Computer Vision Coursework Report

Jialin Yu

April 2017

Contents

List of Figures

Chapter 1

Introduction

In this coursework, we are given a g20 picture. We need to solve two tasks for this coursework, implement face detection and face recognition.

This report consists 4 sections. The first two sections discussed solutions for face detection. The last two sections discussed solutions for face recognition.

Chapter 2

Implementation and Testing

This is the chapter for your Literature Survey.

2.1 Req #1

I start with implementing the correlation algorithm for face detection. Such algorithm takes a template as a kernel then convolve it with the given image. The result would be a matrix, each element of that matrix describes the pixel similarity between the template and the given image.

This algorithm can be analogized by doing the dot product between the template and a moving window moving across the given image. Window with the highest value represent such region is very similar to the template.

In Matlab, the above algorithm can be implemented by the following code:

```
[imRows,imCols,nBands] = size(imf);

 dotProd = zeros(imRows,imCols);
 for i = 1:nBands


     dotProdi = filter2(template(:,:,i),imf(:,:,i),'same');
     dotProd = dotProd + dotProdi;

 end;
```

Where `imf` stands for input image and `filter2` is the convolution operator without flipping.

In theory, if given image contains the template, then the global maxima region should be the template itself, and the local maxima could be some object that is visually similar to the template. In order to implement face detection based on correlation, using non-maxima

suppression to find the local maxima. Bellow shows three template matching output results:

Figure 2.1: Template matching result with Jintao, Cameron and Obama

The template is manually cropped from original image which is highlighted in **red**. The above results produce 30.80%, 37.07% ,35.94% overlap with Viola-Jones detections respectively.

Above results use different template which various in template size and their visual content. Therefore, to obtain a consistent detection output, I use a fix image:

Figure 2.2: Fix template

as template which outputs:

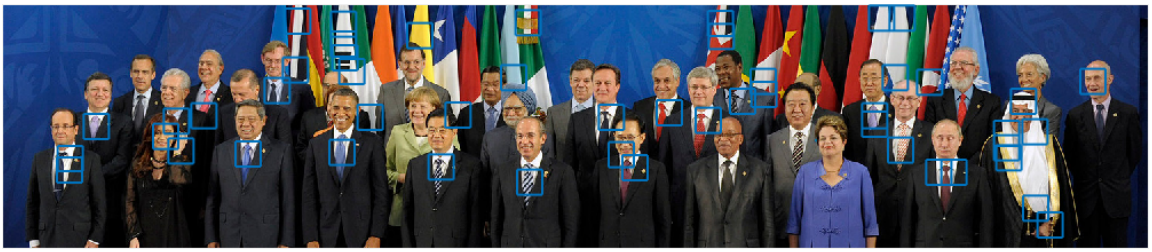

Figure 2.3: Template matching result with fixed image

with 13.78% overlap with Viola-Jones detections.

Using the above method ignores the impact of inconsistent of pixel intensity. Therefore, we can improve the above method by using normalized correlation.

A normalized correlation can be implemented by calculating `r`:

$$r = \frac{(\mathbf{x} - m_x\mathbf{1})^T(\mathbf{y} - m_y\mathbf{1})}{\|\mathbf{x} - m_x\mathbf{1}\|\|\mathbf{y} - m_y\mathbf{1}\|}$$

where `1` stands for vector of 1s and m_x,m_y stand for `mean` of x and y. The above equation can be expressed by `inner product` of

$$\frac{x - m_x\mathbf{1}}{|x - m_x\mathbf{1}|}, \frac{y - m_y\mathbf{1}}{|y - m_y\mathbf{1}|}$$

Operator $|\boldsymbol{x}|$ computes the magnitude of a vector x:

$$|\boldsymbol{x}| = \sqrt{x_1^2 + x_2^2 + ...x_n^2}$$

Thus, we can compute —x— by:

$$\sqrt{Var \times N}$$

where Var is the variance of x:

$$\frac{1}{N} \times \sum_{n=1}^{N} (x_n - \mu)$$

So, in Matlab code:

```
for i = 1:nBands
    meanTem = mean2(template(:,:,i));
    meanImf = mean2(imf(:,:,i));
    stdTem = std2(template(:,:,i));
    stdImf = std2(imf(:,:,i));
```

```
    dotProdi = filter2((template(:,:,i)-meanTem)/sqrt(stdTem^2 *
        numel(template(:,:,i))) ...
         ,(imf(:,:,i)-meanImf)/sqrt(stdImf^2 * numel(imf(:,:,i)))) ...
         ,'same');
    dotProd = dotProd + dotProdi;
end;
```

