영상 처리 응용 사례(2)

담당교수: 김민기

Contents

1. 동전 인식 프로그램
2. 차량 번호 검출 및 인식 프로그램

3. 동전 인식

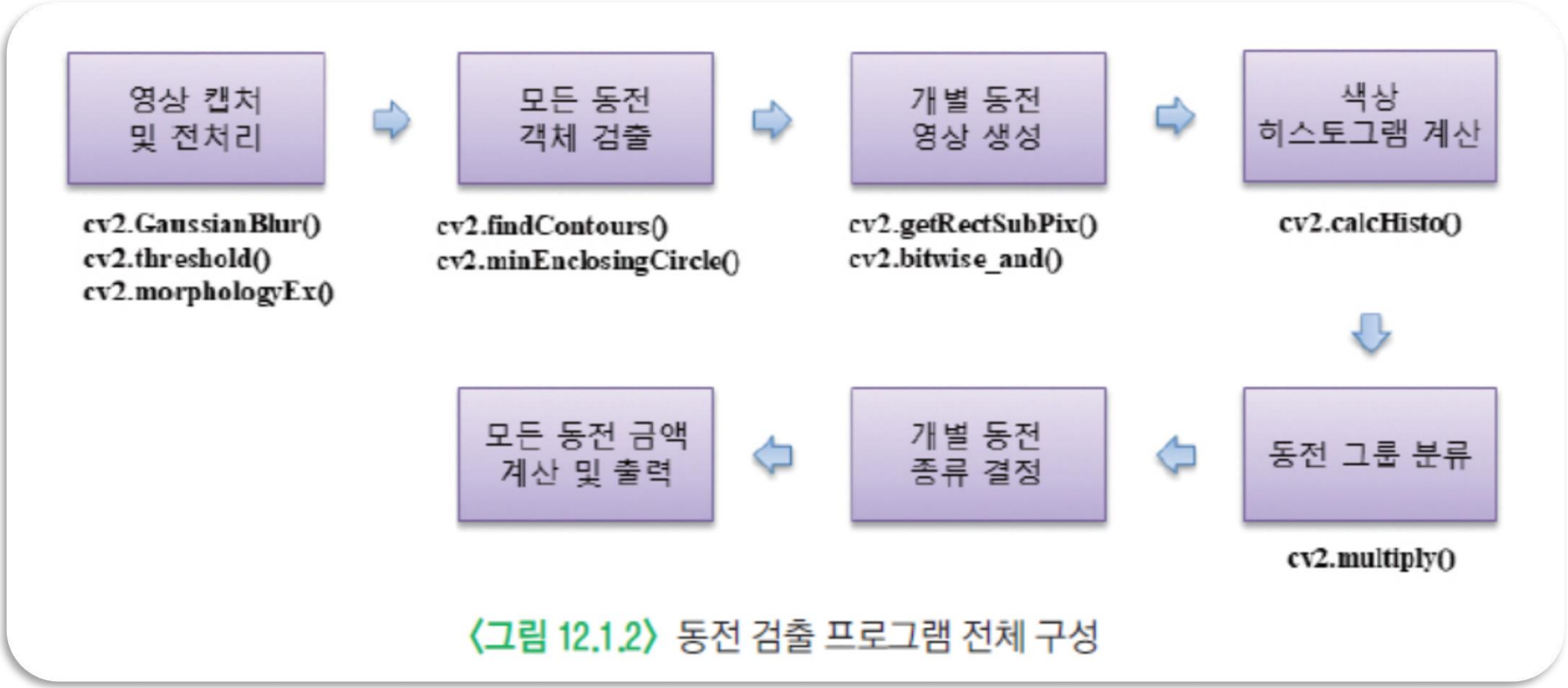
❖대량의 동전 계산 - 주화 계수기

❖일상생활에서 동전 계산

- 영상 처리를 이용하여 소량의 동전들 인식 → 금액 계산 가능

동전 인식 및 계산 시스템

❖동전 인식 및 계산 시스템 전체 구성



〈그림 12.1.2〉 동전 검출 프로그램 전체 구성

동전 영상 캡처

❖동전 캡처 시스템의 구성

- 동전을 놓는 받침대
- USB 카메라
- 카메라 고정하는 스탠드

❖캡처된 동전 영상을 파일로 저장

동전 영상 파일 폴더
- 교재예제/chap12/image/coin/

전처리

- 명도 영상에 대한 가우시안 블러링 수행
- 이진 영상에 대한 모폴로지 열림 연산

```
def preprocessing(coin_no):                                    # 전처리 함수
    fname = "images/coin/{0:02d}.png".format(coin_no)
    image = cv2.imread(fname, cv2.IMREAD_COLOR)                # 영상 읽기
    if image is None: reutrn None, None                        # 예외처리는 메인에서

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)             # 명암도 영상 변환
    gray = cv2.GaussianBlur(gray, (7, 7), 2, 2)                # 블러링
    flag = cv2.THRESH_BINARY + cv2.THRESH_OTSU                 # 오츠(otsu) 이진화 지정
    _, th_img = cv2.threshold(gray, 130, 255, flag)            # 이진화

    mask = np.ones((3, 3), np.uint8)
    th_img = cv2.morphologyEx(th_img, cv2.MORPH_OPEN, mask)    # 열림 연산
    reutrn image, th_img
```

동전 검출

❖동전 검출 과정

- 낱개의 동전 객체 인식 → 중심 좌표와 반지름 계산
- find_coins() 함수에서 구현
  - cv::findContours() 함수 사용해 외곽선 검출
  - cv::minAreaRect() 함수 사용해 동전 최소 영역 검출

```
01  def find_coins(image):
02      results= cv2.findContours(image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
03      contours = results[0] if int(cv2.__version__[0]) >= 4 else results[1]
04
05      ## 반복문 방식
06      # circles = []
07      # for contour in contours:
08      #     center, radius = cv2.minEnclosingCircle(contour)          # 외각 감싸는 원 검출
09      #     circle = (tuple(map(int, center)), int(radius))           # 소스 최적화 위해
10      #     if radius>25: circles.append(circle)
11
12      ## 리스트 생성 방식
13      circles = [cv2.minEnclosingCircle(c) for c in contours]     # 외각 감싸는 원 검출
14      circles = [(tuple(map(int, center)), int(radius))
15                 for center, radius in circles if radius>25]
16      reutrn circles
```

