BAB 12

DETEKSI DANPENGENALAN WAJAH

Bab ini membahas dua topik yang berhubungan dengan wajah orang, yakni:

- cara mendeteksi wajah;
- cara mendeteksi mata;
- pemakaian klasifikasi LBP;
- cara mengidentifikasi wajah seseorang.

12.1 Deteksi Wajah dengan Pengklasifikasi Haar

Deteksi wajah adalah proses untuk mengenali wajah orang dalam suatu citra digital. Gambar 12.1 menunjukkan contoh deteksi wajah pada suatu citra. Tampak bahwa wajah-wajah orang yang dikenali ditandai dengan kotak.

Gambar 1!2.1 Deteksi wajah dapat mengenali lebih dari satu wajah

Untuk mendeteksi wajah, OpenCV menyediakan pengklasifikasi Haar dan pengklasifikasi LBP.

Contoh penggunaan pengklasifikasi Haar dapat dilihat pada skrip

Berkas : **deteksiwajah.py**

```
 # Pendeteksi wajah

import numpy as np
import cv2

pengklasifikasiWajah = cv2.CascadeClassifier(
    "haarcascade_frontalface_default.xml")
```

```
citra = cv2.imread("lena.png")

if citra is None:
    print("Tidak dapat membaca berkas citra")
    exit()

abuAbu = cv2.cvtColor(citra, cv2.COLOR_BGR2GRAY)
dafWajah = pengklasifikasiWajah.detectMultiScale(
    abuAbu, scaleFactor = 1.3, minNeighbors = 2)

print("Jumlah wajah terdeteksi:", len(dafWajah))

for (x, y, w ,h) in dafWajah:
    cv2.rectangle(citra, (x, y), (x + w, y + h),
                  (255, 0, 0), 2)

cv2.imshow('Citra wajah', citra)
```

Akhir berkas

Sebelum menjalankan skrip ini, berkas haarcascade_frontalface_default.xml **yang berada di folder** opencv\sources\data\haarcascades **perlu disalin ke folder tempat skrip berada.**

Gambar 12.2 menunjukkan hasil skrip ini. Tampak bahwa wajah yang terdeteksi diberi kotak.

Gambar 12.12 Wajah diheri kotak

Sekarang, marilah untuk memahami kode pada deteksiwajah.png. **Pertama-tama, pengklasifikasi wajah dalam bentuk XML perlu dimuat. Perintahnya berupa:**

```
pengklasifikasiWajah = cv2.CascadeClassifier(
    "haarcascade_frontalface_default.xml")
```

Pengklasifikasi yang digunakan adalah pengklasifikasi Haar, yang pertama kali diperkenalkan oleh Paul Viola and Michael Jones. Penjelasan lebih lanjut tentang pengklasifikasi ini dapat dibaca pada:

https://docs.opencv.org/3.4.3/d7/d8b/tutorial_py_face_detection.html

Setelah citra dibaca (yakni lena.png), **perlu dikonversi ke citra berskala keabu-abuan. Perintahnya berupa:**

```
abuAbu = cv2.cvtColor(citra, cv2.COLOR_BGR2GRAY)
```

Hal ini diperlukan mengingat pengklasifikasi wajah memerlukan berkas dalam bentuk aras keabu-abuan.

Kemudian, deteksi wajah dilakukan melalui:

```
dafWajah = pengklasifikasiWajah.detectMultiScale(
    abuAbu, scaleFactor = 1.3, minNeighbors = 2)
```

Dalam hal ini, dafWajah **akan berisi senarai kotak yang berisi wajah yang didapat.**

Metode detectMultiScale() **melibatkan beberapa argumen. Pada contoh ini, argumen pertama berupa citra dalam bentuk aras keabu-abuan. Argumen** factorScale **menyatakan faktor skala yang digunakan untuk melakukan kompensasi terhadap wajah yang dekat dengan kamera dan yang jauh dari kamera. Nilai yang rendah membuat penyekalaan mengecil secara perlahan-lahan dan efeknya dapat menangkap banyak wajah atau bahkan ada yang bukan wajah pun dimasukkan. Nilainya harus lebih besar daripada 1. Argumen**

minNeighbors menentukan jumlah objek minimum yang bisa dideteksi di dekat wajah yang sekarang didapatkan. Parameter ini sangat bermanfaat untuk mengurangi deteksi positif palsu.