2.1.1 Result

Running above code gives the output of normalized correlation:

Figure 2.4: Template matching with normalized correlation result with Jintao, Cameron and Obama

The above results produce 34.23%, 44.01% ,36.56% overlap with Viola-Jones detections respectively. After using normalized correlation, we can observe slight improvement on matching accuracy. However, for the fixed template the result still remain the same with 13.78%:

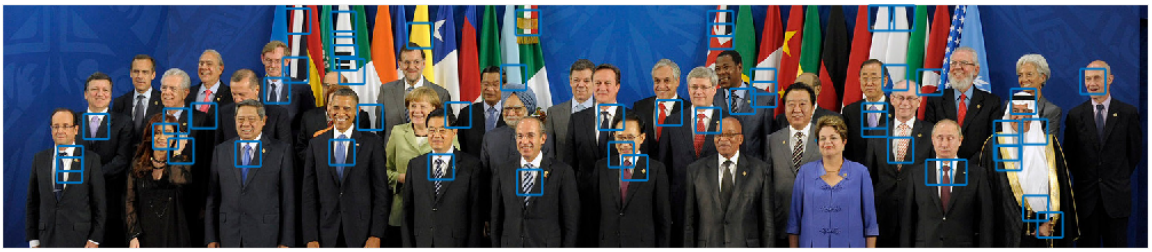

Figure 2.5: Template matching with normalized correlation result using fixed image

2.2 Req #2

This section will demonstrate how I build a classifier for face detection. The classifier is built based on AdaBoost which is introduced by Viola Jones[1].

2.2.1 Haar Feature Definition

A Haar Feature is a rectangle feature which is sensitive to some simple object like corner or line segment. Bellow shows how a Haar feature can be applied to face detection:

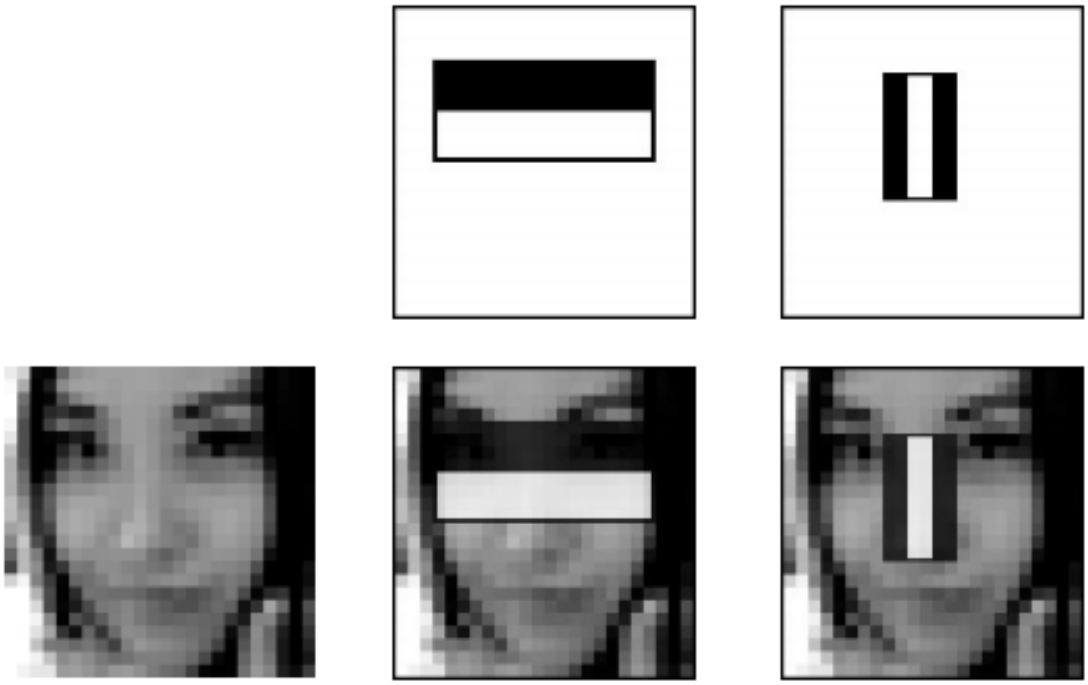

Figure 2.6: Visualization and matching of Haar feature of a 24x24 window

This section will define 5 different types of Haar feature:

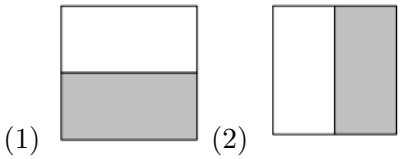

Figure 2.7: Edge feature

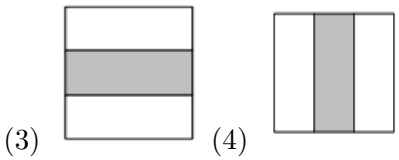

Figure 2.8: Linear feature

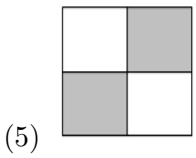

Figure 2.9: Direction feature

2.2.2 Feature Extraction

A feature can be extracted by applying a Haar feature to the image and calculate the pixel value difference between sum of of values under white area and sum of values under black area. To achieve fast computation of haar feature extraction, we can use Integral Image(ii(x,y)):

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} (i(x', y'))$$

Where i(x',y') is the point with corrdinate (x',y') of original image.