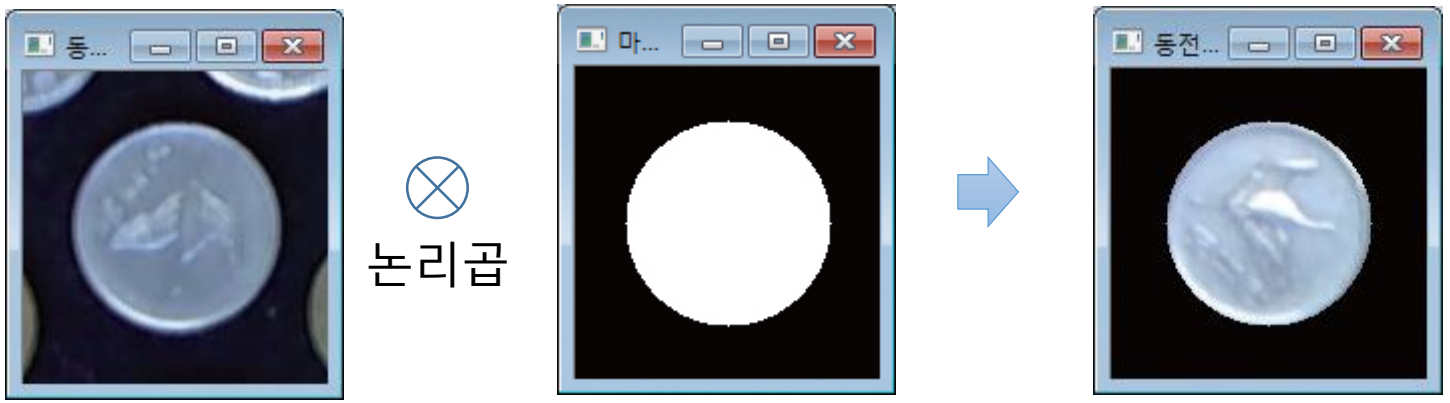
예제 12.1.1 동전 객체 검출 – 01.find_coins.py

```
01  from coin_preprocess import *                          # 전처리 함수 임포트
02
03  image, th_img = preprocessing(70)                      # 전처리 수행
04  if image is None: raise Exception("영상파일 읽기 에러")
05
06  circles = find_coins(th_img)                           # 동전 객체 검출
07  for center, radius in circles:
08        cv2.circle(image, center, radius, (0, 255, 0), 2)  # 영상에 검출 원 표시
09
10  cv2.imshow("preprocessed image", th_img)
11  cv2.imshow("coin image", image)
12  cv2.waitKey(0)
```

개별 동전 영상 생성

❖개별 동전영상 가져오기

- cv::getRectSubPix() 함수
  - 입력영상에서 지정된 좌표를 중심으로 크기만큼 영상 가져옴
  - 중심점 기준으로 특정 크기의 영역을 가져오기 편리함
- 개별 동전영상
  - 주위에 다른 동전 포함 및 주위 잡음 존재 가능
  - 원형 마스크를 통한 논리곱(cv::bitwise_and()) 연산으로 주위 배경 제거 가능



```python
def make_coin_img(src, circles):
    coins = []
    for center, radius in circles:
        r = radius * 3                                  # 검출 동전 반지름 3배
        cen = (r // 2, r // 2)                          # 마스크 중심
        mask = np.zeros((r, r, 3), np.uint8)            # 마스크 행렬
        cv2.circle(mask, cen, radius, (255, 255, 255), cv2.FILLED)
        # cv2.imshow("mask_" + str(center), mask)       # 마스크 영상 보기

        coin = cv2.getRectSubPix(src, (r, r), center)   # 동전 영상 가져오기
        coin = cv2.bitwise_and(coin, mask)              # 마스킹 처리
        coins.append(coin)                              # 동전 영상 저장
    reutrn coins
```

색상 히스토그램 계산

❖동전 인식

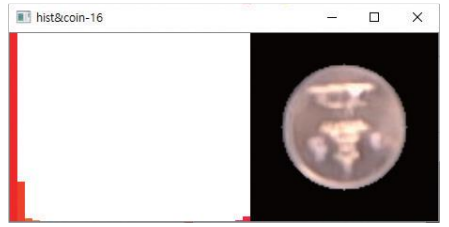
- 일반적으로 동전 객체의 크기 이용
- 동전객체 검출과정에서 해당 동전이 완벽하게 검출되지 않을 수 있음
  - 크기 비슷한 금액 오인식 가능성 높음

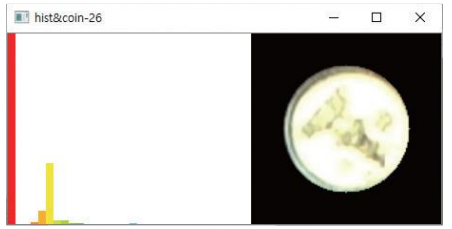
    10원과 50원 / 10원과 100원

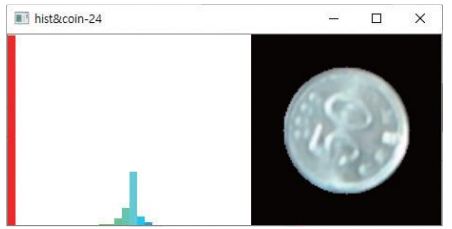
- 10원과 다른 동전 색상 차이 확실함
  - 색상을 동전 분류에 활용
  - 색상 히스토그램 이용

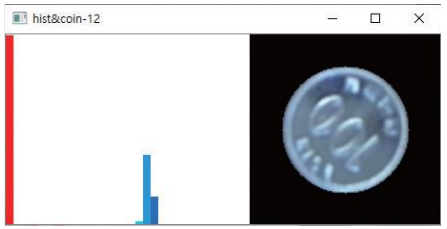
예제 12.1.2 동전 객체 히스토그램 그리기 – 02.draw_coin_histo.py

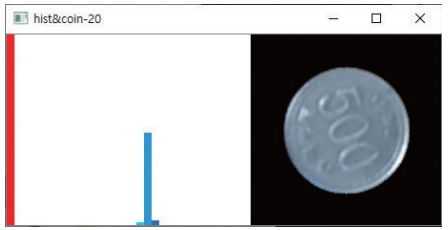
```
01  from coin_preprocess import *
02  from coin_utils import *                          # 모든 함수 임포트
03  from Common.histogram import draw_histo_hue       # 색상 히스토그램 생성 함수
04
05  coin_no = 15                                      # 동전 번호
06  image, th_img = preprocessing(coin_no)            # 전처리 수행
07  circles = find_coins(th_img)                      # 동전 객체 검출
08  coin_imgs = make_coin_img(image, circles)                 # 동전 영상 분리
09  coin_hists = [calc_histo_hue(coin) for coin in coin_imgs]  # 색상 히스토그램
10
11  for i, img in enumerate(coin_imgs):
12      h, w = 200, 256                                       # 히스토그램 영상 크기
13      hist_img = draw_hist_hue(coin_hists[i], (h, w, 3))    # 색상 히스토그램 표시
14
15      merge = np.zeros((h, w+h, 3), np.uint8)
16      merge[:, :w] = hist_img                       # 결과 행렬 왼쪽 – 히스토그램 영상
17      merge[:, w:] = cv2.resize(img, (h, h))        # 결과 행렬 오른쪽 – 동전 영상
18      cv2.imshow("hist&coin - " + str(i), merge)
19
20  cv2.waitKey(0)
```