Terkadang argumen minSize dan maxSize disertakan pula. Argumen minSize menentukan ukuran kotak wajah. Secara bawaan, ukuran kotak adalah (30, 30) yang berarti 30 x 30 piksel. Itulah sebabnya, jika objek wajah pada citra berukuran lebih kecil daripada itu, argumen ini perlu disertakan. Argumen maxSize menentukan ukuran terbesar kotak wajah. Kotak yang lebih besar daripada yang ditentukan pada maxSize akan diabaikan.

Pernyataan berikut digunakan untuk menginformasikan jumlah wajah yang terdeteksi:

```
print("Jumlah wajah terdeteksi:", len(dafWajah))
```

Pernyataan berikut digunakan untuk menggambar kotak pada setiap wajah yang didapat:

```
for (x, y, w ,h) in dafWajah:
    cv2.rectangle(citra, (x, y),  (x + w, y + h),
                  (255, 0, 0),      2)
```

Dalam hal ini, (x, y) menyatakan koordinat pojok kiri-atas kotak, w menyatakan lebar kotak, dan h menyatakan tinggi kotak. Kotak digambar dengan warna biru, yang ditentukan oleh (255, 0, 0) dan dengan ketebalan 2 piksel.

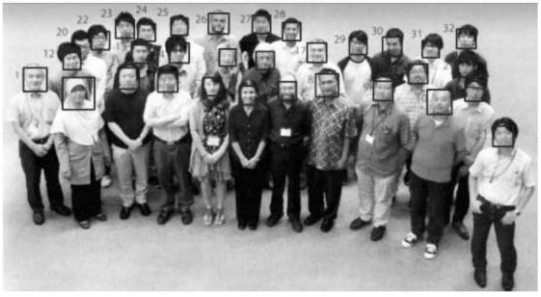
Skrip deteksiwajah.py dapat digunakan untuk mendeteksi banyak wajah. Untuk melihat efeknya, citra lena.png dapat diganti dengan foto apa saja yang mengandung banyak wajah orang.

Deteksi wajah dengan pengklasifikasi Haar tidak selalu memberikan hasil yang sempurna. Terkadang, terdapat wajah orang yang tidak terdeteksi. Adakalanya, objek bukan wajah orang dianggap sebagai wajah orang. Hal ini bisa disebabkan wajah terlalu kecil atau terlalu

maju. Penyebab lain, ada objek yang mempunyai ciri-ciri yang membuat pengklasifikasi menganggapnya sebagai wajah. Oleh karena itu, beberapa pengaturan melalui factorScale, minNeighbors, dan minSize perlu dilakukan. Gambar 12.3 hingga 12.5 menunjukkan pengaturan nilai scaleFactor dan minNeighbors yang memberikan efek yang berbeda.

Gambar Jg,s Hasil dengan scaleFactor = J.S, minNeighbors = 5

Gambar Jg,4 Hasil dengan scaleFactor = 1.S, minNeighbors = g



Gambar Jg,5 Hasil dengan scaleFactor = 1.05, minNeighbors = 5

12.2 Deteksi Mata

OpenCV juga menyediakan pengklasifikasi yang membuat mata pada wajah dapat dideteksi. Untuk keperluan ini, berkas haarcascade_eye.xml **yang berada di folder** opencv\sources\data\haarcascades **perlu disalin ke folder tempat skrip berada.**

Selanjutnya, skrip berikut bisa dicoba:

Berkas : deteksimata.py

```
# Deteksi mata

import numpy as np
import cv2

pengklasifikasiWajah = cv2.CascadeClassifier(
    "haarcascade_frontalface_default.xml")
pengklasifikasiMata = cv2.CascadeClass ifier(
    "haarcascade_eye.xml")

citra = cv2.imread(" lena.png")

if citra is None:
    print("Tidak dapat membaca berkas citra")
    exit()

abuAbu = cv2.cvtColor(citra, cv2.COLOR_BGR2GRAY)
dafWajah = pengklasifikasiWajah.detectMultiScale(
    abuAbu, scaleFactor =1.3, minNeighbors =2)
for (x, y, w ,h) in dafWajah:
    cv2.rectangle(citra, (x, y), (x + w, y + h),
                  (255, 0, 0), 2)

    roiAbuAbu = abuAbu[y : y +  h, x : x + w]
    roiWarna = citra[y: y + h,  x : x + w]
    daftarMata = pengklasifikasiMata.detectMultiScale(
                     roiAbuAbu, 1.3, 2)
    for (mx, my, mw, mh) in daftarMata:
        cv2.rectangle(roiWarna, (mx, my),
                      (mx + mw, my+  mh),
                      (0, 255, 0), 2)
```

```
cv2.imshow("Citra wajah", citra)
waitKey(0)

Akhir berkas
```

