MODUL 11 DASAR-DASAR DETEKSI OBYEK

A. TUJUAN

- Mahasiswa dapat memahami konsep deteksi objek
- Mahasiswa mampu memahami beberapa metode yang dapat digunakan dalam proses deteksi objek
- Mahasiswa mampu memahami konsep Feature Matching
- Mahasiswa mampu memahami konsep face detection
- Mahasiswa dapat mengimplementasikan beberapa metode dalam proses deteksi objek dan proses Feature Matching dan Face detection menggunakan Python pada Google Colab

B. ALAT DAN BAHAN

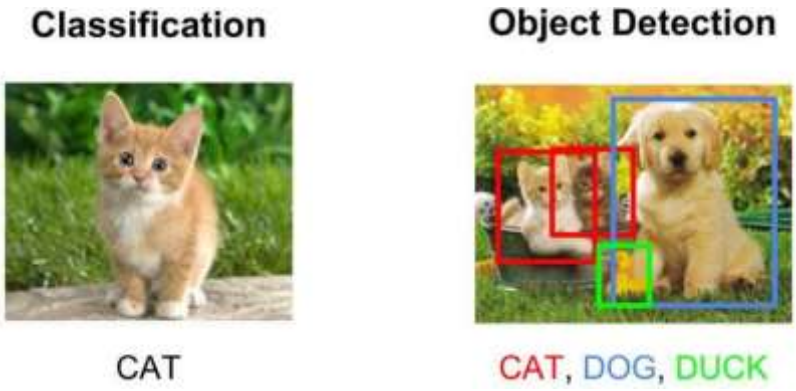
1. PC/LAPTOP
2. Github
3. *Google Colaborator*

C. DASAR TEORI

C.1 Dasar Deteksi Obyek

C.1.1 Konsep Deteksi Objek

Deteksi objek (object detection) merupakan salah satu teknik visi computer (computer vision) yang berfungsi untuk mengidentifikasi dan menemukan objek dalam sebuah gambar atau video. Secara khusus, proses deteksi objek akan menggambar kotak pembatas di sekitar objek yang terdeteksi, sehingga dapat diketahui dimanakah lokasi objek tersebut pada gambar / video. Tujuan dari deteksi objek adalah untuk mendeteksi semua *instance* objek dari kelas yang telah diketahui, seperti orang, mobil atau wajah pada gambar. Umumnya, hanya sedikit objek yang terdapat pada sebuah gambar, tetapi terdapat banyak sekali jumlah kemungkinan lokasi dan skala di mana objek tersebut dapat muncul. Berikut ini adalah ilustrasi dari konsep deteksi objek:



Dalam sistem computer vision, pendeteksian objek merupakan tahap pertama yang dilakukan karena hasil dari deteksi objek dapat memungkinkan untuk memperoleh informasi lebih lanjut mengenai objek yang terdeteksi dan lokasi objek tersebut. Setelah objek terdeteksi (misalnya, wajah), bisa didapatkan informasi lebih lanjut, diantaranya: (i) mengenali contoh spesifik (misalnya, untuk mengidentifikasi subjek dari wajah yang terdeteksi), (ii) melacak objek pada urutan gambar (misalnya, untuk melacak wajah dalam video), dan (iii) mengekstrak informasi lebih lanjut tentang objek (misalnya, untuk menentukan jenis kelamin subjek). Beberapa contoh implementasi deteksi objek antara lain:

1. Video surveillance (CCTV), model deteksi objek mampu melacak banyak orang sekaligus, secara real-time
2. Crowd counting (Penghitungan kerumunan), untuk area padat pengunjung seperti taman hiburan, mall, atau alun-alun kota, deteksi objek dapat membantu pemilik bisnis dan pemerintahan kota untuk mengukur dengan lebih efektif berbagai jenis lalu lintas, dan beberapa rencana seperti jam operasional toko, shift karyawan, dan pengalokasian sumber daya.
3. Anomaly detection, misal di bidang pertanian, model pendeteksian objek dapat secara akurat mengidentifikasi dan menemukan lokasi potensi penyakit tanaman, sehingga memungkinkan petani untuk mendeteksi ancaman terhadap hasil panen mereka yang tidak dapat dilihat dengan mata telanjang.
4. Self-driving cars, deteksi objek merupakan Teknik yang mendasari terciptanya mobil tanpa pengemudi. Sistem ini harus dapat mengidentifikasi, menemukan, dan melacak objek di sekitarnya agar dapat bergerak di jalan raya dengan aman dan efisien.

C.1.2 Template Matching

Pencocokan template (template matching) merupakan bentuk yang paling dasar dari deteksi objek. Template matching adalah metode untuk mencari dan menemukan lokasi gambar template pada gambar yang lebih besar. Diperlukan dua gambar untuk menerapkan metode template matching, yaitu:

- Source image (I): gambar masukan yang diharapkan cocok dengan template.
- Template image (T): potongan objek yang akan dicari pada source image.

Untuk menemukan template di source image sumber, dilakukan perrgeseran template dari kiri ke kanan dan dari atas ke bawah:

Di setiap lokasi (x, y), sebuah metrik dihitung untuk merepresentasikan seberapa "baik" atau "buruk" kecocokan tersebut. Perhitungan koefisien korelasi dilakukan untuk menentukan seberapa mirip intensitas piksel dari dua potongan gambar tersebut. Untuk setiap lokasi ***T*** pada ***I***, metrik hasil perhitungan akan disimpan dalam matriks hasil ***R***. Setiap koordinat (x, y) pada source image akan berisi elemen di matriks hasil ***R***:

Lokasi cerah dari matriks hasil R menunjukkan nilai kecocokan yang terbaik, sedangkan daerah gelap menunjukkan korelasi yang sangat kecil antara gambar sumber dan template.

OpenCV mengimplementasikan template matching menggunakan fungsi **matchTemplate()**. Terdapat 6 metode yang tersedia pada matchTemplate(), yaitu:

1. method=TM_SQDIFF

$$R(x,y) = \sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2$$

2. method=TM_SQDIFF_NORMED

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

3. method=TM_CCORR

$$R(x,y) = \sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))$$

4. method=TM_CCORR_NORMED

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

5. method=TM_CCOEFF

$$R(x,y) = \sum_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))$$

6. method=TM_CCOEFF_NORMED

$$R(x,y) = \frac{\sum_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}}$$

Dokumentasi template matching pada OpenCV dapat dilihat pada link berikut: https://docs.opencv.org/3.4/de/da9/tutorial_template_matching.html

C.1.3 Edge Detection

Deteksi tepi (Edge detection) adalah operasi yang dijalankan untuk mendeteksi garis tepi (edges) yang membatasi dua wilayah citra yang memiliki tingkat kecerahan yang berbeda. Deteksi tepi sebuah citra digital merupakan proses untuk mencari perbedaan intensitas yang menyatakan batas-batas suatu objek (sub-citra) dalam keseluruhan citra digital yang dimaksud. Suatu titik (x,y) dikatakan sebagai tepi dari suatu citra bila titik tersebut mempunyai perbedaan yang tinggi dengan tetangganya. Deteksi tepi banyak dipakai untuk mengidentifikasi suatu objek dalam sebuah gambar. Terdapat 3 metode edge detection yang disediakan oleh OpenCV, yaitu:

1. Sobel Edge Detection

   Sobel adalah salah satu detektor tepi yang paling umum digunakan. Filter Sobel bekerja dengan menghitung gradien intensitas citra pada setiap piksel dalam sebuah citra. Saat menggunakan filter ini, gambar dapat diproses dalam arah X dan Y secara terpisah atau bersamaan. Sobel didasarkan pada konvolusi gambar dalam arah horizontal dan vertical menggunakan filter 3x3, berikut:

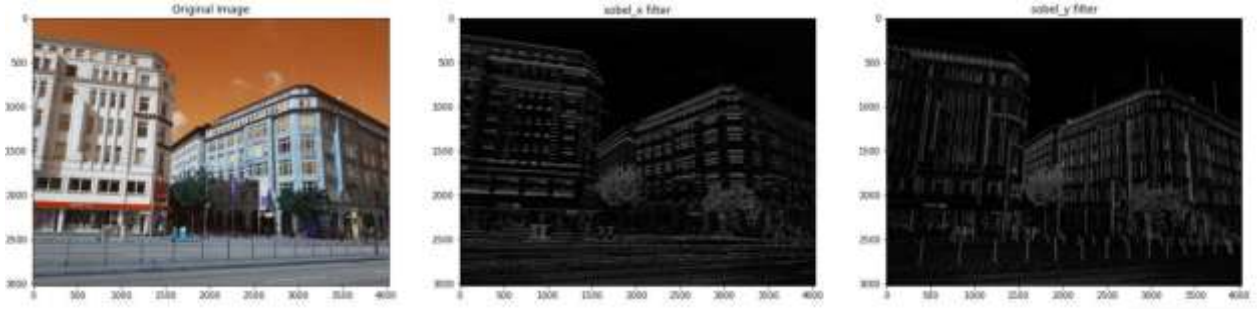
-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Contoh hasil dari sobel edge detection adalah sebagai berikut:



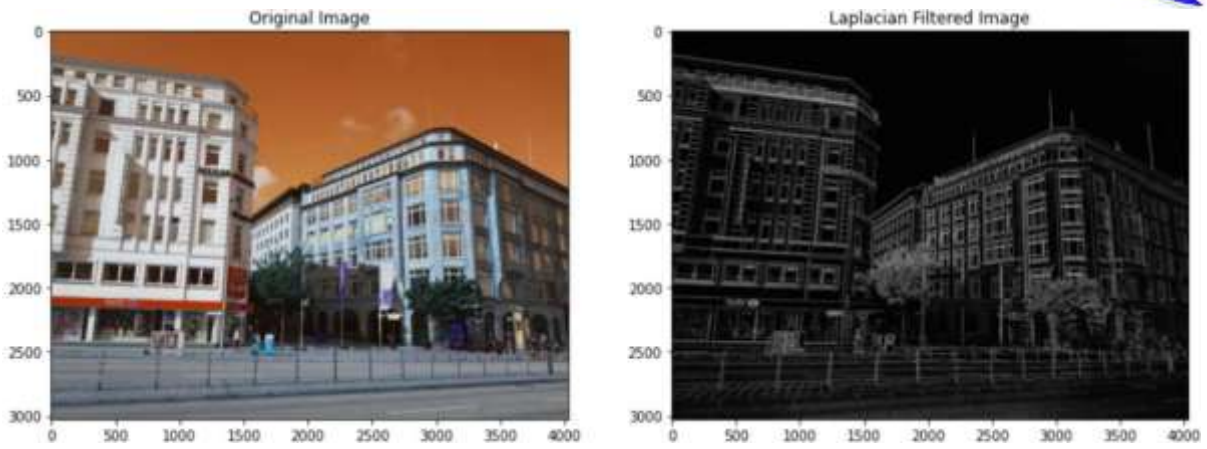
2. Laplacian Edge Detection

   Kernel 3x3 yang umumnya digunakan pada Laplacian Edge Detection adalah sebagai berikut:

0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

   Metode ini sangat sensitive terhadap noise, maka dari itu untuk memperbaikinya, sebuah gambar umumnya akan dihaluskan terlebih dahuku menggunakan Gaussian filter sebelum menerapkan Laplacian. Contoh hasil dari laplacian edge detection adalah sebagai berikut:



3. Canny Edge Detection

   Canny Edge Detection merupakan metode yang paling umum digunakan dan paling efektif. Metode ini bekerja dengan 4 tahap, yaitu:

   1. Menghaluskan gambar dengan filter Gaussian untuk mengurangi noise
   2. Menghitung gradien menggunakan salah satu operator gradien Sobel
   3. Mengekstraksi titip tepi: Non-maximum suppression
   4. Menghubungkan dan thresholding: Hysteresis

Canny edge detection didasarkan pada konsep bahwa intensitas gambar akan bernilai tinggi di tepinya. Permasalahan pada konsep ini adalah jika gambar memiliki noise, maka noise tersebut juga akan terdeteksi sebagai tepi. Sehingga langkah pertama dalam Canny edge detection adalah menghilangkan noise, salah satu caranya dengan menerapkan filter Gaussian untuk menghaluskan gambar sebelum melanjutkan pemrosesan.

Langkah kedua dalam proses deteksi tepi dengan Canny adalah komputasi gradien yang dilakukan dengan menghitung tingkat perubahan intensitas (gradien) pada gambar di sepanjang arah gradien. Citra yang telah dihaluskan pada langkah pertama kemudian difilter dengan kernel Sobel, baik horizontal maupun vertikal untuk mendapatkan turunan pertama dalam arah horizontal (Gx) dan vertikal (Gy). Dari kedua gambar tersebut, dapat dicari gradien tepi dan arah untuk setiap piksel sebagai berikut:

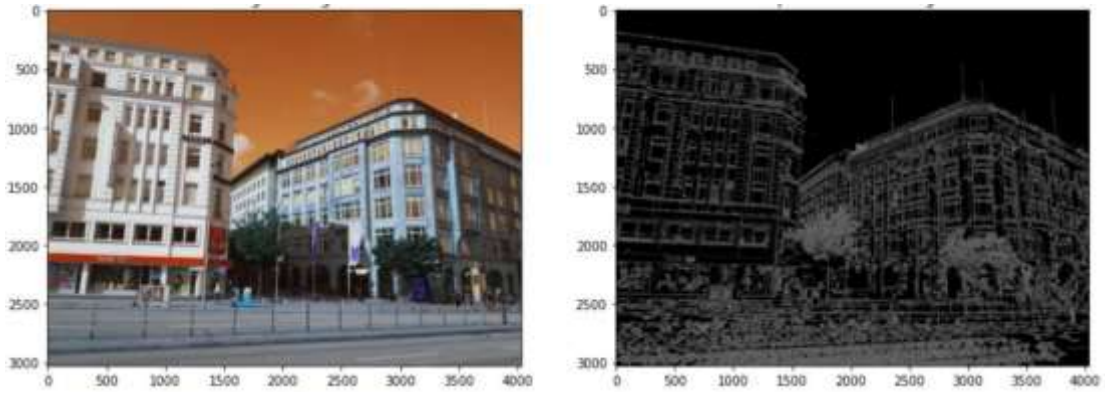
$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Intensitas gambar berada pada titik tertinggi di tepinya, tetapi pada kenyataannya, intensitas tidak mencapai puncaknya pada satu piksel; sebaliknya, ada piksel bersebelahan dengan intensitas tinggi. Di setiap lokasi piksel, Canny edge detection akan membandingkan piksel dan memilih local maximal di 3X3 ketetanggaan ke arah gradien. Proses ini dikenal sebagai Non-maximum suppression.

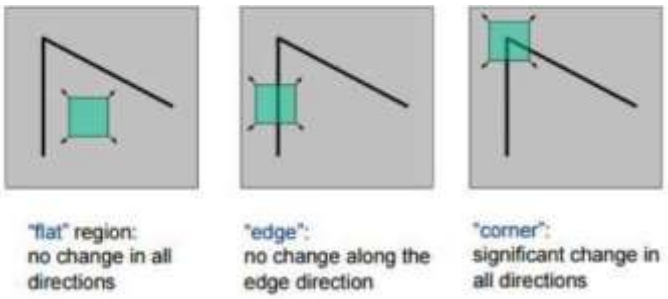
Langkah ketiga menghasilkan garis tepi berukuran tipis tetapi patah. Langkah terakhir adalah memperbaiki / menyambungkan tepi-tepi yang patah tersebut dengan menggunakan teknik hysteresis thresholding. Pada hysteresis thresholding, terdapat dua ambang batas: ambang batas tinggi dan ambang rendah (high and low threshold). Setiap piksel dengan nilai gradien lebih tinggi dari high threshold secara otomatis disimpan sebagai tepi. Sedangkan piksel yang gradiennya berada di antara high threshold dan low threshold akan ditangani dengan dua cara. Piksel akan diperiksa apakah memiliki kemungkinan koneksi dengan tepi; piksel akan disimpan jika tersambung dan akan dibuang/diabaikan jika sebaliknya. Piksel dengan gradien lebih rendah dari low threshold akan dibuang secara otomatis.

Contoh hasil dari canny edge detection adalah sebagai berikut:



C.1.4 Corner Detection

Sudut dapat diartikan sebagai persimpangan dua sisi. Titik sudut tidak bisa didefinisikan pada piksel tunggal, karena di sana hanya terdapat pada satu gradien di setiap titik. Gradien adalah arah perubahan intensitas kecerahan dalam suatu citra. Sudut adalah fitur penting dalam sebuah gambar, dan umumnya disebut sebagai interest point yang tidak berubah apabila dilakukan translasi, rotasi, dan iluminasi.