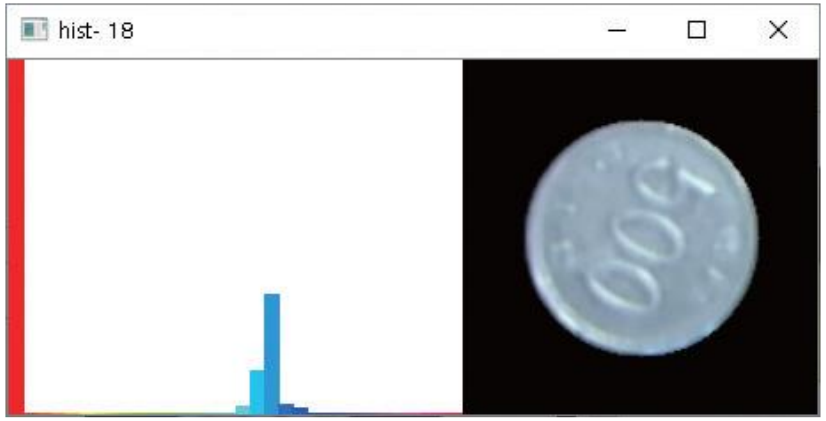




동전 그룹 분류

❖동전 인식

- 색상 히스토그램 이용하여 동전을 두 그룹으로 분류
  - 10원 동전 vs. 50원, 100원 500원 동전



개별 동전 종류 결정

❖동전 객체 반지름으로 동전 인식

- 분류된 그룹에 따라 다른 반지름 적용

조명 등의 영향으로
500, 100 동전도 그룹0로 분류될 수 있음

표 12.1.1 동전 객체의 반지름에 따른 동전 분류

동전 종류		동전 반지름
그룹 0 브론즈 그룹 10원	500 원	48< raduis ≤ 55
	100 원	46< raduis ≤ 48
	10 원	25< raduis ≤ 46
그룹 1 실버 그룹 500, 100, 50원	500 원	48< raduis ≤ 55
	100 원	43< raduis ≤ 48
	50 원	36< raduis ≤ 43
	10 원	25< raduis ≤ 35

예제 12.1.3 동전 계산 프로그램 완성 – calc_coins.py

```
01  from coin_preprocess import *
02  from coin_utils import *                          # 관련 함수 임포트
03  from Common.utlis import put_string               # 문자열 표시 함수 임포트
04
05  coin_no = int(input("동전 영상 번호: "))
06  image, th_img = preprocessing(coin_no)            # 전처리 수행
07  circles = find_coins(th_img)                      # 객체(중심점, 반지름) 검출
08  coin_imgs = make_coin_img(image, circles)         # 개별 동전 영상 생성
09
10  coin_hists= [calc_histo_hue(coin) for coin in coin_imgs]
11  groups = grouping(coin_hists)                     # 동전
12
13  ncoins = classify_coins(circles, groups)          # 동전
14  coin_value = np.array([10, 50, 100, 500])         # 동전
15  for i in range(4):
16      print("%3d원: %3d개" % (coin_value[i], ncoins[i]))  # 동전
17
18  total = sum(coin_value * ncoins)                  # 동전 금액 * 동전별 개수
19  str = "Total coin: {:,} Won".format(total)        # 계산 금액 문자열
20  print(str)                                        # 실행창에 출력
21  put_string(image, str, (10, 50), '', (0, 230, 0))  # 영상에 출력
22
23  ## 동전 객체에 정보(반지름, 금액) 표시
24  color = [(0, 0, 250), (255, 255, 0), (0, 250, 0), (250, 0, 255)]   # 동전별 색상
25  for i, (c, r) in enumerate(circles):
26      cv2.circle(image, c, r, color[groups[i]], 2)
27      put_string(image, i, (c[0] - 15, c[1] - 10), '', color[2])   # 검출 순번
28      put_string(image, r, (c[0], c[1] + 15), '', color[3])        # 동전 반지름
29
30  cv2.imshow("result image", image)
31  cv2.waitKey(0)
```

```
Run:  03.calc_coins
C:\Python\python.exe D:/source/chap12/03.calc_coins.py
동전 영상 번호: 24
 10원:  15개
 50원:   1개
100원:   6개
500원:   5개
Total coin: 3,300 Won
```

4. 차량 번호 검출 및 인식

❖차량 번호 인식(LPR: License Plate Recognition) 시스템
- 주차장에 진입 차량의 번호판 자동 인식 및 식별
- 차량과 주차 관리를 제어하는 시스템

❖차량 번호 인식 프로그램 두 단계 과정
- 입력 영상에서 번호판 영역 검출 단계
- 검출된 번호판 영역에서 숫자나 문자 인식하는 단계

❖본 응용 예제: 자동차 번호판 영역 검출
- 기계 학습 알고리즘 중의 하나인 SVM(Support Vector Machine) 이용

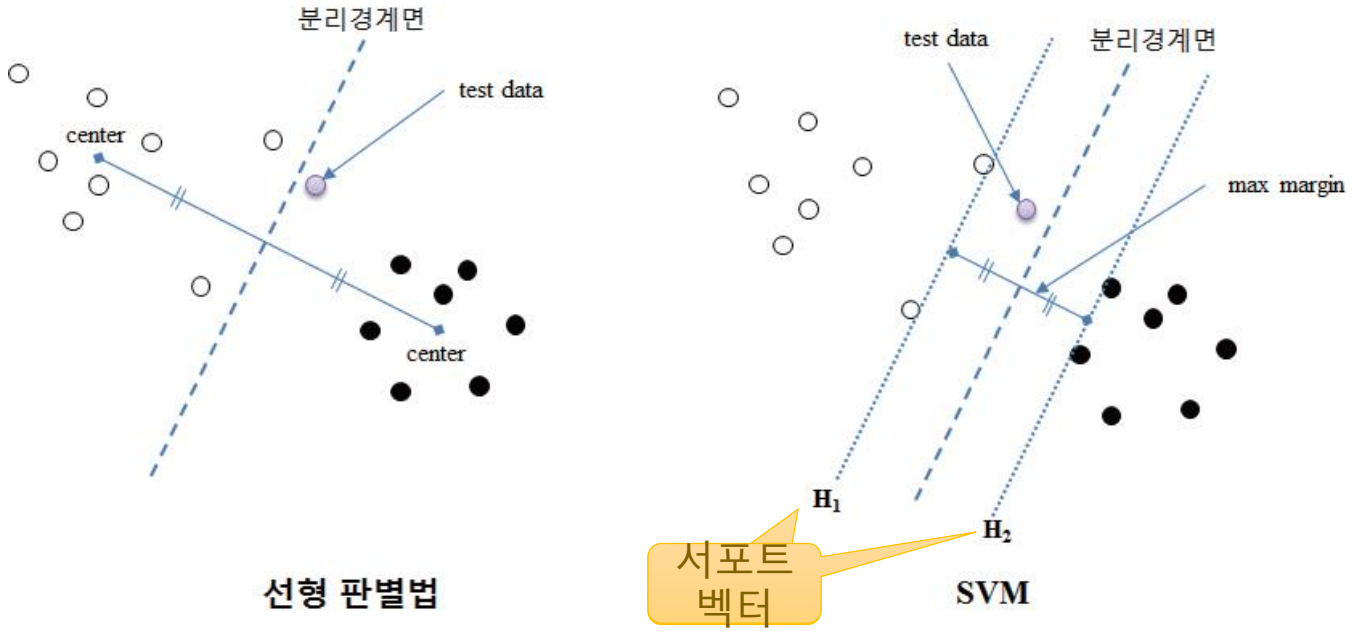
SVM의 개념

❖선형 판별법

- 각 그룹 내에서 데이터 간 거리 측정 → 중심점 계산
- 두 중심점의 중간에 최적 분리 경계면 → 이 경계를 기준으로 새로운 데이터 분류 수행