Skrip ini merupakan hasil pengembangan terhadap skrip deteksimata. py.**Tambahan pertama berupa:**

```
roiAbuAbu = abuAbu[y: y + h, x : x + w]
roiWarna = citra[y: y + h, x : x + w]
```

Pernyataan pertama menentukan area pada kotak wajah untuk citra berskala keabu-abuan dan pernyataan kedua menentukan area pada kotak wajah untuk citra berwarna.

Selanjutnya,

```
daftarMata   pengklasifikasiMata.detectMultiScale(
                 roiAbuAbu, 1.3, 2)
```

merupakan langkah untuk mendapatkan daftar mata. Secara prinsip, perintah ini sama dengan pada pendeteksi wajah. Akan tetapi, citra yang diproses adalah bagian citra yang berisi wajah (bukan citra keseluruhan). Hasilnya digambar melalui:

```
for (mx, my, mw, mh) in daftarMata:
    cv2.rectangle(roiWarna, (mx, my),
                  (mx + mw, my+  mh),
                  (0,  255, 0), 2)
```

Dalam hal ini, warna yang digunakan untuk kotak berupa hijau dengan ketebalan 2 piksel.

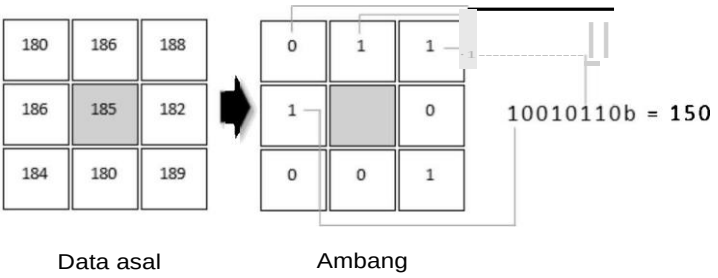
Hasil skrip deteksimata .py ditunjukkan pada Gambar 12.6.

Gambar Jg.6 Hasil identifikasi mata

12.3 Pengklasifikasi LBP

Contoh di depan memberikan gambaran tentang penggunaan pengklasifikasi Haar pada deteksi wajah. Nah, sekarang penggunaan pengklasifikasi LBP dibahas.

Local Binary Pattern (LBP) biasa digunakan pada berbagai aplikasi, misalnya untuk segmentasi dan klasifikasi tekstur, pencarian citra, dan bahkan untuk deteksi wajah. Perhitungan LBP ditunjukkan pada Gambar 12.6. Pada jendela berukuran 3 x 3, piksel yang berada di tengah berkedudukan sebagai nilai ambang. Kemudian, delapan tetangga diubah menjadi O atau 1 berdasarkan nilai ambang ini. Nilai yang lebih besar atau sama dengan ambang diberi nilai 1 dan untuk keadaan sebaliknya diberi nilai 0.



Gambar 1.6 Contoh perhitungan LBP

Baik LBP maupun Haar mempunyai keistimewaan tersendiri. Tabel 12.1 mencantumkan kelebihan dan kekurangan kedua metode tersebut didasarkan pada tulisan yang terdapat pada situs:

https://github.com/informramiz/Face-Detection-OpenCV

Tabel 1 - 1 Perbedaan penggunaan metode Haar dan LBP pada deteksi wajah

Algoritma	Kelebihan	Kekurangan
Haar	1. Akurasi deteksi tinggi 2. Tingkat positif palsu rendah	1. Komputasi kompleks dan pelan 2. Waktu pelatihan lebih lama 3. Kurang akurat pada wajah gelap 4. Ada keterbatasan pada kondisi cahaya gelap 5. Kurang handal terhadap oklusi (wajah yang tersembunyi di belakang wajah lain)
LBP	1. Komputasi sederhana dan cepat 2. Waktu pelatihan lebih singkat 3. Handal terhadap perubahan iluminasi lokal 4. Handal terhadap oklusi	1. Kurang akurat 2. Tingkat positif palsu tinggi

