About Face Recognition

There are a lot of details that you can get by reading the code, and tunable parameters that can be tweaked; I'm going to focus here on the main algorithm.

Working

First, a training set is captured for each person in the training set, eight images that each contain a single face are captured, qualified, and stored to the auto captured folder labeled with the person's name. After all training images (persons) are captured, the problem is essentially a "Detect-Extract-Match" problem. The unique thing about the approach herein is that features are detected and extracted for each person in the training set *from a single stacked-image montage* of that person's photos. The image stream is then restarted and faces are auto-detected. Features are extracted from each detected face and matched to the set of features for each face in the training set. The one that minimizes the sum of absolute differences (SAD) between images is assumed to be the person in the incoming image. Labels are inserted to indicate the name of that person.

Note that each face will be matched to one of the trained feature sets; I have not implemented a way to indicate that a person is not in the training set.

More Detailed

1) Note that USB webcams are supported in core MATLAB. Other cameras are supported via interfaces in the Image Acquisition Toolbox.

2) Instantiate a face detector:

```
faceDetector = vision.CascadeObjectDetector('MergeThreshold',10);
```

Note that this requires the Computer Vision System Toolbox (and its dependencies).

3) In a `while` loop, capture a snapshot of a scene. Convert it to grayscale, and if desired, preprocess it. Pre-process options are user-tunable:

```
RGBFrame = snapshot(vidObj);

grayFrame = rgb2gray(RGBFrame);
if preprocessOpts.matchHistograms %logical flag
grayFrame = imhistmatch(grayFrame,...
     preprocessOpts.targetForHistogramAndResize);
end

if preprocessOpts.adjustHistograms
   grayFrame = histeq(grayFrame);
end
```

4) Detect faces in the captured images, and qualify the detections. Qualification includes verifying that there is exactly one detection in the image, and that is above a size threshold:

```
bboxes = faceDetector.step(grayFrame);

if isempty(bboxes) || size(bboxes,1) > 1 || any(bboxes(3:4) <
QC.minBBSize)
   return
end
```

5) Once a face is detected, optionally do some quality control checks. For instance, make sure that the "face" contains one mouth.

```
if QC.oneMouth
   mouthBox = QC.mouthDetector.step(grayFrame);
   if size(mouthBox,1) ~= 1
      return
   end
end

% (If we made it to here, the capture was successful!)
```

Automatically crop the detected faces (using the bounding boxes returned by the detection algorithm), preprocess as desired, resize to a standard size, and store to an auto-created directory:

```
faceImg = imcrop(grayFrame,bboxes);
faceImg = imresize(faceImg,thumbSize); %thumbSize is a
configurable parameter
imwrite(faceImg,imagePath);
```

6) Repeat steps 3—5 until 8 images are captured. (This is a configurable parameter, too: `nOfEach = 8;`). Then `continue` the `while` loop until images have been captured for the last person to be trained. Each person's images are automatically written to a different subfolder in a parent capture directory. The operator will be prompted for the names of the persons in the training set; these names will be associated with their respective capture folders. (The operator will also be afforded an opportunity to remove bad images from the training set. It is not necessary that each person has the same number of training images.)

7) Now we slurp in all of the captured face images and train a detector!

```
imgSet = imageSet(targetDirectory,'recursive');
sceneFeatures = trainStackedFaceDetector(imgSet);
```

TRAINING ALGORITHM

8) For training, I use an approach I call "aggregated features from stacked images." There's no reference to steer you to on this. It's just an approach that occurred to me—one that yields very good results fast with relatively few images, and that is easy to implement. In a nutshell:

a) For each face in the training set, create a stacked-image montage of 5* randomly images randomly selected from the images of that person. (Notice that by default we captured 8 images of each, but are training on only 5. (This number, too, is user-configurable) That simplifies the situation if some of the images were discarded for poor quality in step 6.

b) Instantiate a feature detector, and detect, extract, and store features from each montage. I tried many different detectors and extractors, with many different input parameters, before settling on FAST features, and a SURF extractor:

```
fcnHandle = @(x) detectFASTFeatures(x,...
   'MinQuality',0.025,...
   'MinContrast',0.025);
extractorMethod = 'SURF';
scenePoints{ii} = fcnHandle(trainingImage);
[sceneFeatures{ii}, scenePoints{ii}] = ...
   extractFeatures(trainingImage, scenePoints{ii},...
      'Method',extractorMethod);
```

c) We now have a set of features (sceneFeatures) calculated for each face. This set of features can now be used as a classifier! (If all went well, the feature extraction should have taken only a few seconds!)

RECOGNIZE

9) Restart the streaming image capture, and detect faces. Crop each detected face (again, using the bounding box returned by the face detector), and detect and extract features in each. (The same preprocessing steps described above are implemented here). Match the extracted feature set to each of the feature sets extracted from the training-image montages, and find which training-set image it matches most closely. That one is the "best guess prediction." Extract the label corresponding from that closest-match from the name of the parent directory for the training images. And that's it!

(Note: after much experimentation, I implemented a minimization of the sum-of-absolute-differences to determine the best guess.)

```
RGBFrame = snapshot(vidObj);
grayFrame = rgb2gray(RGBFrame);
bboxes = faceDetector.step(grayFrame);

for jj = 1:size(bboxes,1)
   bestGuess = myPrediction(thisFace,sceneFeatures,numel(imgSet));
   bestGuess = imgSet(bestGuess).Description;
end
```

Where:

myPrediction is:

```
metric = 'SAD'; %#ok
boxPoints = fcnHandle(testImage);
[boxFeatures, boxPoints] = extractFeatures(testImage, boxPoints,...
    'Method',extractorMethod,...
    'BlockSize',3,...
    'SURFSize',64);
matchMetric = zeros(size(boxFeatures,1),nFaces);
for ii = 1:nFaces
  [~,matchMetric(:,ii)] = ...
     matchFeatures(boxFeatures,sceneFeatures{ii},...
         'MaxRatio',1,...
         'MatchThreshold',100,...
         'Metric',metric);
end
[~,detectionIndex] = min(mean(matchMetric));
```