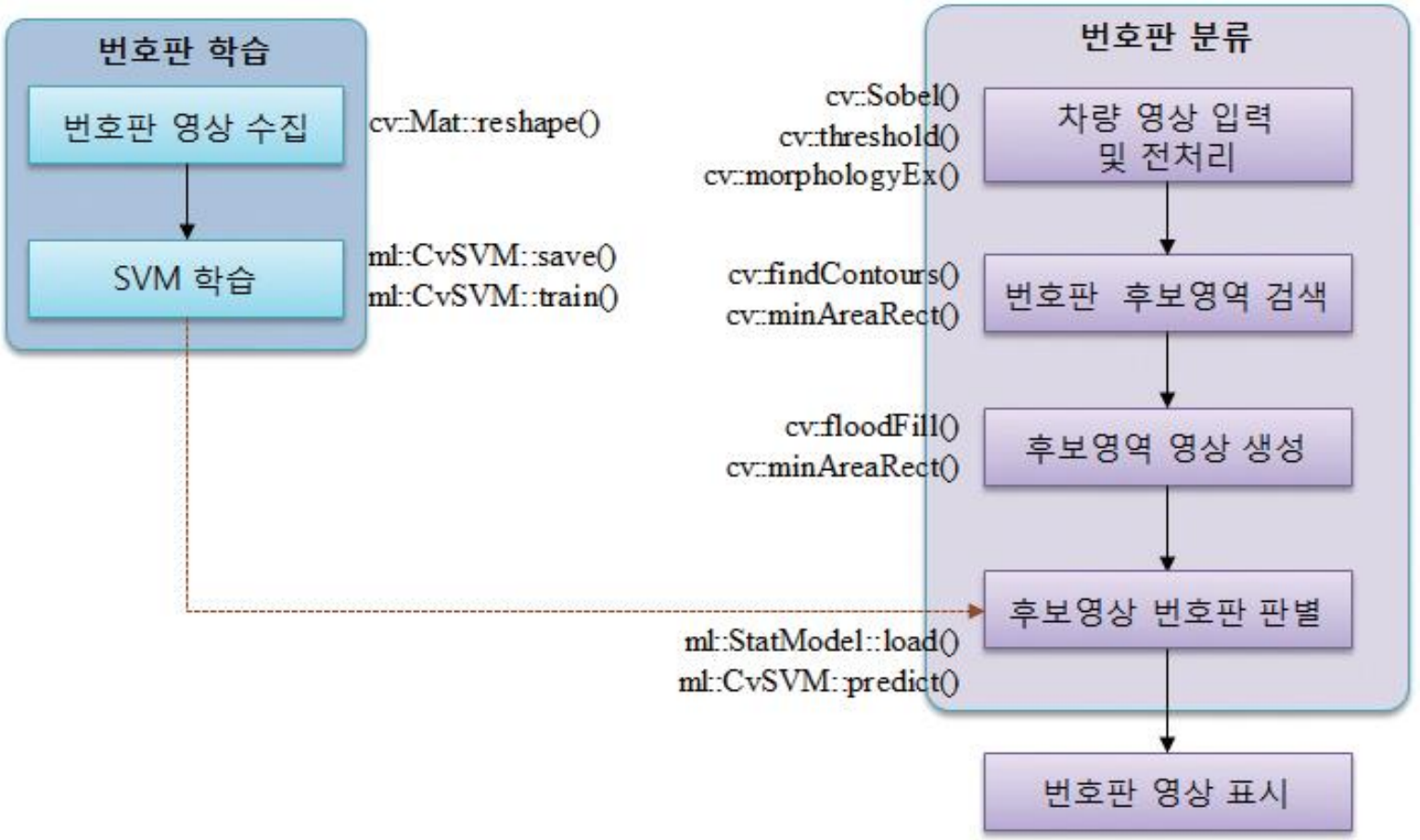
❖SVM

- 데이터들 분리하는 분리 경계면 중 분류 데이터들과의 거리가 가장 먼 분리 경계면 찾음
- 분리 경계와 실제 데이터들 사이의 `margin'이 가장 크도록 분리 경계를 설정하는 것



번호판 검출 프로그램의 전체 과정

❖번호판 학습 모듈과 번호판 분류 모듈로 구성



번호판 영상 학습

❖번호판 영상의 수집

- SVM을 수행하려면 먼저 학습 데이터로 학습을 수행해야 함.
  - 학습 데이터 - 번호판 영상과 번호판 아닌 영상

- 일반 차량 번호 공개 불가
  - 학습 데이터 xml 파일로 제공 : SVMDATA.xml
  - 직접 학습데이터를 만들고 싶다면

    번호판 영상은 직접 촬영해서 000.png~069.png로 ../image/plates 폴더에 저장

    번호판 아닌 영상은 070.png~139.png로 ../image/plates 폴더에 저장

    학습 영상의 크기는 144x28

예제 12.2.1 번호판 영상 수집 및 학습 – 04.collect_carimage.py

```
01  import numpy as np, cv2
02  from Common.functions import imread
03
04  def SVM_create(type, max_iter, epsilon):
05        svm = cv2.ml.SVM_create()                          # SVM 객체 선언
06        ## SVM 파라미터 지정
07        svm.setType(cv2.ml.SVM_C_SVC)                      # C-Support Vector Classification
08        svm.setKernel(cv2.ml.SVM_LINEAR)                   # 선형 SVM
09        svm.setGamma(1)                                    # 커널 함수의 감마값
10        svm.setC(1)                                        # 최적화를 위한 C 파라미터
11        svm.setTermCriteria((type, max_iter, epsilon))     # 학습 반복 조건 지정
12        reutrn svm
13
14  nsample = 140                                            # 학습 영상 총 개수
15  trainData = [cv2.imread("images/plate/%03d.png" %i, 0) for i in range(nsample)]
16  trainData = np.reshape(trainData, (nsample, -1)).astype('float32')
17  labels = np.zeros((nsample, 1), np.int32)                # 레이블 행렬
18  labels[:70] = 1                                          # 번호판 레이블 번호
19
20  print("SVM 객체 생성")
21  svm = SVM_create(cv2.TERM_CRITERIA_MAX_ITER, 1000, 1e-6)      # SVM 객체 생성
22  svm.train(trainData, cv2.ml.ROW_SAMPLE, labels)               # 학습 수행
23  svm.save("SVMtrain.xml")                                      # 학습된 데이터 저장
24  print("SVM 객체 저장 완료")
```