m Berkas : deteksimutu.llY **LBP untuk mendeteksi wajah:**

```
# Pendeteksi wajah menggunakan LBP

import numpy as np
import cv2

pengklasifikasiWajah = cv2.CascadeClassifier(
    "lbpcascade frontalface.xml")

citra = cv2.imread("lena.png")

if citra is None:
    print("Tidak dapat membaca berkas citra")
    exit()

abuAbu = cv2.cvtColor(citra, cv2.COLOR_BGR2GRAY)
dafWajah = pengklasifikasiWajah.detectMultiScale(
    abuAbu, scaleFactor = 1.3, minNeighbors = 1)

for (x, y, w ,h) in dafWajah:
    cv2.rectangle(citra, (x, y), (x + w, y + h),
                  (255, 0, 0), 2)

cv2.imshow("Citra wajah", citra)
cv2.waitKey(0)
```

Akhir berkas

Sebelum menjalankan skrip ini, berkas lbpcascade frontalface.xml **yang berada di folder** opencv\sources\data\lbpcascades **perlu disalin ke folder ternpat skrip berada.**

Pada hasil pengujian, nilai minNeighboars = 2 **membuat wajah Lena tidak dapat dideteksi. Itulah sebabnya, pada skrip nilai 1 digunakan.**

12.4 Pengenalan Wajah

Pengenalan wajah *(face recognition)* adalah proses untuk mengenali wajah sehingga pemiliknya bisa diketahui. Jika deteksi wajah hanya terbatas untuk mendapatkan wajah pada citra, pengenalan wajah berusaha lebih dalam untuk mengenali pemilik wajah tersebut.

Untuk membuat komputer mampu mengenali seseorang melalui wajahnya, terdapat tiga langkah yang perlu dilakukan.

1. Pengumpulan data. Data wajah semua orang yang ingin dikenali harus dikumpulkan terlebih dahulu.
2. Pelatihan kepada pengenal wajah. Data yang diperoleh pada langkah pertama perlu diberikan ke pengenal wajah. Dalam hal ini, setiap wajah perlu diberi label pemiliknya. Hasilnya, pengenal wajah akan mengingat semua data tersebut.
3. Pengujian kepada pengenal wajah. Hal ini dilakukan dengan memberikan data suatu wajah dan pengenal wajah berusaha untuk mengidentifikasinya.

Ketiga langkah tersebut akan dibahas satu per satu. Sebelum menuju ke sana, ada beberapa dasar yang perlu diketahui dibahas terlebih dahulu.

12.4.1 Pengenal Wajah pada OpenCV

Untuk keperluan mengenali wajah, OpenCV menyediakan tiga pengenal wajah, yaitu:

- EigenFace, berupa:

  ```
  cv2. EigenFaceRecognizer_create()
  ```

- FisherFace, berupa:

  ```
  cv2.EigenFaceRecognizer_create()
  ```

- Local *Binary Pattern Histogram* (LBPH), berupa:

  `cv2.LBPHFaceRecognizer_create()`

Perbedaan ketiga pengenal wajah ini dapat dibaca di:

https://thecodacus.com/face-recognition-opencv-train-recognizer /

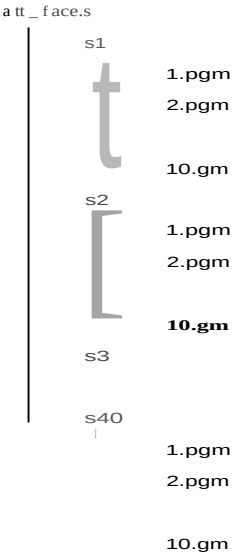
12.4.2 Pengumpulan Data

Untuk mengumpulkan data, Anda bisa mengumpulkan sendiri foto orang-orang yang ingin dikenali. Proses ini mungkin membutuhkan waktu yang cukup lama. Supaya jauh lebih cepat, untuk keperluan percobaan, Anda bisa menggunakan *dataset* yang tersedia di Internet. Salah satu *dataset* yang bisa dipakai adalah *AT&T facedatabase,* yang
bisa diunduh melalui:

http://www.cl.cam.ac.uk/Resea rch/DTG/atta rchive/pub/data/att_faces.zip

Basis data wajah ini mengandung 40 orang dengan masing-masing diwakili oleh 10 citra.