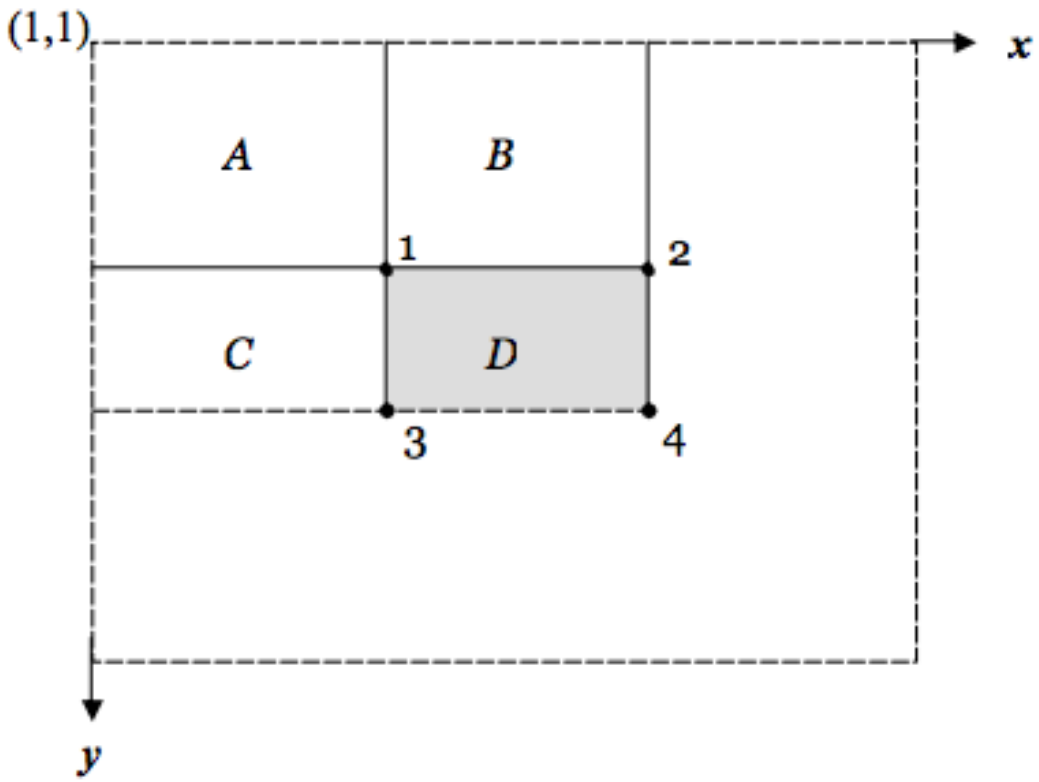

Figure 2.10: Area of D can be computed by $ii_4 + ii_1 - (ii_2 + ii_3)$

Here is the Matlab implementation of calculating Haar feature value based on Integreal Image calHaar:

```
function f1r = calHaar( image,feature )
%Calculate Haar feature value. The input Image matrix is "Integral␣
    Image".
I = image;
[nrow,ncol] = size(feature);
[rowII,colII] = size(image);
for i = 2:rowII-nrow+1
    for j = 2:colII-ncol+1
        [nrow,ncol] = size(feature);
        for ii = 1:nrow
            for jj = 1:ncol
                subtra(ii,jj) = I(i+(ii-1),j+(jj-1)) +...
                    I(i+(ii-1)-1,j+(jj-1)-1)-I(i+(ii-1)-1,...
                    j+(jj-1))-I(i+(ii-1),j+(jj-1)-1);
            end
        end
```

```
        result = dot(feature,subtra);
        f1r(i-1,j-1) =    sum(result);
    end
end
end
```

By scaling and moving across the template window, a 24 by 24 window can generate over 160,000 different features: edge feature: 86,400, linear feature: 55,200 and direction feature: 20,736. Such amount of number is practically impossible to use, thus we need to reduce the number of features.

2.2.3 AdaBoosting

AdaBoost[REF] is a machine learning technique used to train a strong classifier by combining several weak classifiers. In this report, a weak classifier can be considered as one Haar feature. The workflow of AdaBoost training consists of seven steps:

1. For a given training sample: (x_n, y_n), define $y_i = 1$ for positive sample and $y_i = 0$ for negative sample.
2. Apply weight initialization function:
   - for l positive samples:
   $$w_{1,i} = \frac{1}{2l}$$
   - for m negative samples:
   $$w_{1,i} = \frac{1}{2m}$$
3. To train Ts weak classifiers: $t = 1, ..., T$:
   - Weight normalization:
   $$q_{t,i} = \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
   - for each feature f, train a weak classifier $h(x, f, p, \theta)$ and compute its weighted error rate ε_f :
   $$\varepsilon_f = \sum_i q_i \left| h(x_i, f, p, \theta) - y_i \right|$$
   [details are provided at section 2.2.4]
   - Finding the best weak classifier$h_t(x)$:
   $$\varepsilon_t = min_{f,p,\theta} \sum_i q_i \left| h(x_i, f, p, \theta) - y_i \right|$$
   [details are provided at section 2.2.4]

- Assign Weight:

$$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$$

$$\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$$

where $e_i = 0$ represents x_i has been correctly labelled, $e_i = 0$ represents x_i has not been correctly labelled.

- The final strong classifier is:

$$C(x) = \begin{cases} 1, & \text{if } \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0, & \text{otherwise} \end{cases}$$

where $\alpha_t = log\frac{1}{\beta_t}$

2.2.4 Finding the best weak classifier

A weak classifier h:

$$|h(x_i, f, p, \theta)| = \begin{cases} 1, & \text{if } pf(x) < p\theta \\ 0, & \text{otherwise} \end{cases}$$

where f is a haar feature, θ is threshold and p stands for the sign of inequation.

Training a weak classifier means to determine the threshold value for that haar feature. After training session, compute the error rate for each classifier based on the threshold we just obtained. So the best weak classifiers can be found by locating the feature with lowest error rate.