Terdapat 2 metode corner detection yang disediakan oleh OpenCV, yaitu:

1. **Harris corner detection**

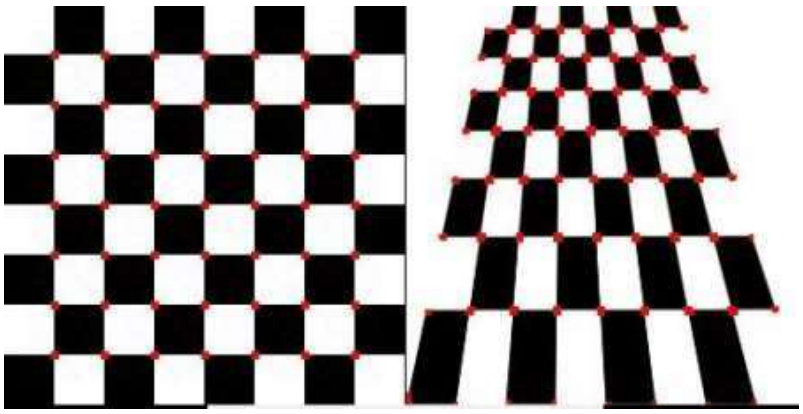
   Harris Corner Detection adalah system pendeteksi sudut yang sering digunakan karena mampu menghasilkan nilai yang konsisten pada citra yang mengalami rotasi, penskalaan, variasi pencahayaan, maupun memiliki banya noise pada gambar. Deteksi sudut dengan metode Harris didasarkan pada variasi intensitas sinyal. Variasi intensitas yang besar menunjukkan adanya sudut pada citra. Cara kerja metode Harris adalah dengan menemukan perbedaan intensitas untuk perpindahan (u, v) ke segala arah, dengan persamaan berikut:

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} \underbrace{[I(x+u,y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2$$

   Window function dapat berupa filter Gaussian yang memberi bobot pada piksel di bawahnya. Fungsi E (u, v) ini harus dimaksimalkan untuk deteksi sudut.
   Pada OpenCV, metode yang digunakan adalah:
   cv2.cornerHarris(src, dest, blockSize, kSize,
   freeParameter, borderType) dengan parameter:

   - Src: citra input
   - Dest: cistra untuk menyimpan hasil Harris detector, ukurannya sama dengan citra input
   - blockSize: ukuran ketetanggaan
   - ksize: kernel size untuk operator Sobel
   - freeParameter: parameter bebas untuk Harris detector
   - borderType: metode pixel extrapolation

Contoh hasil Harris corner detection adalah sebagai berikut:



Dokumentasi lengkap dapat dibaca pada link berikut:

https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html

2. Shi-Tomasi Corner Detection

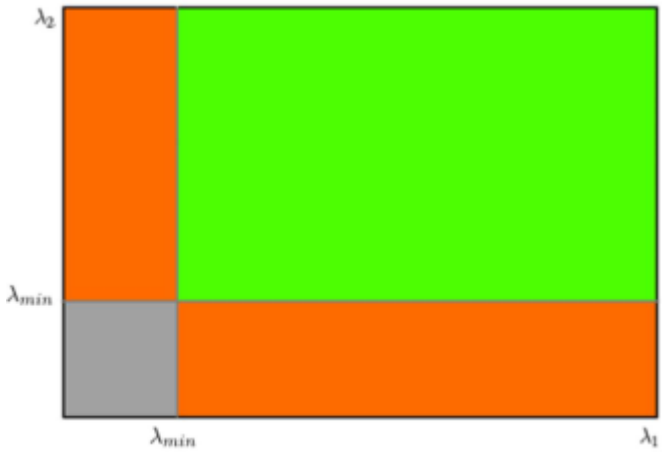
Shi-Tomasi Corner Detection menggunakan dasar bahwa sudut dapat dideteksi dengan mencari perubahan signifikan ke segala arah. Misal terdapat sebuah window kecil pada gambar kemudian memindai/scan seluruh gambar untuk mencari sudut. Menggeser window kecil ini ke segala arah akan menghasilkan perubahan besar pada tampilan, jika window tersebut terletak di sudut.
Pada ametode Harris, penilaian sebuah titik dikatakan sudut atau bukan dilakukan dengan memberikan nilai R

$$R = det(M) - k(\mathrm{Tr}(M))^2$$

Shi-Tomasi mengajukan ide untuk penilaian R hanya dengan menentukan nilai Eigen dari M saja

$$R = \min(\lambda_1, \lambda_2)$$

Jika nilainya lebih besar dari nilai threshold, maka dianggap sebagai sudut. Jika diplot di ruang λ1
− λ2, akan menjadi seperti berikut:



Dari gambar tersebut dapat dilihat bahwa hanya jika λ1 dan λ2 di atas nilai minimum, λmin dianggap sebagai sudut (area berwarna hijau).

OpenCV memiliki fungsi, cv.goodFeaturesToTrack(). Fungsi ini akan menemukan N sudut terkuat pada gambar dengan metode Shi-Tomasi. Citra input harus berupa gambar grayscale. Pengaturan
/ parameter dari metode ini antara lain: (a) jumlah sudut yang ingin ditemukan, (b) tingkat kualitas, yang merupakan nilai antara 0-1, yang menunjukkan kualitas minimum dari sudut, dan
(c) jarak euclidean minimum antar sudut yang terdeteksi. Dengan parameter-parameter tersebut, fungsi akan menemukan sudut-sudut pada gambar. Semua sudut di bawah tingkat kualitas akan ditolak. Sudut yang tersisa akan diurutkan berdasarkan kualitas dalam urutan menurun. Kemudian fungsi akan mengambil sudut terkuat pertama, membuang semua sudut terdekat dalam kisaran jarak minimum dan mengembalikan N sudut terkuat.

Dokumentasi lengkap dapat dibaca pada link berikut:

https://docs.opencv.org/master/d4/d8c/tutorial_py_shi_tomasi.html

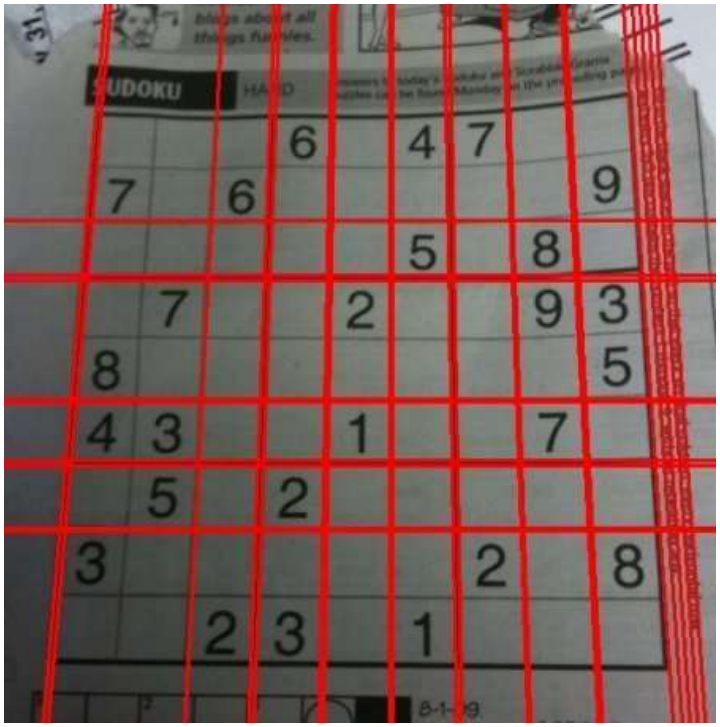
C.1.5 Grid Detection

Grid detection dapat dilakukan dengan metode Hough Transform. Hough Transform adalah teknik populer untuk mendeteksi bentuk apa pun, jika bentuk tersebut dapat direpresentasikan dalam bentuk matematika. Hough Transform dapat mendeteksi bentuk bahkan jika bentuknya rusak atau sedikit terdistorsi.

Tahapan untuk melakukan grid detection adalah sebagai berikut:

1. Menerapkan Canny Edge Detection
2. Dilasi gambar tepi (Canny bisa menemukan kedua tepi pemisah pada grid sebagai tepi yang berbeda, dilatasi dapat membuat kedua tepi ini bergabung lagi)
3. Erosi (setelah proses dilasi, garis border menjadi tebal sehingga Hough akan mendeteksi banyak garis)
4. Menerapkan HoughLines dengan fungsi cv2.HoughLines() pada OpenCV
5. Menggabungkan garis yang mirip (similar)

Contoh hasil Hough Transform adalah sebagai berikut:



Dokumentasi lengkap mengenai HoughTransform dapat dibaca pada link berikut:

https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html

C.1.6 Contour Detection

Kontur adalah kurva tertutup yang menghubungkan semua titik kontinu yang memiliki beberapa warna atau intensitas. Kontur mewakili bentuk objek yang ditemukan dalam gambar. Deteksi kontur adalah teknik yang berguna untuk analisis bentuk serta

deteksi dan pengenalan objek. Kontur adalah kumpulan abstrak dari titik dan segmen yang sesuai dengan bentuk objek pada gambar. Kita dapat memanipulasi kontur dalam program kita seperti menghitung jumlah kontur, menggunakannya untuk mengkategorikan bentuk objek, atau memotong objek dari gambar (segmentasi gambar).

