CSE 473/573 - Computer Vision and Image Processing

Project #3: Face Detection in the Wild

JingJing Zhuang
UBIT: jzhuang3
50290190

Ruiying Liu
UBIT: rliu25
50247993

Work Distribution:

We both done research on Viola-Jones algorithm, and use the algorithm to implement our project. Basically, Ruiying is in charge of training dataset and define features. We both worked on training classifiers. And at the end, JingJing is in charge of testing the trained classifiers into face detection and Ruiying helps analyzing the results.

Viola-Jones algorithm overview:

Viola-Jones algorithm is a widely used algorithm that can be used in real-time face detection. Although the training for this algorithm takes time, it is very efficient when used in face detection.

- Detection Target
  - All the images in [data-directory]

- Training Dataset
  - 5,171 faces in a set of 2,845 images (FDDB)
  - 712 background image

- Define Base Features
  - Feature2h = $[1 \quad -1]$
  - Feature2v = $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$
  - Feature3h = $[-1 \quad 1 \quad -1]$
  - Feature3v = $\begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$
  - Feature4 = $\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$

- Training Classifier
  - Adaptive Boosting: An efficient feature selector
    - Integral images
    - Feature selection
    - Building fast classifiers: Combines multiple weak classifiers into a strong classifier.
    - Cascaded classifier

- Test the classifiers on target images and get the results

Description of project implementation

- **Scripts for implementation**

1. unzip Project3.zip
2. cd Project3
3. python FaceDetector.py [data-directory]

- **Preprocessing of the training dataset:**

1. Load the dataset from file, the dataset contains 5,171 faces in a set of 2,845 images from FDDB and 712 background image
   We random choose 1000 faces and different size background images.

```
p=1000;n=1000
face_data=sample_data(face_grey,p)
back_data=sample_data(back_grey,n)
```

2. Convert the dataset into greyscale and resize them into 24x24 dimension.
3. Get a label for each of the data, if data is face then label = 1, if data is background then label = -1.
4. Normalize all the processed dataset images(0-1), then get the integral images from normalized images.
   - Integral image $I = \sum_{x=0}^{i} \sum_{y=0}^{j} I_{i,j}$

```
def to_integral(img,x,y):
    result = np.zeros((x, y), dtype=np.float32)
    result = cv2.normalize(img, dst=result, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX)
    sum_ = np.zeros((x+ 1, y + 1), dtype=np.float32)
    imageIntegral = cv2.integral(result, sum_, cv2.CV_32FC1)
    return imageIntegral
```

- **Get the training feature value:**

1. Define base features
   - Feature2h = $[1 \quad -1]$
   - Feature2v = $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$
   - Feature3h = $[-1 \quad 1 \quad -1]$
   - Feature3v = $\begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$
   - Feature4 = $\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$

   Define possible shape based on the window size=24

```
def shapes(width, height, WINDOW_SIZE):
    shape = []
    for w in range(width, WINDOW_SIZE + 1, width):
        for h in range(height, WINDOW_SIZE + 1, height):
            shape.append([w, h])
    return shape
```

Define possible x, y position based on the shape

```
def position(WINDOW_SIZE, width, hight):
    pos = []
    for x in range(WINDOW_SIZE - width + 1):
        for y in range(WINDOW_SIZE - hight + 1):
            pos.append([x, y])
    return pos
```

The value of each feature can get use the positon and shape.

Take feature2h as example:

```
def feature2h(image, x, y, width, height):
    h_w = width // 2
    position = []
    left_value = image[x + h_w, y + height] - image[x, y + height] - image[x + h_w, y] + image[x, y]
    right_value = image[x + width, y + height] - image[x + h_w, y + height] - image[x + width, y] + image[x + h_w, y]
    position.append(x), position.append(y), position.append(width), position.append(height)
    return position, left_value - right_value
```

Finally, we can get total 162336 number of feature:

Feature2h: 43200

Feature2v: 43200

Feature3h: 27600

Feature3v: 27600

Feature4: 20736

2. Get the feature values by applying each trained data into five base features.

```
for i in range(train_data.shape[0]):
    fea_2h, value_2h = getfea2h(train_data[i])
    fea_2v, value_2v = getfea2v(train_data[i])
    fea_3h, value_3h = getfea3h(train_data[i])
    fea_3v, value_3v = getfea3v(train_data[i])
    fea_4, value_4 = getfea4(train_data[i])
    value.append(value_2h + value_2v + value_3h + value_3v + value_4)
    if i%100==0:
        print(i)
```