Jika berkas `att_faces.zip` diekstraksi, akan diperoleh susunan subfolder dan berkas seperti terlihat pada Gambar 12.7.



Gambar Jg. 7 Hierarki berkas pada att_laces

Untuk keperluan pemrosesan skrip di belakang, folder att faces perlu diletakkan di C: \LatOpenCV. Gantilah nama att faces menjadi data-wajah.

Kemudian, hapuslah berkas README.

12.4.3 Pelatihan ke Pengenal Wajah

Tahap berikutnya adalah mengenalkan gambar-gambar yang terdapat pada folder da ta-waj ah ke pengenal wajah. Dalam hal ini, pengenal wajah yang digunakan adalah LBPH.

ng diperlukan seperti berikut:

Berkas : **det ksimiil •-l'Y**

```
# Pelatihan ke pengenal wajah

import numpy as np
import os
```

```
import cv2

pengenalWajah = cv2.face.LBPHFaceRecognizer_create()
detektor = cv2.CascadeClassifier(
              "haarcascade frontalface default.xml")

def perolehCitraDanLabel(lintasan):
    #peroleh semua subfolder di lintasan
    daftarFolderCitra = [os.path.join(lintasan, f) \
        for fin  os.listdir(lintasan)]

    #Mula-mula, daftarSampelWajah dan daftaridWajah
    #    berupa senarai kosong
    daftarSampelWajah = []
    daftaridWajah= []

    #Proses semua berkas di subfolder
    for folderCitra in daftarFolderCitra:
        daftarBerkas = os.listdir(folderCitra)
        for berkas in daftarBerkas:
            berkasCitra = folderCitra + "\\" + berkas
            print("Pemrosesan berkas citra", berkasCitra)

            #Baca berkas citra. mode skala keabu-abuan
            citra = cv2.imread(berkasCitra, 0)

            # Ambil angka saja (buang s)
            idWajah   os.path.basename(folderCitra)[1:]
            idWajah = int(idWajah)

            #Ambil  wajah
            daftarWajah = detektor.detectMultiScale(
                              citra)

            #Simpan wajah dan ID ke senarai
            for (x, y, w, h) in daftarWajah:
                daftarSampelWajah.append(
                    citra[y: y + h, x : x + w])
                daftaridWajah.append(idWajah)

    return daftarSampelWajah, daftaridWajah

#Proses pelatihan
daftarWajah, daftaridWajah = perolehCitraDanLabel(
                               "data-wajah")
pengenalWajah.train(daftarWajah, np.array(daftaridWajah))
    enalWajah.save("pelatihan.yml")

    Akhir berkas
```

Hasil skrip ini seperti berikut:

```
C:\LatOpenCV>python pe1atihan.py<P

Pemrosesan berkas citra data-wajah\sl\1.pgm
Pemrosesan berkas citra data-wajah\sl\10.pgm
Pemrosesan berkas citra data-wajah\sl\2.pgm
Pemrosesan berkas citra data-wajah\sl\3.pgm
Pemrosesan berkas citra data-wajah\sl\4.pgm
Pemrosesan berkas citra data-wajah\sl\5.pgm
Pemrosesan berkas citra data-wajah\sl\6.pgm
Pemrosesan berkas citra data-wajah\sl\7.pgm
----------------------------
Pemrosesan berkas citra data-wajah\sl0\1.pgm
Pemrosesan berkas citra data-wajah\sl0\10.pgm

C:\LatOpenCV>
```

Perhatikan bahwa untuk setiap subfolder di folder data-wajah, berkas `8.pgm` **dan** `9.pgm` **tidak dilibatkan. Hal ini disengaja karena kedua berkas tersebut dapat digunakan untuk pengujian.**

Setelah eksekusi skrip berakhir, akan terbentuk berkas bernama `pelatihan.yml`. **Hal ini bisa dicek melalui perintah berikut:**

```
.....................................................................
C:\LatOpenCV>dir *.yml<P
 Volume in drive C is Acer
 Volume Serial Number is D4EC-C220

 Directory of C:\LatOpenCV

25/02/2019   19:37        42.228.985
pelatihan.yml
               1 File(s)     42.228.985 bytes
               0 Dir(s)  262.905.032.704 bytes
free

C:\LatOpenCV>
```

Perlu diketahui, YAML (ekstensi berkas berupa .yml) adalah format data dalam bentuk teks. Berkas pelatihan.yml ini akan digunakan pada proses pengenalan wajah.