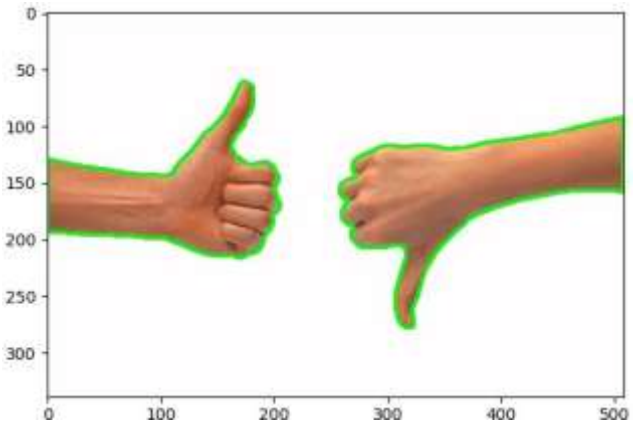
Untuk akurasi yang lebih baik, berikut ini merupakan tahapan untuk mendeteksi kontur pada gambar:

1. Ubah citra input menjadi citra biner (bisa didapat dengan mengimplementasikan thresholding atau canny edge detection).
2. Menemukan kontur menggunakan fungsi findContours() pada OpenCV.
3. Menggambarkan kontur pada citra input menggunakan fungsi drawContours().

Fungsi findContours menerima tiga buah parameter, yaitu :

- Parameter pertama adalah Image yang akan dianalisis konturnya
- Parameter kedua menunjukkan metode retrieval. Metode retrieval tersebut dapat digunakan untuk melihat hierarcy dari kontur dalam sebuah citra. Hierarcy menunjukkan hubungan parent-child. Parents merupakan shape/contour yang lebih luar dibandingkan dengan child atau dengan kata lain, parents merupakan shape/contour yang melingkupi child.
- Parameter ketiga menunjukkan metode yang digunakan untuk proses approximation dari shape. Methode tersebut dapat berupa metode cv2.CHAIN_APPROX_SIMPLE atau metode cv2.CHAIN_APPROX_NONE. Pada flag cv2.CHAIN_APPROX_SIMPLE, class cv2.findContours akan mengembalikan seminimum mungkin jumlah koordinat yang menunjukkan contours sedangkan pada flag cv2.CHAIN_APPROX_SIMPLE, class cv2.findContours akan mengembalikan sebanyak mungkin jumlah koordinat yang menunjukkan contours.

Contoh hasil contour detection adalah sebagai berikut:



Dokumentasi lengkap dapat dibaca pada link berikut:

https://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html

C.2. FEATURE MATCHING DAN FACE DETECTION

C.2.1 Konsep Feature Matching

Feature matching melakukan ekstraksi ekstraksi fitur penting dari sebuah citra menggunakan ide dasar dari corner, edge, dan contour detection. Selanjutnya akan dilakukan perhitungan distance untuk mencari kecocokan antara image sumber dengan image template. Dengan menggunakan cara ini, anda tidak perlu lagi menggunakan template yang ada di image sumber. Jika objek pada citra template tidak berukuran sama dengan citra sumber, deteksi tetap bisa dilakukan dengan baik. Pada praktikum kali ini akan dicobakan 2 metode Feature Matching yaitu:

A Brute-Force Matching menggunakan ORB Descriptor (Orient FAST and Rotated BRIEF)

B Brute-Force Matching menggunakan SIFT Descriptor dan Test Ratio (Scale-Invariant Feature Transform)

ORB adalah berasal dari "OpenCV Labs". Algoritma ini diajukan oleh Ethan Rublee, Vincent Rabaud, Kurt Konolige dan Gary R. Bradski dalam artikel mereka berjudul "*ORB: An efficient alternative to SIFT or SURF*" pada tahun 2011. Seperti yang tertulis pada judulnya, ORB adalah alternatif pengganti dari SIFT dan SURF dalam hal biaya komputasi, kinerja matching dan terutama adalah paten. SIFT dan SURF dipatenkan dan kita harus membayar untuk menggunakannya, beda dengan ORB yang tidak perlu membayar untuk penggunaannya.

ORB pada dasarnya adalah perpaduan dari detektor keypoint FAST dan deskriptor

BRIEF dengan banyak modifikasi untuk meningkatkan kinerjanya. Sebagai langkah awal, FAST digunakan untuk menemukan keypoint, lalu Harris Corner detection digunakan untuk mencari N-point tertinggi. Akan tetapi FAST tidak menghitung orientasinya. Modifikasi dilakukan agar algoritma ini invariant terhadap orientasi. Penjelasan lebih detil dapat dilihat pada dokumentasi OpenCV: https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html

ORB Descriptor telah disediakan pada library Opencv cv.ORB(). Berikut adalah code untuk menunjukkan penggunakan ORB Descriptor. Pada praktikum ini anda diminta untuk menggunakan file reeses_puff dan cereals yang telah disediakan. Perhatikan pada 2 gambar tersebut, reeses puff memiliki resolusi yang lebih besar daripada reeses puff pada image cereals.

```
def display(img,cmap='gray'):
  fig = plt.figure(figsize=(12,10))
  ax = fig.add_subplot(111)
  ax.imshow(img,cmap='gray')
reeses = cv.imread('/content/drive/MyDrive/Polinema/Kuliah/PCVK/Images/facedet/reeses_puffs.png'
display(reeses)
```


```
cereals = cv.imread('/content/drive/MyDrive/Polinema/Kuliah/PCVK/Images/facedet/many_cereals.jpg
display(cereals)
```

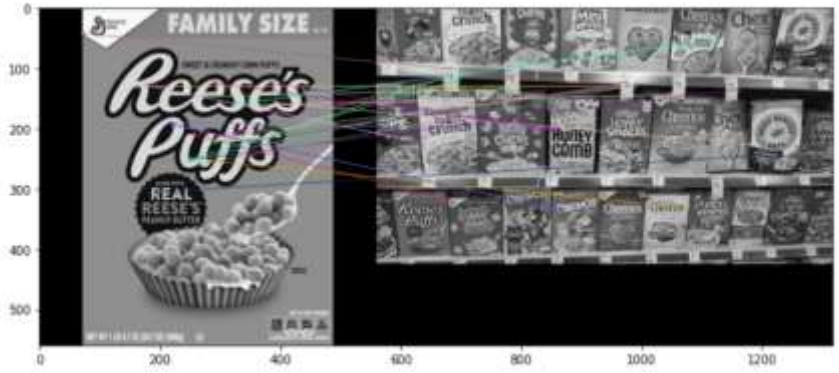
```
orb = cv.ORB_create()
kp1,des1 = orb.detectAndCompute(reeses,None)
kp2,des2 = orb.detectAndCompute(cereals,None)
bf = cv.BFMatcher(cv.NORM_HAMMING,crossCheck = True)
matches = bf.match(des1,des2)
single_match = matches[0]
single_match.distance
```

```
71.0
```

```
len(matches)
```

```
139
```

```
matches = sorted(matches,key=lambda x:x.distance)
reeses_match = cv.drawMatches(reeses,kp1,cereals,kp2,matches[:25],None,flags=2)
display(reeses_match)
```



Perhatikan hasil pencocokan diatas. Dengan memilih 25 titik yang paling cocok, hasil dari pencocokan pada kasus diatas tidak akurat sama sekali.

Pada pembahasan sebelumnya tentang Harris corner detection misal, metode-

metodenya adalah rotation-invariant, artinya walaupun image dalam kondisi terrotasi, corner dapat dideteksi dengan baik. Hal ini jelas sekali, karena corner akan tetap berupa corner walaupun objek dirotasikan. Berbeda kasus dengan scaling. Corner bisa jadi bukan lagi corner jika citra discaling. Perhatikan gambar berikut. Corner pada citra kecil dengan window berukuran kecil akan berupa garis jika diperbesar dan menggunakan window berukuran sama. Hal ini menunjukkan bahwa Harris corner merupakan algoritma yang scale-invariant.

Pada tahun 2004, D. Lowe dari University of British Columbia, menemukan algoritma baru, Scale Invariant Feature Transform (SIFT) dalam artikelnya yang berjudul *"Distinctive Image Features from Scale-Invariant Keypoints",* yang mengekstrak keypoint dan menghitung deskriptornya. Artikel ini mudah dipahami dan dianggap sebagai materi terbaik yang tersedia di SIFT.

