta\) times as much as precision; the formula used to compute \(F_\beta\) follows, in (A.3).

\[
F_\beta = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{(\beta^2 \cdot \text{Precision}) + \text{Recall}} \tag{A.3}
\]

In our testing, we used the \(F_{0.5}\)-score, as FullContact valued precision twice as much as recall.
The face detectors used in our system are based on an object detection method that uses Histograms of Oriented Gradients ("HOGs"). This method attempts to determine the shape of an object to be sought by recording the orientation of edges within an image. Edges, in turn, are described by intensity gradients, which indicate how the intensity of colors change in an image. (Note that the colors themselves are not taken into account.) By training on a set of images with the desired object’s locations indicated, a HOG filter can be created, which shows the expected gradients associated with the object to be found. For example, the filter used by dlib’s provided face detector is visualized in figure B.1.

When creating the HOG filter for an object, a training program divides an image into cells, and calculates the gradient for each pixel within a cell and records their orientation; this record is the histogram that gives the method its name. Histograms for local groups of cells, called blocks, are collected; blocks may overlap in order to reduce potential noise in a single block.

To search for an object in an image, a program effectively slides a HOG filter across the image, searching for matches using a support vector machine classifier. If instructed, the program will upsample the image and run the detector multiple times, allowing the classifier to locate smaller matches that would not have been caught previously.
C. STRUCTURE AND EXPLANATION OF JSON OUTPUT

Although there is no strictly standardized method for outputting data from a facial analysis API, the commercial solutions all had certain ubiquitous structural elements. Using these as guidelines, a proposed format was submitted to key employees at FullContact in order to refine it to meet their requirements. After several revisions, the final format was determined as follows:

```json
{
    "faceCount":  # of faces,
    "faceData": [ {
        "age": "(age bucket)",
        "gender": "(M, F)",
        "boundingBox": {
            "topLeft": {
                "x": 123,
                "y": 123
            },
            "size": {
                "width": 123,
                "height": 123
            }
        },
        "percentArea": 12.3,
        "rawData": {
            "maleScore": 0.12,
            "femaleScore": 0.12,
            "18-20Score": 0.12,
            "21-24Score": 0.12,
            "25-34Score": 0.12,
            "35-44Score": 0.12,
            "45-54Score": 0.12,
            "55-64Score": 0.12,
            "65+Score": 0.12,
            "ageConfidence": 0.12,
            "genderConfidence": 0.12
        }
    } ],
    "imageDims": {
        "width": 123,
        "height": 123
    }
}
```
In the event that no faces are returned, `faceCount` will be 0, and the `faceData` array will be an empty array. `imageDims` will still contain information about the size of the original image.

When exactly one face is detected, the output will be exactly as outlined above. Entries with an "*" in front of them will only be returned in this case, when one face is detected.

If more than one face is detected, `faceCount` will contain an integer value representing how many faces were detected. The `faceData` array will contain entries equal to the number of faces detected, with each entry containing information about the location of the face, but with none of the values denoted with an asterisk.

`rawData` contains all of the values produced by the age and gender estimation module. These are passed into the post-processing module, which will return a finalized estimate of age and gender, along with the confidence values associated with the estimates. The age and gender confidence values are derived from the individual scores in a given category using (C.1), and they serve as an indicator of how sure the program is about the corresponding estimates it has produced.

\[
Confidence \text{ in guess from scores} = \frac{\max(scores)}{\sum scores} \tag{C.1}
\]

The reason they have been included is that it is better to not guess in cases where it is not fairly clear which demographic classifications apply to the individual in the photo. In this manner, precision can fairly easily be improved, at the expense of recall.