❖ 전처리 함수 구현 (./header/plate_preprocess.py)
- ✓ 명암도 영상 변환, 블러링 및 소벨 에지 검출
- ✓ 이진화 및 모폴로지 열림 연산 수행

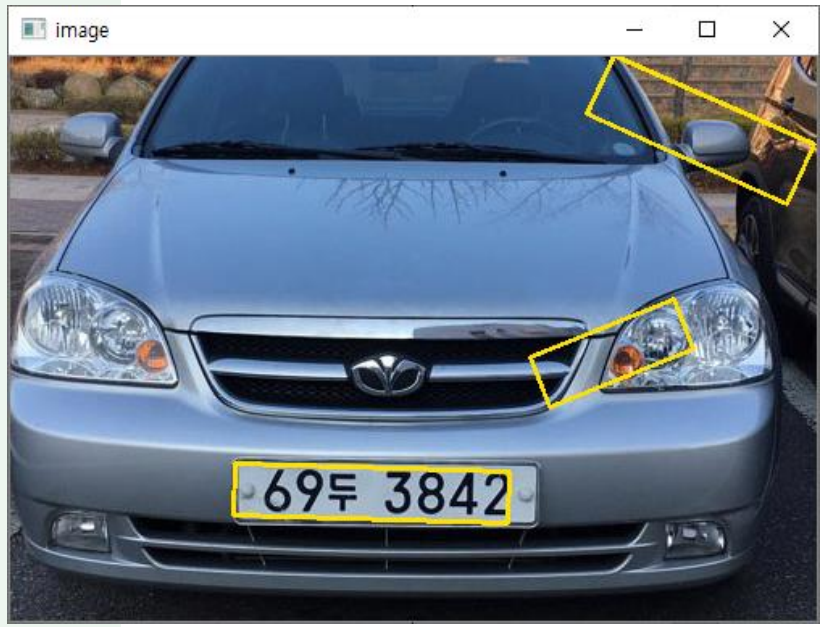
```
def preprocessing(car_no):
    image = cv2.imread("images/car/%02d.jpg" %car_no, cv2.IMREAD_COLOR)
    if image is None: return None, None

    kernel = np.ones((5, 17), np.uint8)                    # 닫힘 연산 마스크(커널)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)         # 명암도 영상 변환
    gray = cv2.blur(gray, (5, 5))                          # 블러링
    gray = cv2.Sobel(gray, cv2.CV_8U, 1, 0, 3)             # 수직 에지 검출

    th_img = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY)[1]        # 이진화 수행
    morph = cv2.morphologyEx(th_img, cv2.MORPH_CLOSE, kernel, iterations=3)

    #cv2.imshow("th_img", th_img); cv2.imshow("morph", morph)       # 결과표시
    reutrn image, morph
```

❖ 번호판 후보영역 판정 함수 (./header/plate_preprocess.py)

```
def verify_aspect_size(size):
    w, h = size
    if h == 0 or w == 0: reutrn False

    aspect = h / w if h > w else w / h              # 세로가 길면 역수 취함
    chk1 = 3000 < (h*w) < 12000                    # 번호판 넓이 조건
    chk2 = 2.0 < aspect < 6.5                      # 번호판 종횡비 조건
    reutrn (chk1 and chk2)

def find_candidates(image):
    results = cv2.findContours(image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contours = results[0] if int(cv2.__version__[0]) >= 4 else results[1]

    rects = [cv2.minAreaRect(c) for c in contours]    # 회전 사각형 반환
    candidates = [(tuple(map(int, center)), tuple(map(int, size)), angle) # 정수형 변환
                  for center, size, angle in rects if verify_aspect_size(size)]
    reutrn candidates
```

예제 12.2.2 번호판 후보 영역 검색 – 05.find_plates.cpp

```
01  from plate_preprocess import *                          # 전처리 및 후보 영역 검사 함수
02
03  car_no = int(input("자동차 영상 번호(0~15): "))
04  image, morph = preprocessing(car_no)                    # 전처리– 소벨&열림 연산
05  if image is None: Exception("영상파일 읽기 에러")
06
07  candidates = find_candidates(morph)                     # 번호판 후보 영역 검색
08  for candidate in candidates:                            # 후보 영역 표시
09      pts = np.int32(cv2.boxPoints(candidate))
10      cv2.polylines(image, [pts], True, (0, 225,255), 2)  # 다중 좌표 잇기
11      print(candidate)
12
13  if not candidates:                                      # 리스트 원소가 없으면
14      print("번호판 후보 영역 미검출")
15  cv2.imshow("image", image)
16  cv2.waitKey(0)
```



번호판 후보 영역 영상 생성

❖ 번호판 후보영역 영상 생성 과정

- 후보영역 개선 → 후보영역 재검증 → 후보영상 회전 보정 → 후보영상 생성

- 후보영상 개선
  - 검출한 번호판 후보영역들은 이진영상으로 검출
  - 컬러 영상 이용해 번호판과 좀 더 유사한 영역으로 개선
  - refine_candidates() 함수로 구현

    cv::floodFill() 함수 – 영역 채움 함수

    cv::minAreaRect() 함수 – 회전 사각형의 최소영역 반환

❖ 후보영상 개선 함수 – 컬러 활용 (./header/plate_candidate.py)

```
01  def color_candidate_img(image, candi_center):
02        h, w = image.shape[:2]
03        fill = np.zeros((h + 2, w + 2), np.uint8)                    # 채움 행렬
04        dif1, dif2 = (25, 25, 25), (25, 25, 25)                      # 채움 색상 범위
05        flags = 0xff00 + 4 + cv2.FLOODFILL_FIXED_RANGE               # 채움 방향 및 방법
06        flags += cv2.FLOODFILL_MASK_ONLY                             # 결과 영상만 채움
07
08        ## 후보 영역을 유사 컬러로 채우기
09        pts = np.random.randint( -15, 15, (20, 2) )                  # 임의 좌표 20개 생성
10        pts = pts + center                                           # 중심 좌표로 평행이동
11        for x, y in pts:                                             # 임의 좌표 순회
12             if 0 <= x < w and 0 <= y < h:                           # 후보 영역 내부이면
13                  _, _, fill, _ = cv2.floodFill(image, fill, (x, y), 255, dif1, dif2, flags)  #채움누적
14
15        reutrn cv2.threshold(fill, 120, 255, cv2.THRESH_BINARY)[1]
```