For each feature f, sort their feature value. After the iteration over that sorted list, we can determine the threshold by computing the followings:

1. Sum of all face sample's weight T^+
2. Sum of all non-face sample's weight T^-
3. Sum of all face sample's weight before the current sampleS^+
4. Sum of all non-face sample's weight before the current sample S^-

The threshold would be the value between the current feature value F_{k_j} and its previous feature value $F_{k_{j-1}}$. We define the feature less than this threshold is face and feature larger than this threshold is non-face. And the error rate for each feature e can be computed by:

$$e = min(S^+ + (T^- - S^-), S - +(T^+ - S^+))$$

So after one iteration through the sorted list, we can find the best weak classifier. The Matlab implementation of finding the best weak classifier are written in `file: aWeakClassifier.m`

After T times iteration, we can construct those Ts weak classifiers into a strong classifier:

$$C(x) = \begin{cases} 1, & \text{if } \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0, & \text{otherwise} \end{cases}$$

with

$$\alpha_t = log\frac{1}{\beta_t} = log\frac{1-\varepsilon_t}{\varepsilon_t} = -log\varepsilon_t$$

Each weak classifier has its own weight α_t, when perform face detection, each classifier will start "voting". if more than a half number of weak classifiers gives result 1 then the output is 1 and vice versa.

2.2.5 Result

For instruction of training and running Req#2 classifiers please read Appendix[User Documentation].

In this report the number of face data for training is 840 and number of non-face data for training is 600. All training datas are normalized by histogram equalization. According to Viola Jones[1], a good strong classifier should consists 200 weak classifier. However training a strong classifier by AdaBoost is a very time-consuming task. By using 2 GHz Intel Core i7 CPU and 8 GB 1600 MHz DDR3, training a strong classifier with T=1(only one weak classifier) takes time:

```
>> vj
[ Reading source image ]
[ Extracting features ]
Elapsed time is 364.620006 seconds.
```

Figure 2.11: Elapsed time for training a Strong Classifier with T=1

With limited amount of time,I have trained two different sized classifiers $C1$ and $C2$. C1 is a 50 sized strong classifier which takes 5 hours for training and C2 is a 100 sized classifier which takes 12 hours for training.

For face detection part, a moving window is moving across the g20 image. For each moment, the current window will be resized to [24 24] size. After that, applying the strong classifier to the window.

Bellow gives the result of running C1:

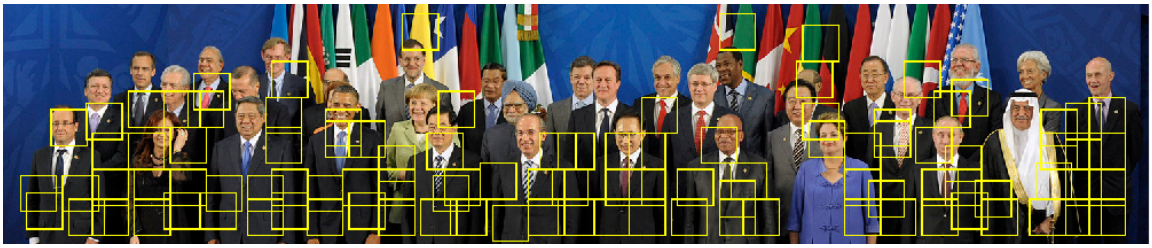

Figure 2.12: Result from running C1

The result of running C2:

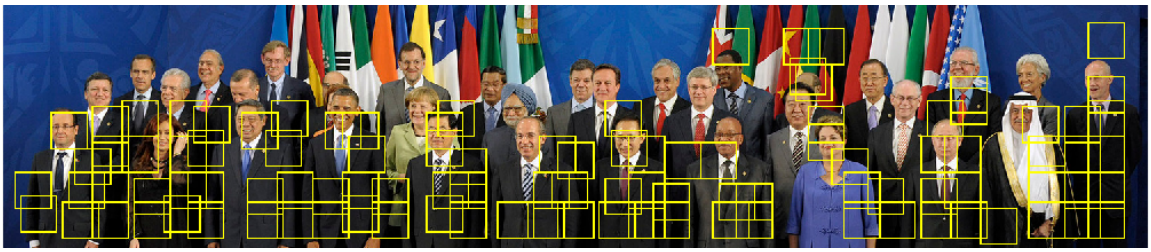

Figure 2.13: Result from running C2

The above results show that the classifier has a pretty high false positive rate C1: 100/102 and C2: 97/104. However, strong classifier with more weak classifiers does improve the accuracy(more true positive, less false positive).