Sekarang, marilah memahami skrip pelatihan.py. Pertama-tama, perintah berikut digunakan untuk membentuk objek pengenal wajah:

```
pengenalWajah = cv2.face.LBPHFaceRecognizer_create()
```

Dalam hal ini, metode LBPH yang digunakan. Adapun perintah berikut digunakan untuk membentuk objek pendeteksi wajah:

```
detektor = cv2.CascadeClassifier(
              "haarcascade_frontalface_default.xml")
```

Selanjutnya, terdapat pembuatan fungsi bernama perolehCitraDanLabel(), yang ditujukan untuk membaca berkas-berkas citra yang terdapat pada folder lintasan. Perlu diketahui, folder lintasan harus mengadung berkas-berkas dengan hierarki seperti yang ditunjukkan pada Gambar 12.7.

Pernyataan berikut digunakan untuk memperoleh senarai nama-nama subfolder yang berada pada lintasan:

```
daftarFolderCitra = [os.path.join(lintasan, f) \
    for fin  os.listdir(lintasan)]
```

Adapun dua senarai yang digunakan untuk mencatat data wajah dan label wajah diberi nilai kosong pada keadaan awal. Hal ini dapat dilihat pada dua pernyataan berikut:

```
daftarSampelWajah =[]
daftaridWajah= []
```

Perintah berikut digunakan untuk memproses setiap subfolder yang berada pada senarai daftarFolderCitra:

```
for folderCitra in daftarFolderCitra:
```

Pada setiap iterasi, folderCi tra berisi satu nama subfolder yang mengandung nama-nama berkas citra. Adapun

```
daftarBerkas = os.listdir
```

digunakan untuk mendapatkan nama-nama berkas citra yang terdapat pada subfolder yang tercatat di folderCitra. **Berkas masing-masing diproses melalui:**

```
for berkas in daftarBerkas:
```

Nama lengkap berkas (yang mengandung nama subfolder) dibentuk melalui:

```
berkasCitra = folderCitra + "\\" + berkas
```

Selanjutnya, berkas citra dibaca dengan mode skala keabu-abuan dengan menggunakan pernyataan:

```
citra = cv2.imread(berkasCitra, 0)
```

Variabel idWajah **digunakan untuk mendapatkan ID pada subfolder yang berisi berks-berkas citra, yaitu dengan mengambil bilangannya saja. Sebagai contoh, ID untuk subfolder** s12 **berupa 12. Hal ini dilakukan melalui:**

```
idWajah = os.path.basename(folderCitra)[1:]
idWajah = int(idWajah)
```

Pengambilan wajah pada ci tra **dilakukan melalui:**

```
daftarWajah = detektor.detectMultiScale(citra)
```

Selanjutnya, wajah-wajah yang diperoleh diproses untuk dimasukkan ke dalam senarai daftarSampelWajah **dan** daftaridWaj ah **menggunakan:**

```
for (x, y, w, h) in daftarWajah:
    daftarSampelWajah.append(
                    citra[y : y + h, x    x + w])
    daftaridWajah.append(idWajah)
```

Pertama-tama, data wajah ditambahkan ke senarai daftarSampelWajah. Kemudian, data ID wajah ditambahkan ke senarai daftaridWajah.

Proses pelatihan ditangani melalui tiga pernyataan berikut:

```
daftarWajah, daftaridWajah = perolehCitraDanLabel(
                                    "data-wajah")
pengenalWajah.train(daftarWajah, np.array(daftaridWajah))
pengenalWajah.save("pelatihan.yml")
```

Mula-mula, fungsi perolehCitraDanLabel() dipanggil dengan melewatkan nama folder "data-wajah". Setelah eksekusi fungsi ini berakhir, data nilai baliknya diberikan ke metode train() milik pengenal wajah. Metode inilah yang melakukan pelatihan kepada pengenal data dengan menggunakan data yang dibentuk oleh perolehCitraDanLabel(). Setelah pelatihan berakhir, data hasil pelatihan disimpan ke berkas pelatihan.yml melalui:

```
pengenalWajah.save("pelatihan.yml")
```

Berkas dengan ekstensi .yml inilah yang nanti digunakan untuk memprediksi suatu citra yang berisi wajah seseorang.