- **Adaptive Boosting**

  For training data: $\{\{x_1, y_1\}, \{x_2, y_2\}, \{x_n, y_n\}\}$

1. Initial the weight matrix: $D_1 = (w_{1,1}, w_{1,2}, .. w_{1,n})$, $w_{1,i} = \frac{1}{n}$, $\mathrm{i} = 1,2, \ldots \mathrm{n}$
2. For i=1, 2 …. M(classifierNumberMax)
   I. For j=1,2…… sample number

      Find best feature, threshold and polarity which can get smallest error.

      ```
      best_class, error, pre_label = findbestfeature(dataset, labelset, D, step)
      ```

   II. Use the error get weight of weak classifier G(i):

      $a_i = 0.5 * log((1 - error)/error)$

   III. Get new weight matrix for next loop:

```
expon = np.multiply(-1 * alpha * labelset, pre_label)
expon = np.exp(expon)
D = np.multiply(D, expon)
Z = np.sum(D)
D = D / Z
```

.

3. Get final strong classifier
   $S = \sum_1^m a_i G(i)$

- **Preprocessing testing dataset:**

1. Load the testing images from data directory
2. Convert the testing images to grayscale, then get the integral data from the grayscale images.

- **Cascaded classifier:**

1. Combines multiple weak classifiers into a strong classifier
2. First cascade contains 2 weak classifier
3. $S(x) = \begin{cases} 1 & if \ \sum_{i=1}^{T} a_i w_i(x) > 0 \\ -1 & otherwise \end{cases}$
4. If strong classifier > 1, then execute the second cascade stage.
5. Second cascade contains 10 weak classifiers, if strong classifier is positive, then go to the third stage.
6. Run third cascade stage which contains 25 weak classifier.
7. Cascade classifier returns a coordinate list which contains the face x and y location.

- **NMS to eliminate overlap window**

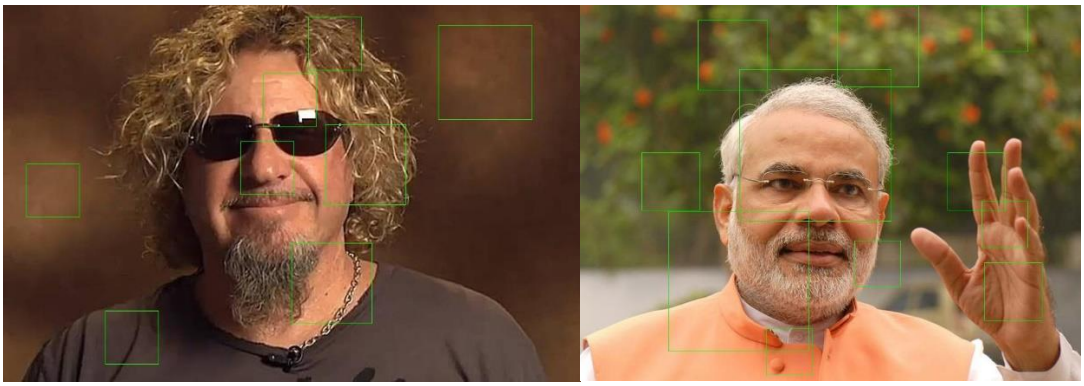
1. Sort the all score
2. Find the window i which has largest score
   I. For j in other window
   II. Get the overlap area between the two window
   III. $\text{iuo} = \frac{overlap}{area(i)+area(j)-overlap}$
   IV. If iuo larger than threshold, get rid of this window
   V. Continue until no windows
3. Store face position in json list and write into the result.json file.

Results of Face Detector on FDDB

1. Training data: 600 face, 500 background
2. Number of weak classifier: 36. Use three levels.
3. Nms threshold: 0.3
4. Training time: 10 hours

Good result:

Bad result:



Results analysis:

- During the train step, we set a stop sign once the final predict error is zero. This method may shorter the training time, but will sacrifice the performance.
- Due to some bad result, we have many methods to solve this:
  1. Use more training data.
  2. Train more weak classifiers.
  3. Use more cascade classifier.
  4. Set better threshold in the Ada boosting and NMS method.