2.2.6 Conclusion

Classifier with small size does not perform well, to further improve those classifier, a good way is to train strong classifier with large capacity(e.g T = 200). By reviewing C2:

	1	2
1	2.6800	[99,10.1059,0.0642,4,6,6,6,19]
2	1.0907	[110,27.5019,0.2515,4,6,10,7,15]
3	1.1961	[115,65.7745,0.2322,4,7,24,4,1]
4	0.5747	[104,15.1882,0.3602,4,6,8,7,17]
5	1.1371	[108,12.9392,0.2429,4,6,7,7,18]
6	1.4478	[109,16.0804,0.1903,4,5,9,8,16]
7	2.3901	[110,10.2569,0.0839,4,7,5,4,19]
8	1.1244	[111,81.0157,0.2452,3,24,8,1,1]
9	2.8184	[111,17.9706,0.0563,4,6,9,7,16]
10	6.8338	[104,13.4275,0.0011,4,6,7,6,18]
11	4.3164	[112,6.6824,0.0132,4,7,4,4,20]
12	4.6704	[110,8.0961,0.0093,4,6,5,6,20]
13	1.5701	[113,78.5951,0.1722,3,22,8,1,1]
14	10.8432	[112,6.4725,1.9537e-05,4,6,4,6,20]
15	4.0106	[114,77.0647,0.0178,3,22,8,3,1]
16	4.1990	[113,76.0922,0.0148,3,23,8,1,1]
17	8.3359	[115,13.0510,2.3970e-04,4,5,8,8,17]
18	12.3149	[115,79.1443,4.4843e-06,3,23,8,2,1]
19	13.0570	[114,7.2627,2.1352e-06,4,6,4,5,20]
20	32.1723	[116,9.4002,1.0660e-14,4,5,6,8,19]
21	32.1489	[116,6.5863,1.0913e-14,4,6,4,6,21]
22	24.8601	[118,75.1824,1.5973e-11,3,21,8,4,1]
23	18.0216	[119,10.7176,1.4904e-08,4,5,7,8,18]
24	18.0802	[119,15.1059,1.4056e-08,4,6,8,6,17]
25	8.4188	[120,73.1206,2.2064e-04,3,21,8,2,1]
26	37.9891	[120,78.2955,3.1736e-17,3,22,8,2,1]
27	89.1745	[120,5.1533,1.8707e-39,4,6,3,6,21]

Figure 2.14: First 27 weak classifiers from C2

The first element of second column vector represents number of mis-labelling. Most of mis-labelling come from facedata.png. So the classier is not fully trained to tackle faces from g20. The training data is not suited in this coursework. Trying different training data might be another way to improve the classifier.

2.3 Req#3

This section will give a solution of how to recognize face by using Nearest-Neighbour Classifier.

The Nearest-Neighbour(NN) classifier is based on the Hypothesis that each clusters are well separate. So in the feature space, there should exist k different number of clusters and each cluster represent only one class. For a new observation, work out the distance between its feature and other clusters, and that new observation should belong to the nearest cluster.

The feature used for this section is Pixel Value. So we can compute Euclidean distance by:

```
euDistance =[];
for t = 1:nRects
    for i = 1:nPeople
        for j = 1:nExamples
            %faceCell{1,1}(:)represents first sample from class 1.
            diff = test_data(:,t)- faceCell{i,j}(:);
            distance = diff.'*diff; %this equals squared Euclidean
                Distance.
        euDistance(i,j,t) = distance;
        end
    end
end
```

And perform classification:

```
% Classify test data
test_pred = zeros(nRects, 1);
for i = 1:nRects
    [m n] = min(avgDis(:,i));
    test_pred(i) = n;
end;
```

2.3.1 Result

Running the NN classifier produces the following result:

Figure 2.15: Req#3 output

with accuracy 5/18.

2.4 Req#4

For this section, I tried different features and different classification algorithm to perform face recognition. I will start with introducing HOG feature and GMM.

2.4.1 HOG with GMM

HOG stands for histogram of oriented gradients. It is a feature descriptor which measures the gradient orientation or so-called edge direction for a given image. In theory, images of people with different facial appearance should differ in their HOG value. Bellow gives the image and its HOG visualization, the figure on the right is the HOG plotted on that image.