12.4.4 Pengujian Pengenalan Wajah

Setelah pelatihan kepada pengenal wajah dilaksanakan, data dalam berkas YAML dapat digunakan untuk memprediksi wajah seseorang. Mandannya seperti berikut:

Berkas : prediksi.py

```
# Bagian untuk mengenali wajah seseorang

import numpy as np
import os
import cv2

pengenalWajah    cv2.face.LBPHFaceRecognizer_create()
```

```
detektor    cv2.CascadeClassifier(
               "haarcascade frontalface default.xml")

def prediksiWajah(namaBerkas):
    citra = cv2.imread(namaBerkas)

    if citra is None:
        print("Tidak dapat membaca berkas citra")
        return

    abuAbu = cv2.cvtColor{citra, cv2.COLOR_BGR2GRAY)
    daftarWajah = detektor.detectMultiScale(
        abuAbu, scaleFactor = 1.3, minNeighbors = 5)
    if daftarWajah is None:
        print("Wajah tidak terdeteksi")
        return

    for {x, y, w ,h) in daftarWajah:
        cv2.rectangle {citra, (x, y), (x + w, y + h),
                  (255, 0, 0), 2)
        wajah = abuAbu[ y:y+h, x:x+w]
        labelid, konfiden = pengenalWajah.predict(wajah)
        if konfiden < 500:
            cv2.putText(citra, "(%s) %.Of" % \
                        (labelid, konfiden),
                        (x , y - 2) ,
                        cv2.FONT_HERSHEY PLAIN,
                        1, (0, 255, 0))
        else:
            cv2.putText {citra, "Entah", (x, y - 2),
                        cv2.FONT_HERSHEY_PLAIN, 1,
                        (0, 255, 0))

    cv2.imshow("Hasil", citra)
    cv2.waitKey(0)

# Program utama
pengenalWajah.read("pelatihan.yml")
prediksiWajah("data-wajah/sl5/8.pgm")
prediksiWajah("data-wajah/s40/8.pgm")
prediksiWajah("data-wajah/sl2/8.pgm")
prediksiWajah("data-wajah/s21/8.pgm")
    iksiWajah("lena.png")
```

Akhir berkas

Pertama-tama, objek pengenal wajah dibentuk. Perintahnya berupa:

```
    pengenalWajah    cv2.face.LBPHFaceRecognizer create()
```

Adapun perintah berikut digunakan untuk membuat objek pendeteksi wajah:

```
detektor = cv2.CascadeClassifier(
                "haarcascade frontalface_default.xml")
```

Fungsi prediksiWaj ah () **digunakan untuk memprediksi wajah seseorang berdasarkan citra yang diberikan. Mula-mula, citra dibaca. Kemudian, dikonversi menjadi citra berskala keabu-abuan. Kemudian, wajah yang diperoleh dikirimkan ke metode** predict () **milik pengenal wajah. Metode ini memberikan nilai balik berupa label yang didapat dan tingkat keyakinan (konfiden). Dalam hal ini, nilai konfiden yang kecil menyatakan dapat dipercaya. Pada skrip, nilai konfiden yang lebih kecil dari 500 menyatakan bahwa citra dapat diidentifikasi (walaupun hasilnya belum tentu benar). Pada kondisi ini, tulisan yang berisi label (yang ditulis dalam tanda kurung) dan nilai konfiden ditampilkan.**

Pada skrip, fungsi prediksiWaj ah () **dipanggil lima kali. Secara berturutan digunakan untuk menguji citra** s15/8 .pgm, s40/8 .pgm, s 12 / 8 . pgm, s21 / 8 . pgm, **dan** lena. png. **Hasil yang didapat menunjukkan bahwa empat citra pertama dapat teridentifikasi dengan benar, sedangkan** lena. png **teridentifikasi tidak benar karena dianggap memiliki ID 15. Gambar 12.8 menunjukkan hasil untuk** lena.png.

Gambar 1 .8 Lena diberi label 15 yang menyatakan kesalahan identifikasi