번호판 후보 영역 영상 생성

- 후보영상 보정 함수 rotate_plate() 구현
  - 후보 영역은 회전 사각형으로 저장

    중심점과 회전 각도 포함

    각도만큼 역회전 → 후보 영상(번호판)을 수평 직사각형으로 배치 가능

  - 각도의 계산

    종횡비가 1 이상 → 수직 긴 사각형 → 90도 회전 필요

  - 회전보정

    cv::getRotationMatrix2D() 와 cv::warpAffine() 함수 사용하여 역회전 수행

❖ 후보영상 보정 함수 (./header/plate_candidate.py)

```
def rotate_plate(image, rect):
    center, (w, h), angle = rect                          # 중심 좌표, 크기, 회전 각도
    if w < h :                                            # 세로가 긴 영역이면
        w, h = h, w                                       # 가로와 세로 맞바꿈
    angle += 90                                           # 회전 각도 조정

    size = image.shape[1::-1]                             # 행태와 크기는 역
    rot_mat = cv2.getRotationMatrix2D(center, angle, 1)   # 회전 행렬 계산
    rot_img = cv2.warpAffine(image, rot_mat, size, cv2.INTER_CUBIC)      # 회전

    crop_img = cv2.getRectSubPix(rot_img, (w, h), center) # 후보 영역 가져오
    crop_img = cv2.cvtColor(crop_img, cv2.COLOR_BGR2GRAY) # 명암도 영상
    reutrn cv2.resize(crop_img, (144, 28))                # 크기변경 후 반환
```

번호판 후보 영상의 번호판 판별

❖번호판 판별 위해 SVM 분류 수행

- SVM_create() 함수 호출 - SVM 클래스 객체 생성
- SVM 객체를 포인터(→)로 접근해서 train() 함수 호출 → 학습 수행

- 후보영상 분류 수행 함수 classify_plates() 구현
  - 입력 인수: 학습을 수행한 svm 객체 , 번호판 후보영상들
  - 분류 수행 위해

    회전 보정된 번호판 후보영상을 1행 데이터로 변환

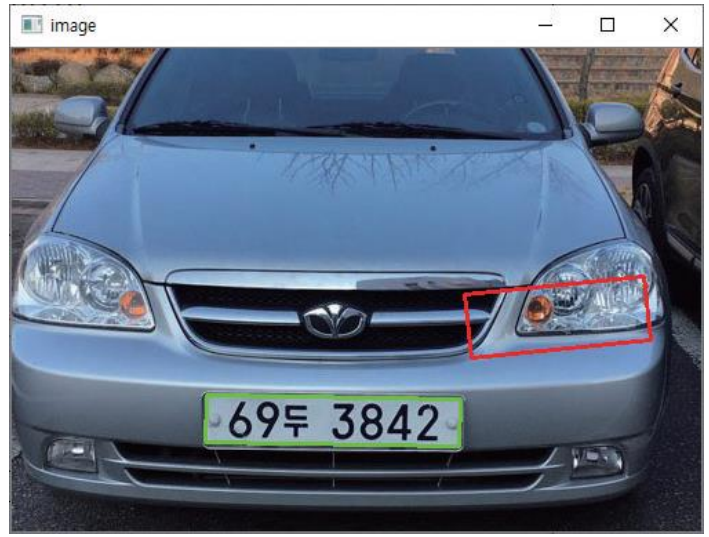
    자료형을 CV_32F로 변경
  - SVM 객체를 포인터(→)로 접근해서 predict() 함수 호출

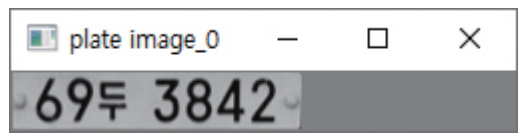
    → 분류 수행 (학습 및 분류를 위한 행렬 데이터는 1행(1열) 데이터 , float 형)

예제 12.2.4 번호판 인식 프로그램 완성 – 07.classify_plate.py

```
01  from plate_preprocess import *              # 전처리 및 후보 영역 검출 함수
02  from plate_candidate import *               # 후보 영역 개선 및 후보 영상 생성 함수
03
04  car_no = int(input("자동차 영상 번호(0~15): "))
05  image, morph = preprocessing(car_no)
06  candidates = find_candidates(morph)         # 번호판 후보 영역 검색
07
08  fills = [color_candidate_img(image, size) for size, _, _ in candidates]   # 후보 영역 재생성
09  new_candis = [find_candidates(fill) for fill in fills]          # 재생성 영역 검사
10  new_candis = [cand[0] for cand in new_candis if cand]           # 재후보 있으면 저장
11  candidate_img =[rotate_plate(image, cand) for cand in new_candis]   # 후보 영역 영상
12
13  svm = cv2.ml.SVM_load("SVMtrain.xml")                           # 학습된 데이터 적재
14  rows = np.reshape(candidate_imgs, (len(candidate_imgs), -1))    # 1행 데이터들로 변환
15  _, results = svm.predict(rows.astype('float32'))                # 분류 수행
16  correct = np.where(results == 1)[0]                             # 정답 인덱스 찾기
17
18  print("분류 결과:\n", results)
19  print("번호판 영상 인덱스:", correct )
20
```

```python
for i, idx in enumerate(correct):
    cv2.imshow("plate image_" + str(i), candidate_imgs[idx])    # 후보영역 영상 출력
    cv2.resizeWindow("plate image_" + str(i), (250,28))         # 윈도우 크기 조정

for i, candi in enumerate(new_candis):
    color = (0, 255, 0) if i in correct else (0, 0, 255)      # 후보영역 확인 및 색 지정
    cv2.polylines(image, [np.int32(cv2.boxPoints(candi))], True, color, 2) # 후보 영역 표시

print("번호판 검출완료") if len(correct)>0 else print("번호판 미검출")

cv2.imshow("image", image)
cv2.waitKey(0)
```