Terdapat 4 tahap yang terjadi pada algoritma SIFT. Penjelasan lebih detil dapat anda baca pada dokumentasi OpenCV:
https://docs.opencv.org/master/da/df5/tutorial_py_sift_intro.html . Berikut akan ditunjukkan penggunaan algoritma SIFT pada image reese puffs dan cereal. Hasil yang

```
!pip install opencv-contrib-python==4.4.0.44
```

```
sift = cv.xfeatures2d.SIFT_create()
kp1,des1 = sift.detectAndCompute(reeses,None)
kp2,des2 = sift.detectAndCompute(cereals, None)
bf = cv.BFMatcher()
matches = bf.knnMatch(des1,des2,k=2)

good = []

#makin kecil jarak makin cocok
#ratio match1 < 75% match2
for match1,match2 in matches:
  #jika jarak match1 lebih kecil dari jarak 75% match2
  #descriptor disimpan
  if match1.distance < 0.75*match2.distance:
    good.append([match1])

len(good)
```

```
79
```

```
len(matches)
```

```
1581
```

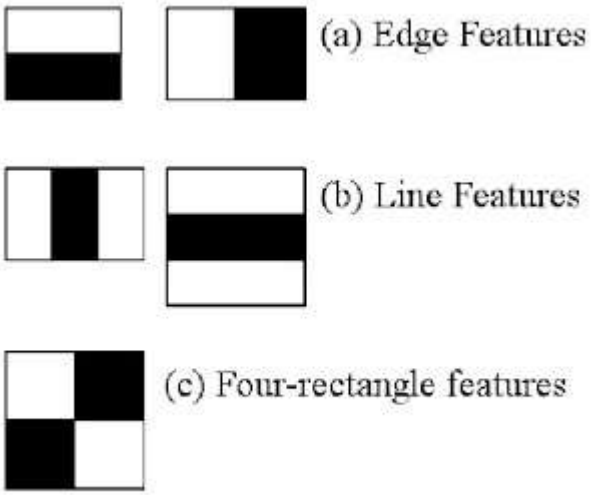
```
sift_matches = cv.drawMatchesKnn(reeses,kp1,cereals,kp2,good,None,flags=2)
display(sift_matches)
```

didapatkan juga jauh lebih baik jika dibandingkan dengan ORB Descriptor. Untuk menggunakan SIFT descriptor, anda harus menggunakan OpenCV Versi 4.4.0.44 keatas. Lakukan instalasi menggunakan pip terlebih dahulu sebalum menjalankan code SIFT.

C.2.2 Konsep Face Detection

Pada materi ini akan dijelaskan cara kerja face detection menggunakan Haar Cascades, yaitu komponen kunci dari framework deteksi object Viola-Jones. Metode ini dapat mendeteksi wajah dengan cepat pada sebuah image dan mengetahui lokasinya dengan tepat. Metode ini digunakan sebagai tahap awal dalam mengenali wajah (Facial Recognition).

Pada tahun 2001, Paul Viola dan Michael Jones mempublikasikan metode face detection berdasarkan pada konsep sederhana dari beberapa fitur kunci. Ide lain yang digunakan adalah melakukan perhitungan awal dari integral image untuk mempercepat waktu komputasi. Perhatikan tipe-tipe fitur utama yang diajukan oleh Viola dan Jones.



Tiap fitur adalah merupakan nilai tunggal yang didapatkan dengan mengurangkan rata-rata pixel pada area kotak putih dengan rata-rata pixel pada area kotak hitam.

0	0	1	1
0	0	1	1
0	0	1	1
0	0	1	1

0	0.1	0.8	1
0.3	0.1	0.7	0.8
0.1	0.2	0.8	0.8
0.2	0.2	0.8	0.8

mean(area gelap) – mean(area terang).

Sum([0.8, 1, 0.7, 0.8, 0.8, 0.8, 0.8, 0.8]) = 6.5

Sum([0, 0.1, 0.3, 0.1, 0.1, 0.2, 0.2, 0.2]) = 1.2

Mean = 6.5 / 8 = 0.8125

Mean = 1.2 / 8 = 0.15

Delta = 0.8125 – 0.15 = 0.6625

Nilai delta tersebut kemudian di threshold sesuai kebutuhan kita. Semakin tinggi nilai delta (==1) maka akan semakin cocok fitur tersebut sebagai edge. Jika dimisalkan diberi nilai Threshold = 0.5, sedangkan nilai delta adalah 0.6625, maka fitur tersebut dianggap sebagai edge.

Menghitung nilai delta untuk keseluruhan citra akan membuat komputasi menjadi sangat mahal. Algoritma Viola-Jones mengajukan metode integral image.

31	2	4	33	5	36
12	26	9	10	29	25
13	17	21	22	20	18
24	23	15	16	14	19
30	8	28	27	11	7
1	35	34	3	32	6

31	33	37	70	75	111
43	71	84	127	161	222
56	101	135	200	254	333
80	148	197	278	346	444
110	186	263	371	450	555
111	222	333	444	555	666

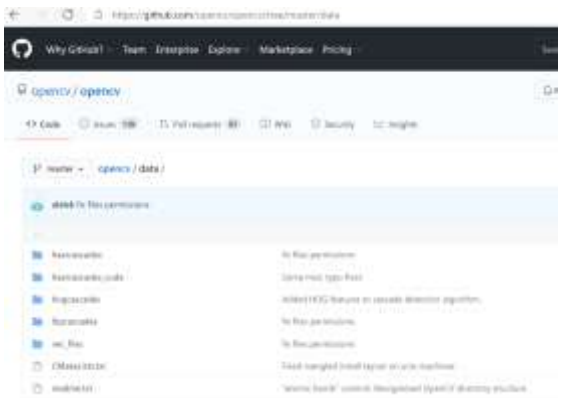
15 + 16 + 14 + 28 + 27 + 11 =

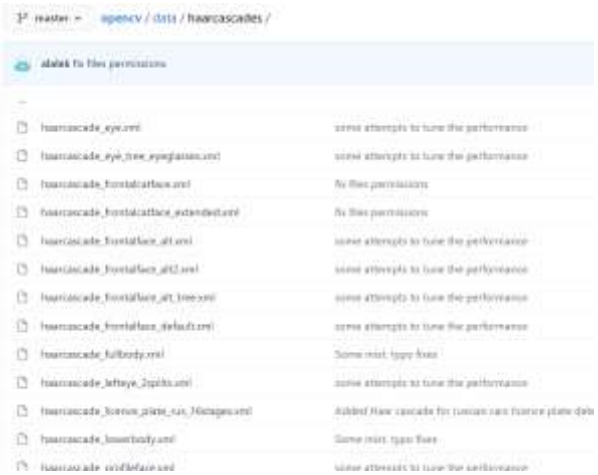
101 + 450 – 254 – 186 = 111

Algoritma ini juga dijalankan dengan cepat dengan menggunakan **cascade classifier**. **Cascade classifier** adalah menghitung nilai fitur image secara seri berdasarkan pada fitur sederhana yang telah ditunjukkan sebelumnya. Misalkan dilakukan ekstraksi fitur edge terlebih dahulu, dan jika tidak termasuk sebagai fitur edge, maka fitur line dan rectangle tidak dilakukan dan bergeser ke pencarian berikutnya.

Langkah awal yang dilakukan adalah dengan mengubah citra masukan ke dalam Grayscale. Fitur awal yang digunakan sebagai pencarian adalah edge feature yang menunjukkan adanya mata dan pipi. Jika diimage tidak menunjukkan adanya fitur ini, dianggap bahwa tidak ada wajah pada image tersebut. Jika terindikasi ada, maka dilanjutkan dengan mencari fitur berikutnya, seperti nose bridge (bagian tengah hidung),

lips (bibir), alis. Pencarian terus dilakukan menggunakan cascading, yang berarti akan terdapat ribuan fitur yang terkumpul. Secara teori, pendekatan ini dapat digunakan pada berbagai objek dan deteksi. Sebagai contoh, dapat digunakan untuk mendeteksi mata, bibir, hidung dan lain sebagainya. Atau dapat juga digunakan untuk mendeteksi objek lain seperti mobil, motor, pesawat, dan lain sebagainya. Kelemahan dari algoritma ini adalah kebutuhan akan referensi dataset yang besar untuk mendapatkan fitur yang diinginkan. Saat ini fitur-fitur deteksi objek menggunakan Viola-Jones dapat ditemukan dengan mudah. Pada OpenCV juga disediakan fitur-fitur tertentu yang sering digunakan dan dipake oleh banyak peneliti. Fitur-fitur tersebut disediakan OpenCV dengan format XML.





Percobaan berikut akan digunakan untuk melakukan face detection menggunakan OpenCV dan algoritma Viola Jones. Dokumentasi lengkap dapat dilihat pada OpenCV Documentation:
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html
Beberapa images telah disediakan pada folder (/images/facedet). Untuk pretrained features juga telah disediakan pada folder (/images/haarcascades). Silahkan akses secara langsung pada folder tersebut. Fitur pretrained yang digunakan adalah

```
cascade_wajah = cv.CascadeClassifier('/content/drive/MyDrive/Polinema/Kuliah/PCVK/Images/haarcascades/haarcascade_frontalface_alt.xml')
#sesuai posisi folder Haarcascade masing-masing

prabowo = cv.imread('/content/drive/MyDrive/5 Perkuliahan JTI/Materi Perkuliahan/MK PCVK/PCVK 2025/Images/facedet/prabowo.jpg',0)
roi_wajah = cascade_wajah.detectMultiScale(prabowo)
for(x,y,w,h) in roi_wajah:
  cv.rectangle(prabowo,(x,y),(x+w,y+h),(255,255,255),1)
plt.figure(figsize = (10,10))
plt.imshow(prabowo, cmap='gray')
```

“haarcascade_frontalface_alt.xml”. fungsi OpenCV yang digunakan adalah menggunakan “detectMultiScale”. Jika ditemukan wajah, maka akan digambarkan rectangle pada lokasi wajah tersebut menggunakan “rectangle”.