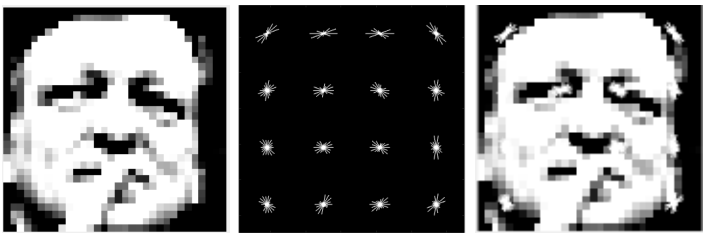

Figure 2.16: Image with its HOG

As can be seen, the HOG can describe human face appearance.

Building the GMM classifier rely on the hypothesis that the HOG distribution follows the Gaussian Distribution. So each class can be represented by Multivariate Gaussian Distribution:

$$p(x|\theta_i) \triangleq \frac{1}{(2\pi)^{K/2}|C_i|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_i)^T C_i^{-1}(x-\mu_i)\right)$$

Where θ stands for class label, K stands for feature dimensionality(324 for HOG), C stands for covariance matrix and μ stands for mean.

So we can fit the GMM:

$$p(x) = \sum_{i=1}^{n} \alpha_i N(x|\mu_i, C_i)$$

by calculating covariance and mean for each cluster: For mean μ

```
%% Calculate N dimension mean for each class. N = 324
for i = 1:20
    for j = 1:324
        hogMeanValue(i,j) = mean(hogValue(:,j,i));
    end
end
```

For covariance:

```
%% Calculate N dimension covariance for each class.
for i = 1:20

    hogCovValue(:,:,i) = cov(hogValue(:,:,i));

end
```

The weight α for each cluster is equally weighted.

The classification stage is based on maximum a posteriori probability (MAP) estimate. A posteriori probability can be expressed by:

$$p(\theta|x) = \frac{(x|\theta)p(\theta)}{p(x)}$$

So, by finding

$$\underset{\theta}{\operatorname{argmax}}\, p(\theta|x) \tag{2.1}$$

We can obtain the most possible label θ for datum x. In this coursework, $p(x)$ is constant, so just compute:

$$\arg\max_{\theta} p(\theta|\mathbf{x}) = \arg\max_{\theta} \log\left(p(\mathbf{x}|\theta)p(\theta)\right)$$
$$= \arg\max_{\theta} \left[\log\left(p(\mathbf{x}|\theta)\right) + \log\left(p(\theta)\right)\right]$$

I use log to avoid under flow error. By substituting the above equation into the Multivariate Gaussian distribution equation:

$$\underset{\theta}{\text{argmax}}\, p(\theta|x) = \underset{\theta}{\text{argmax}}[log(p(\theta)) - 0.5log(|C_\theta|) - 0.5(x-\mu_\theta)^T C_\theta^{-1}(x-\mu_\theta)]$$

Implementation in Matlab:

```
function classification =
    classifyFace(featureHog,hogMeanValue,hogCovValue)
% This function is used to calculate p(theta|x)
currentMax = -inf;
for i = 1:20
    currentMean = hogMeanValue(i,:);
    currentCov = hogCovValue(:,:,i);
    prior = 1/20;
    [u,d]=eig(currentCov);
    argMax = log(prior) - 0.5*sum(real(log(diag(d)))) ...
        -0.5*(featureHog-currentMean)*inv(currentCov)*...
        transpose(featureHog-currentMean);
    if argMax > currentMax
        currentMax = argMax;
        classification = i;

    end

end
end
```

2.4.2 SVM

A support vector machine is a supervised learning method. In feature space, a SVM is used to find the hyperplane which separate classes with largest margin.

If classes are linear separable, then I will use linear SVM, if not, I will try different kernel functions for the non-linear classification.

In this section, I will use two different features to train SVM: the pixel values and HOG.

2.4.3 Result

By running HOG with GMM:

Figure 2.17: Result from HOG with GMM classifier

Which gives 1/18 accuracy.