첫번째 후보 결과

두번째 후보 결과

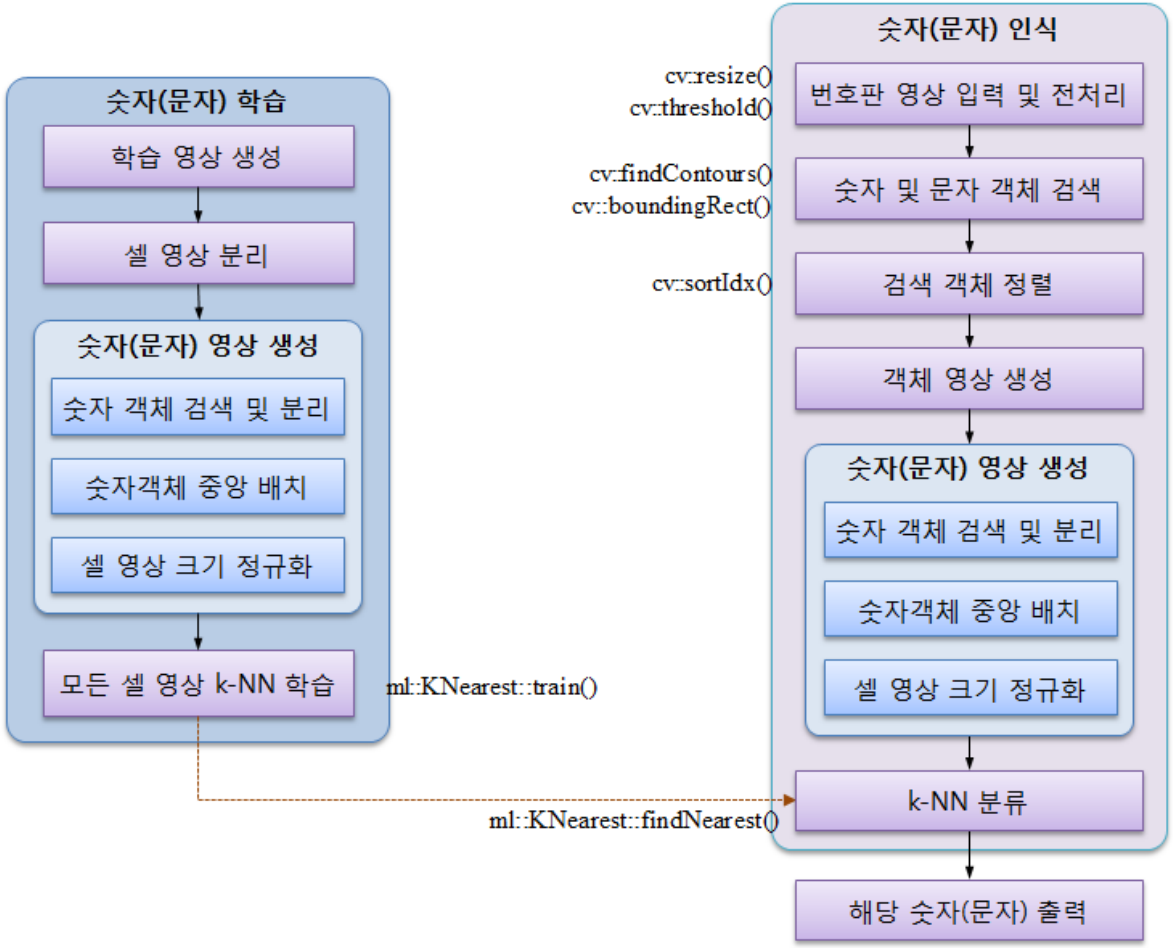
분류 결과가 1이면 번호판

```
Run: 07.classify_plates
C:\Python\python.exe D:/source/chap12/07.classify_plates.py
자동차 영상 번호 (0~15): 0
분류 결과:
 [[1.]
 [0.]]
번호판 영상 인덱스: [0]
번호판 검출완료
```

K-NN을 이용한 차량 번호 인식

❖차량 번호판 인식 프로그램 전체 과정

- 숫자(문자) 학습 모듈과 숫자(문자) 인식 모듈로 구성



❖ 숫자 및 문자 인식에 k-NN 알고리즘 사용 (./header/plate_classify.py)

- 예제_10.3.4에서 KNN_number.py 소스
  - 학습데이터 로드→ knn 객체 생성 → 학습 수행

```
def kNN_train(train_fname, K, nclass, nsample):
    size = (40, 40)                                          # 숫자 영상 크기
    train_img = cv2.imread(train_fname, cv2.IMREAD_GRAYSCALE)  # 학습 영상 읽기
    h, w = train_img.shape[:2]
    dx = w % size[0] // 2                                    # 좌우 여백 크기
    dy = h % size[1] // 2                                    # 상하 여백 크기
    train_img = train_img[dy:h-dy-1, dx:w-dx-1]               # 학습 영상 여백 제거
    cv2.threshold(train_img, 32, 255, cv2.THRESH_BINARY, train_img)
    cells = [np.hsplit(row, nsample) for row in np.vsplit(train_img, nclass)] # 셀 영상 분리
    nums = [find_number(c) for c in np.reshape(cells, (-1, 40, 40))]        # 숫자 객체 검출
    trainData = np.array([place_middle(n, size) for n in nums])             # 객체 중앙 배치
    labels = np.array([i for i in range(nclass) for j in range(nsample)], np.float32)

    knn = cv2.ml.KNearest_create()
    knn.train(trainData, cv2.ml.ROW_SAMPLE, labels)                  # k-NN 학습 수행
    reutrn knn
```