Face Tracking menggunakan live camera sebenarnya juga mudah sekali dilakukan pada Python dengan OpenCV, sayang sekali untuk Google Colab tidak bisa secara langsung menggunakan Camera yang ada pada komputer personal atau laptop, sehingga dibutuhkan code JavaScript untuk dapat mengakses kamera. code snippet untuk akses kamera telah tersedia juga pada Menu kiri Google Colab.

D. TUGAS PRAKTIKUM

Catatan: Untuk gambar pada praktikum ini menggunakan gambar pada link berikut: https://drive.google.com/drive/folders/1M30vcqgPBD1PHQpqFUtxWdtiMrwSpijC?usp=drive_link

D1. Praktikum Deteksi Obyek Dasar

1. Buka https://colab.research.google.com/. Setelah dipastikan bahwa google Colab terhubung dengan Github Anda, buat notebook baru dan beri nama "Week11.ipynb". Kemudian import beberapa library dan akses folder yang ada di Drive Anda dengan cara sebagai berikut.

2 Implementasikan 6 metode template matching pada OpenCV dengan menggunakan gambar cats_and_bunnies.jpg dan cat2_templatejpg.jpg sebagai templatenya.

Template Matching

Menggunakan library openCV:

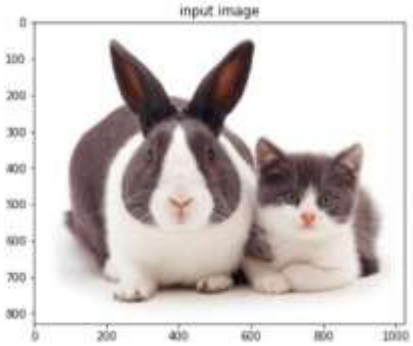
cv.matchTemplate(), dengan parameter:

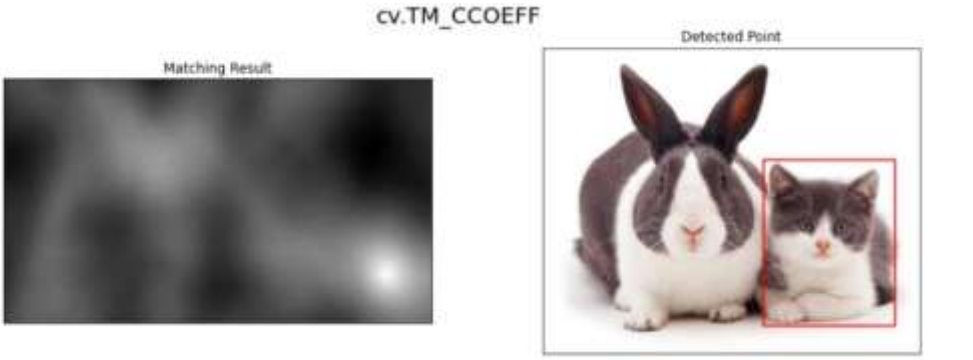
- **image**: citra input
- **templ**: template yang dicari, ukurannya tidak boleh lebih besar dari citra input
- **method**: metode dari template matching

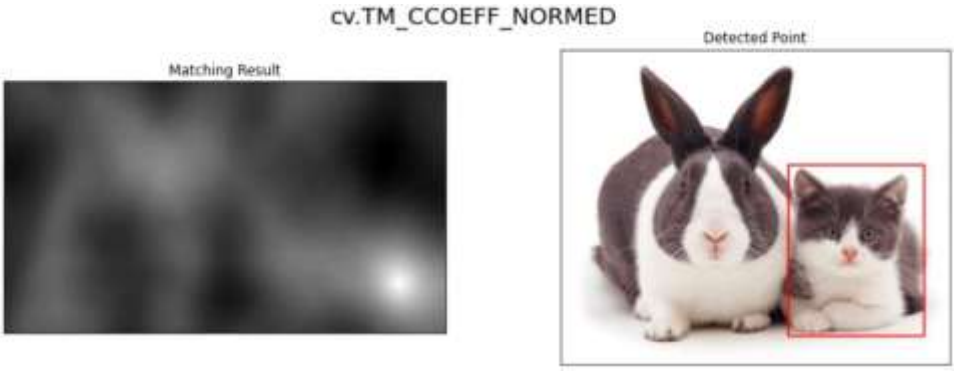
Jenis metode template matching di OpenCV:

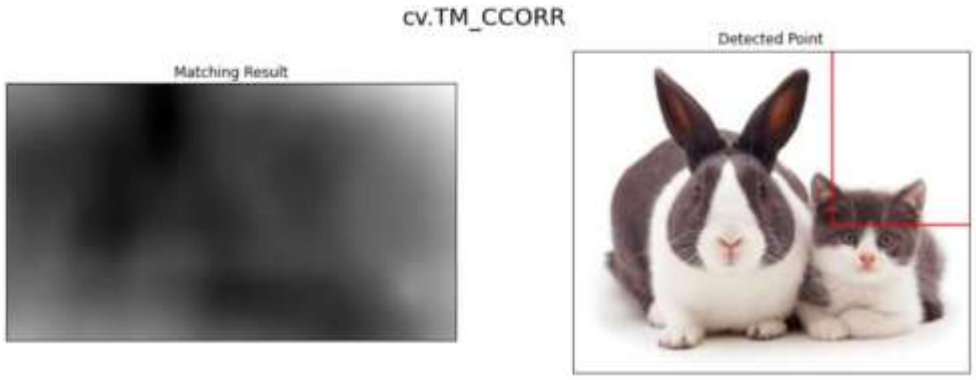
1. TM_SQDIFF
2. TM_SQDIFF_NORMED
3. TM_CCORR
4. TM_CCORR_NORMED
5. TM_CCOEFF
6. TM_CCOEFF_NORMED

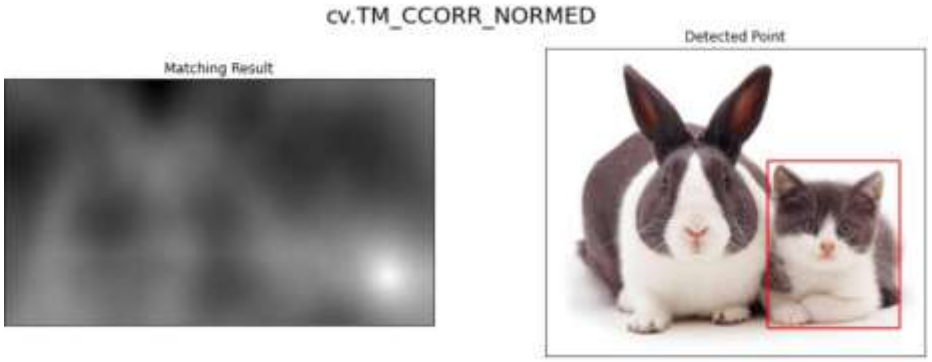
Sehingga menghasilkan luaran seperti berikut:

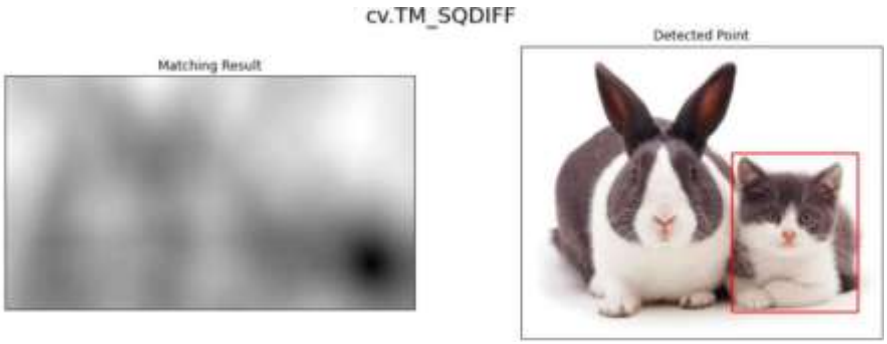


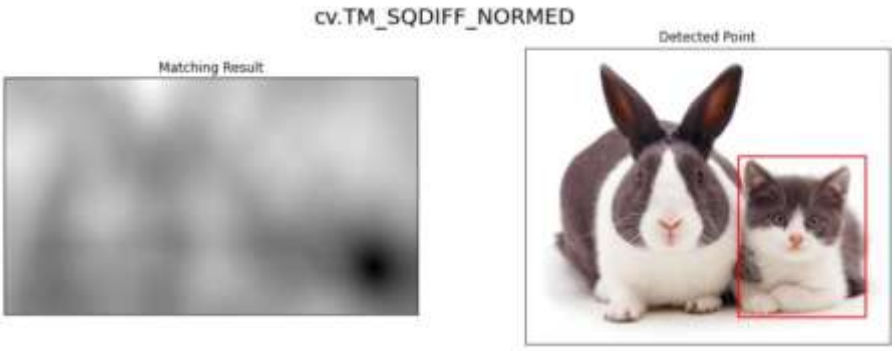




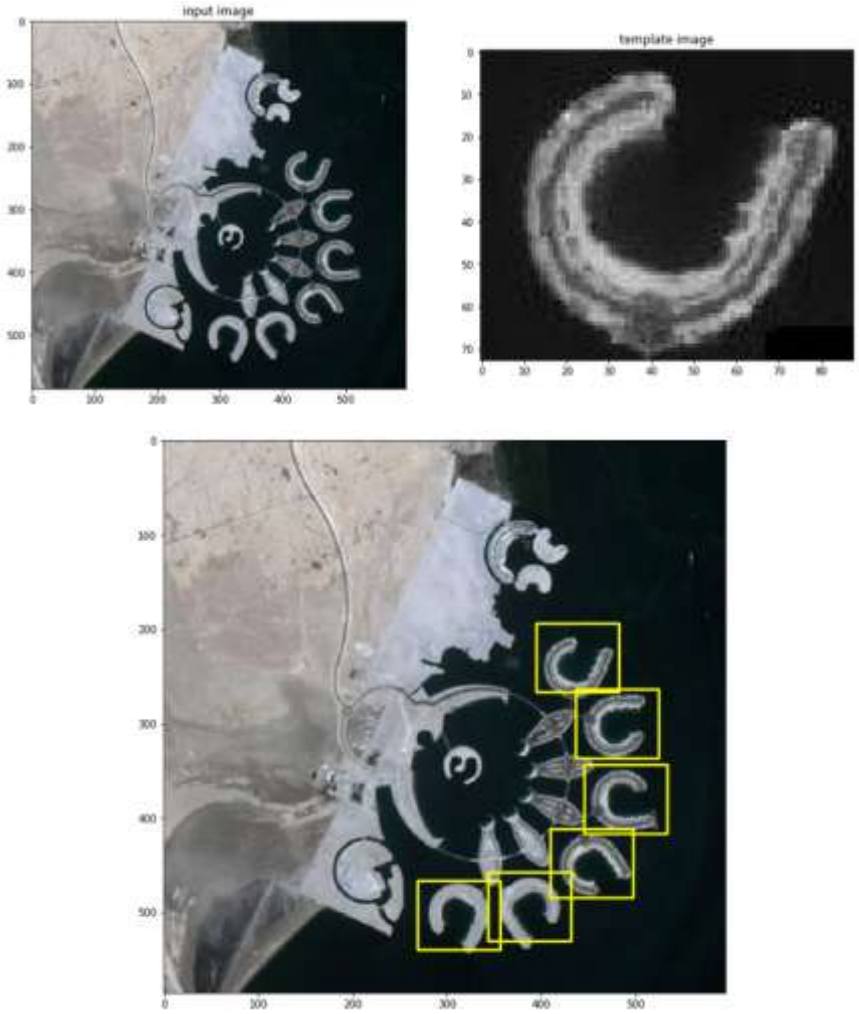






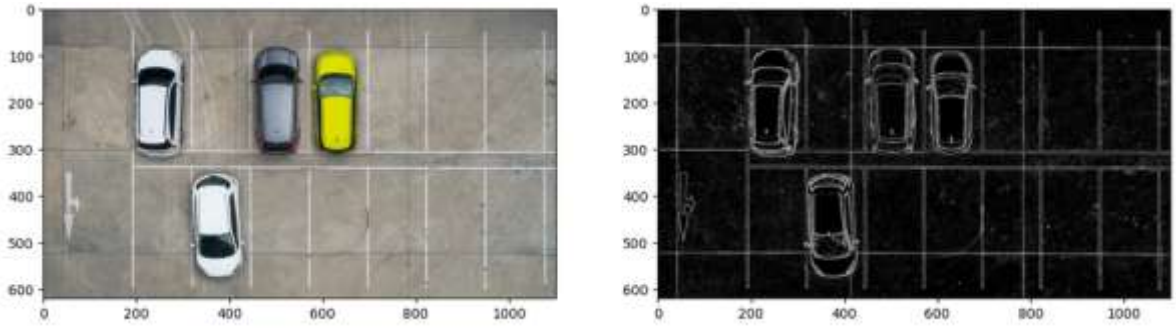


3 Implementasikan konsep template matching tanpa menggunakan library OpenCV untuk multiple object, menggunakan gambar bahrain.jpg untuk citra masukan dan bahrain-template.jpg sebagai citra template, sehingga menghasilkan output sebagai berikut:

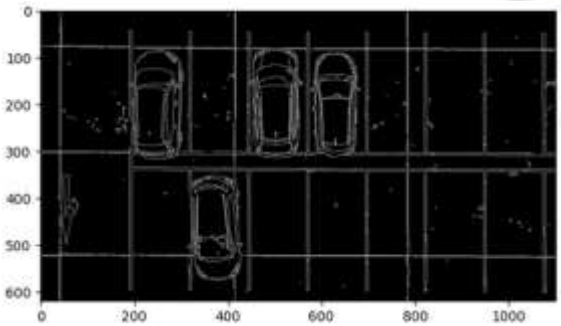


4 Implementasikan metode Sobel Edge Detection, Canny Edge Detection, dan Laplacian Edge Detection pada OpenCV dengan menggunakan gambar car-park.jpg, sehingga menghasilkan luaran sebagai berikut:

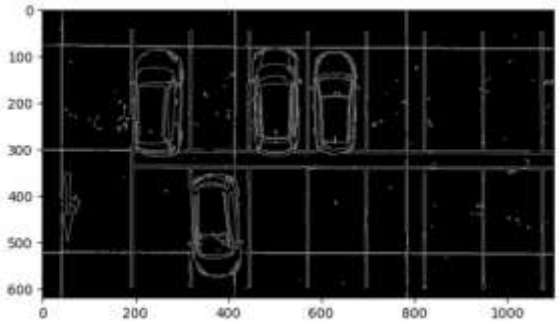
a. Sobel Edge Detection



b. Canny Edge Detection

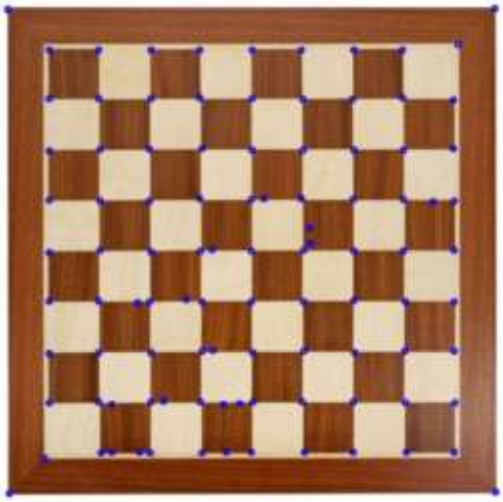
c. Laplacian Edge Detection

5 Implementasikan metode Harris Corner Detection dan Shi-Tomasi Detection pada OpenCV dengan menggunakan gambar chess-board.jpg, sehingga menghasilkan luaran sebagai berikut:

a. Harris Corner Detection

b. Shi-Tomasi Detection



6 Implementasikan metode Hough Transform pada OpenCV dengan menggunakan gambar sudoku.jpg. Tahapan proses grid detection sesuai yang terdapat pada ulasan teori, sehingga menghasilkan luaran sebagai berikut:

7. Implementasikan fungsi findContours() pada OpenCV untuk contour detection dengan menggunakan gambar laptop.jpg, sehingga menghasilkan luaran sebagai berikut:

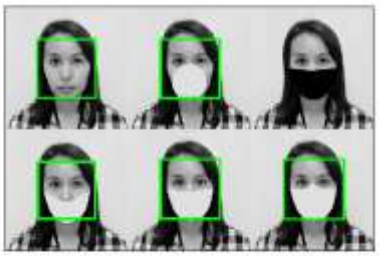
D2. Praktikum Feature Matching dan Face Detection

1. Lakukan Face Detection untuk image object lain yang tersedia pada (/images/facedet). Tampilkan seperti pada contoh berikut.





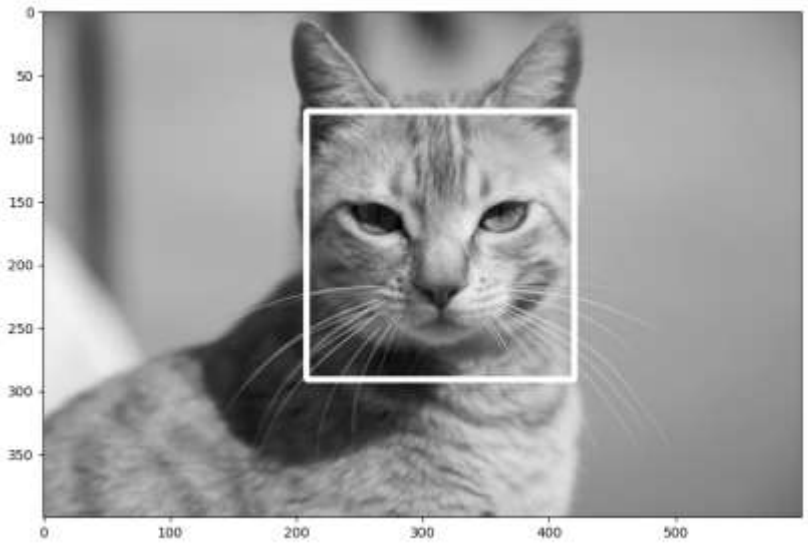




Perhatikan pada hasil face detection diatas. Secara keseluruhan, face detection dapat dilakukan dengan baik, bahkan untuk image berupa gambar bukan foto, wajah bermasker, atau wajah yang berukuran kecil (solvay).