The result from SVM trained by Pixel values:

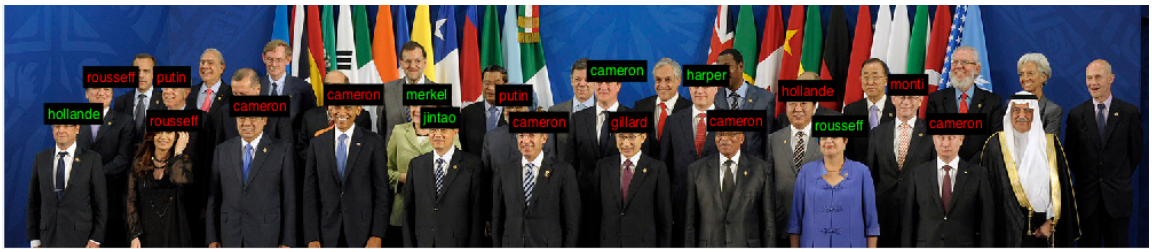

Figure 2.18: Result from SVM, feature: pixel value

The above result produces 6/18 accuracy.

The result from SVM trained by HOG:

Figure 2.19: Result from SVM, feature: pixel value

This gives 7/18 accuracy.

The above SVM result are gathered from running SVM on polynomial kernel function:

$$(\gamma * u' * v + coef0)^d$$

The following result is based on linear kernel:

Figure 2.20: Result from SVM, feature: pixel value,kernel:linear

With accuracy 8/18.

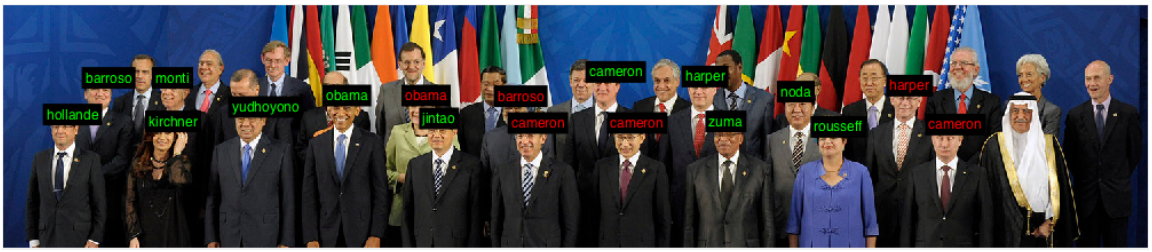

Figure 2.21: Result from SVM, feature: HOG, kernel:linear

With accuracy 12/18.

2.4.4 Conclusion

The result of HOG with GMM classifier does not perform well, all faces are labelled with Calderon. Which means the HOG distribution does not follow the Gaussian Distribution. The result from HOG with SVM by using linear kernel may support this idea.

By running SVM on two features: HOG and Pixel Value, we can observe that HOG in face detection perform better than pixel value. Using linear kernel cause improvement on both SVM could conclude that these features are linearly separable.

Bibliography

[1] Paul Viola and Michael J. Jones. *Robust Real-Time Face Detection*. International Journal of Computer Vision, 2004.

Appendix A

User Documentation

Req#1: Run script FaceDetect.m

Req#2:

- Calculate Training Data: put face file in : data face and put non-face file in : data nonface
- Run image_ini : this will create two cells:haarFeatureVolume and haarFeatureNonfaceVolume
- AdaBoosting: Run script vj.m to train a strong classifier this would produce alphaSet
- Classification: Run script classificationFace.m this will produce classification(i,j) matrix
- Draw Rectangle: Run script convergeDetections.m this will show the detection result

I have provide two classifiers C1 and C2 with name alphaSet.mat and 100alphaSet.mat, load these file and do the classification.

Req#3: Run script FaceDetect.m

Req#4:

- Run REQ4.m: HOG + GMM
- Run REQ4ver2.m: Pixel + SVM. Change kernel: change 3rd parameter for svmtrain (e.g '-t 0' for linear)
- Run REQ4ver3.m: HOG + SVM. Change kernel: change 3rd parameter for svmtrain