2. Pada Soal No.2. wajah kucing tidak bisa dideteksi dengan baik. Lakukan deteksi wajah kucing hingga muncul rectangle pada bagian wajahnya. Petunjuk pada soal ini, perhatikan pretrained features yang telah disediakan OpenCV. Gunakan xml yang ada jika memang telah disediakan. Jika belum ada, coba cari dengan searching melalui search engines.



3. Cobakan juga untuk eyes detection pada wajah Pak Prabowo:

4. Lakukan deteksi senyuman pada gambar people:

5. Lakukan Face Tracking menggunakan Google Colab. Petunjuk, Tutorial selengkapnya tentang akses kamera dan FaceDetection pada google colab dapat dilihat di link berikut: https://www.youtube.com/watch?v=YjWh7QvVH60
6. Lakukan Blurring pada bagian wajah yang terdeteksi. Berikut contoh keluarannya. Petunjuk: anda dapat menggunakan cv.medianBlur untuk melakukan Blurring

7. Mengolah gambar KTM masing-masing
    a. Berikut ini contoh KTM

b. Tampilkan Foto KTM seperti contoh berikut ini:

c. Lakukan Deteksi Obyek KTM pada gambar yang berisi KTM dengan barang lainnya dengan menggunakan **Konsep Feature Matching**

Tambahan Tugas Praktikum:

Tampilkan karakter-karakter pada KTM ini Deep Learning untuk melakukan pengenalan karakter. Sebelum melakukan training data, terlebih dahulu di siapkan data yang akan dilakukan training yaitu data image nomor angka 0-9. Untuk membuat data tersebut bisa gunakan image editor untuk dilakukan croping satu persatu. Kemudian selanjutnya gunakan source code di bawah ini untuk proses persiapan untuk training.

```
import os
import tqdm
import cv2
import random
import numpy as np
import pickle

# Direktori data training
DATADIR = "dataset/training"
dirs = []

training_data = []
width, height = 100, 100

# Looping direktori data training untuk diambil nama karakternya
for char_name in sorted(os.listdir(DATADIR)):
    dirs.append(char_name)

# Looping semua image data training untuk diubah menjadi array
for char_name in dirs:
    path = os.path.join(DATADIR, char_name)
    class_number = dirs.index(char_name)
    for img in tqdm(os.listdir(path)):
        try:
            img_array = cv2.imread(os.path.join(data_dir_testing, car,
char_image), cv2.IMREAD_ANYCOLOR)
            new_array = cv2.resize(img_array, (width, height))
            training_data.append([new_array, class_number])
        except Exception as e:
            pass

random.shuffle(training_data)
X = []
Y = []

for feature, label in training_data:
    X.append(feature)
    Y.append(label)

X = np.array(X).reshape(-1, width, height, 1)

# Tulis ke file pickle
pickle_out = open("X.pickle", "wb")
pickle.dump(X, pickle_out)
pickle_out.close()

pickle_out = open("Y.pickle", "wb")
pickle.dump(Y, pickle_out)
pickle_out.close()
```

Penjelasan kode di atas adalah sebagai berikut ini

i. Import terlebih dahulu beberapa yang paket-paket yang dibutuhkan, ada beberapa paket yang baru misalkan tqdm digunakan untuk meload data diikuti dengan progress bar, numpy merupakan sebuah paket yang digunakan untuk melakukan operasi-operasi matriks

atau array serta pickle adalah depedensi untuk menyimpan file untuk model data training.

ii. Looping data training yang kelak digunakan untuk melabeli hasil pengenalan. Looping semua file training untuk diubah ke dalam sebuah image array.

iii. Variabel X dan variabel Y digunakan untuk menyimpan label dan feature, label berisi karakter A-Z dan 0-9 sedangkan feture berisi data image array masing-masing label tersebut.

iv. Terakhir tulis isi variabel X dan variabel Y ke dalam sebuah file pickle. File pickle tersebut nanti akan diload ketika proses training data.

Lakukan Training Data

Code berikut digunakan untuk melakukan training data

```
import pickle
from keras.layers import Input, Conv2D, MaxPooling2D, Flatten,
Dense, ZeroPadding2D
from keras.models import Model
from keras.optimizers import Adam
from keras.utils.np_utils import to_categorical

# Load file pickle
pickle_in = open("X.pickle", "rb")
X = pickle.load(pickle_in)

pickle_in = open("Y.pickle", "rb")
Y = pickle.load(pickle_in)

Y = to_categorical(Y)
X = X / 255.0
width, height = 100, 100

# Input layer
inputs = Input(shape=(width, height, 1))
conv_layer = ZeroPadding2D(padding=(2, 2))(inputs)
conv_layer = Conv2D(16, (5, 5), strides=(1, 1),
activation='relu')(conv_layer)
conv_layer = MaxPooling2D((2, 2))(conv_layer)
conv_layer = Conv2D(32, (3, 3), strides=(1, 1),
activation='relu')(conv_layer)
conv_layer = Conv2D(32, (3, 3), strides=(1, 1),
activation='relu')(conv_layer)
conv_layer = MaxPooling2D((2, 2))(conv_layer)
conv_layer = Conv2D(64, (3, 3), strides=(1, 1),
activation='relu')(conv_layer)

flaten = Flatten()(conv_layer)

fc_layer = Dense(256, activation='relu')(flaten)
fc_layer = Dense(64, activation='relu')(fc_layer)

# Output layer
outputs = Dense(34, activation='softmax')(fc_layer)

adam = Adam(lr=0.0001)
model = Model(inputs=inputs, outputs=outputs)
model.compile(optimizer=adam, loss='categorical_crossentropy',
metrics=['accuracy'])

model.fit(X, Y, epochs=20, verbose=1)

model.save('anpr.model')
```

Dengan kode di atas kita akan membuat sebuah model, kode di atas menggunakan deep learning dengan arsitektur CNN(Convolutional Neural Network). Sebenarnya yang saya ketahui perbedaan mendasar arsitektur ini dengan neural network biasa adalah masalah feature extraction, feature merupakan sebuah ciri yang khas yang membedakan antara objek satu dengan objek lainnya. Sebagai contoh karakter A dan karakter B pada

pelat kendaraan tentunya memiliki ciri dengan bentuk yang berbeda. Feature extraction yang dimiliki CNN(Convolutional Neural Network) sudah disediakan, kita tinggal mengutak-utik arsitekturnya/parameter yang ada di dalamnya. Sedangkan neural network biasa kita harus mencari sendiri atau dilakukan secara manual untuk mencari feature tersebut. Di bawah ini adalah bagian kode yang digunakan untuk membuat atau membangun sebuah feature tersebut.

```
conv_layer = ZeroPadding2D(padding=(2, 2))(inputs)
conv_layer = Conv2D(16, (5, 5), strides=(1, 1),
activation='relu')(conv_layer)
conv_layer = MaxPooling2D((2, 2))(conv_layer)
conv_layer = Conv2D(32, (3, 3), strides=(1, 1),
activation='relu')(conv_layer)
conv_layer = Conv2D(32, (3, 3), strides=(1, 1),
activation='relu')(conv_layer)
conv_layer = MaxPooling2D((2, 2))(conv_layer)
conv_layer = Conv2D(64, (3, 3), strides=(1, 1),
activation='relu')(conv_layer)
```

Testing Data

Contoh kode berikut digunakan untuk melakukan testing pengenalan karakter

```
import os
import cv2
import tensorflow as tf
import numpy as np

data_dir_training = "dataset/training-bak"
data_dir_testing = "dataset/testing"
dirs = []
width, height = 100, 100

model = tf.keras.models.load_model("anpr.model")

for char_name in sorted(os.listdir(data_dir_training)):
    dirs.append(char_name)

for car in sorted(os.listdir(data_dir_testing)):
    temp = ""
    for char_img in sorted(os.listdir(os.path.join(data_dir_testing,
car))):
        img_array = cv2.imread(os.path.join(data_dir_testing, car,
char_img), cv2.IMREAD_ANYCOLOR)
        new_array = cv2.resize(img_array, (width, height))
        new_array = np.array(new_array).reshape(-1, width, height,
1)

        new_array = new_array / 255.0

        prediction = model.predict(new_array)
        temp += dirs[np.argmax(prediction[0])]

    print("folder name: {} no: {}".format(car, temp))
```

--- SEMANGAT BELAJAR ---