학습 영상

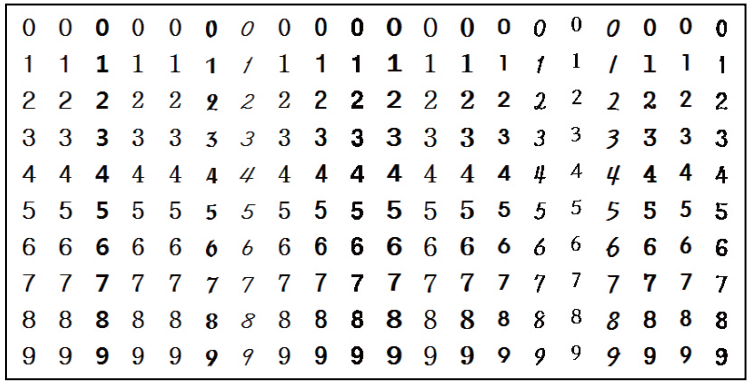
❖숫자 및 문자 인식에 k-NN 알고리즘 사용

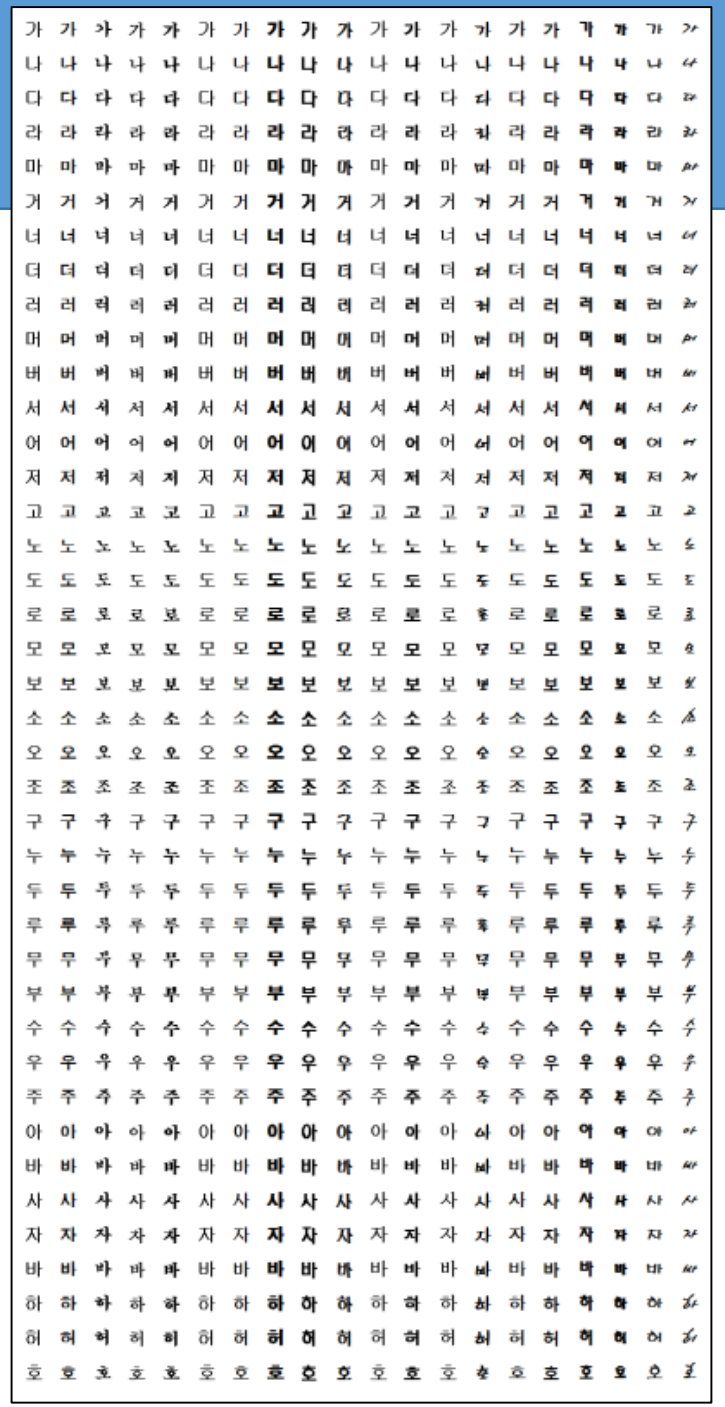
- 숫자 학습 영상

  train_numbers.png

- 문자 학습 영상 구성

  train_texts.png





문자학습

kNN_train("../image/trainimage/train_numbers.png", K1, 10, 20);

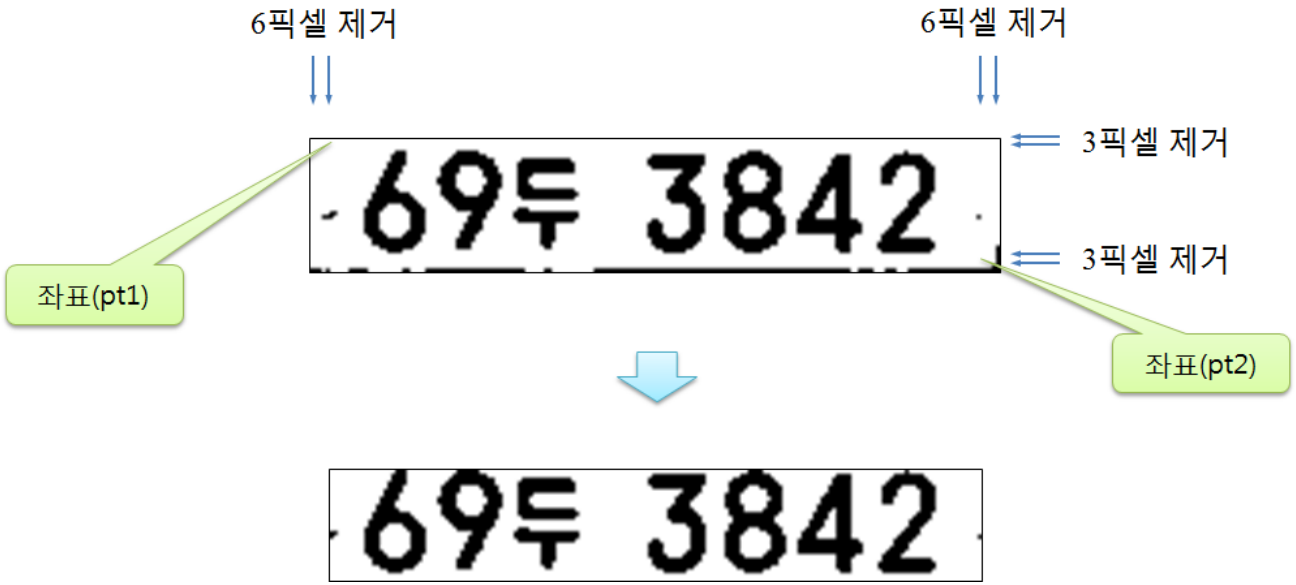
숫자 학습

kNN_train("../image/trainimage/train_texts.png ", K2, 40, 20);

전처리

❖전처리 과정

- 번호판 영상의 크기 180×35로 변경한 후에 이진화 수행
- 모서리 부분(좌우로 6픽셀 , 상하 3픽셀) 제거
  - 상하좌우 모서리 부분에서 잡음



❖ 전처리 수행 함수 구현 (./header/plate_classify.py)

```
def preprocessing_plate(plate_img):
    plate_img = cv2.resize(plate_img, (180, 35))                # 번호판 영상 크기 정규화
    flag = cv2.THRESH_BINARY | cv2.THRESH_OTSU                  # 이진화 방법
    cv2.threshold(plate_img, 32, 255, flag, plate_img)          # 이진화

    h, w = plate_img.shape[:2]
    dx, dy = (6, 3)                                             # 좌우 6화소, 상하 3화소
    ret_img= plate_img[dy:h-dy, dx:w-dx]                        # 여백 제거
    reutrn ret_img
```

숫자와 문자 객체 검출

❖숫자와 문자 객체 찾기 함수 find_objects() 로 구현

- cv::findContours() 함수로 객체 외곽선 검출
- 검출 객체 넓이로 잡음 제거
  - 150보다 작은 것은 잡음으로 처리
- 검출 객체 종횡비로 잡음제거
  - 가로로 긴 객체는 숫자, 문자 아님
- 번호판은 숫자와 문자 공존
  - 번호판 영상의 크기가 일정하기 때문에 문자 영역 위치 확인 가능
    - 45~80픽셀 범위 객체는 문자를 구성하는 연결요소로 판단
  - 문자 객체 여러 부분으로 분리되어 검출 가능
    - 문자 객체 영역들을 논리합으로 누적하여 하나의 영역으로 만듦
    - 문자 객체는 최소 넓이가 60이상으로 설정 (즉 넓이 60 미만은 잡영으로 판단)

❖ 숫자나 문자를 구성하는 연결요소의 ROI 찾기 (./header/plate_classify.py)

```
01  def find_objects(img):
02        results=cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
03        contours = results[0] if int(cv2.__version__[0]) >= 4 else results[1]
04
05        rois = [cv2.boundingRect(contour) for contour in contours]
06        rois = [(x, y, w, h, w*h) for x, y, w, h in rois if w / h < 2.5]
07        text_rois = [(x, y, x+w, y+h) for x, y, w, h, a in rois if 45<x<80 and a > 60]
08        num_rois = [(x, y, w, h) for x, y, w, h, a in rois if not(45<x<80) and a > 150]
09
10        if text_rois:                                  # 분리된 문자 영역 누적
11              pts= np.sort(atext_rois, axis=0)         # 세로 방향 정렬
12              x0, y0 = pts[0, 0:2]                     # 시작좌표 중 최소인 x, y 좌표
13              x1, y1 = pts[-1, 2:]                     # 종료좌표 중 최대인 x, y 좌표
14              w, h = x1-x0, y1-y0                      # 너비, 높이 계산
15              num_rois.append((x0, y0, w, h))          # 문자 영역 구성 및 저장
16        reutrn num_rois
```

문자 객체의 위치 범위

문자 객체 잡음 기준

숫자 객체 잡음 기준

숫자와 문자 객체 검출

❖윤곽선 검출

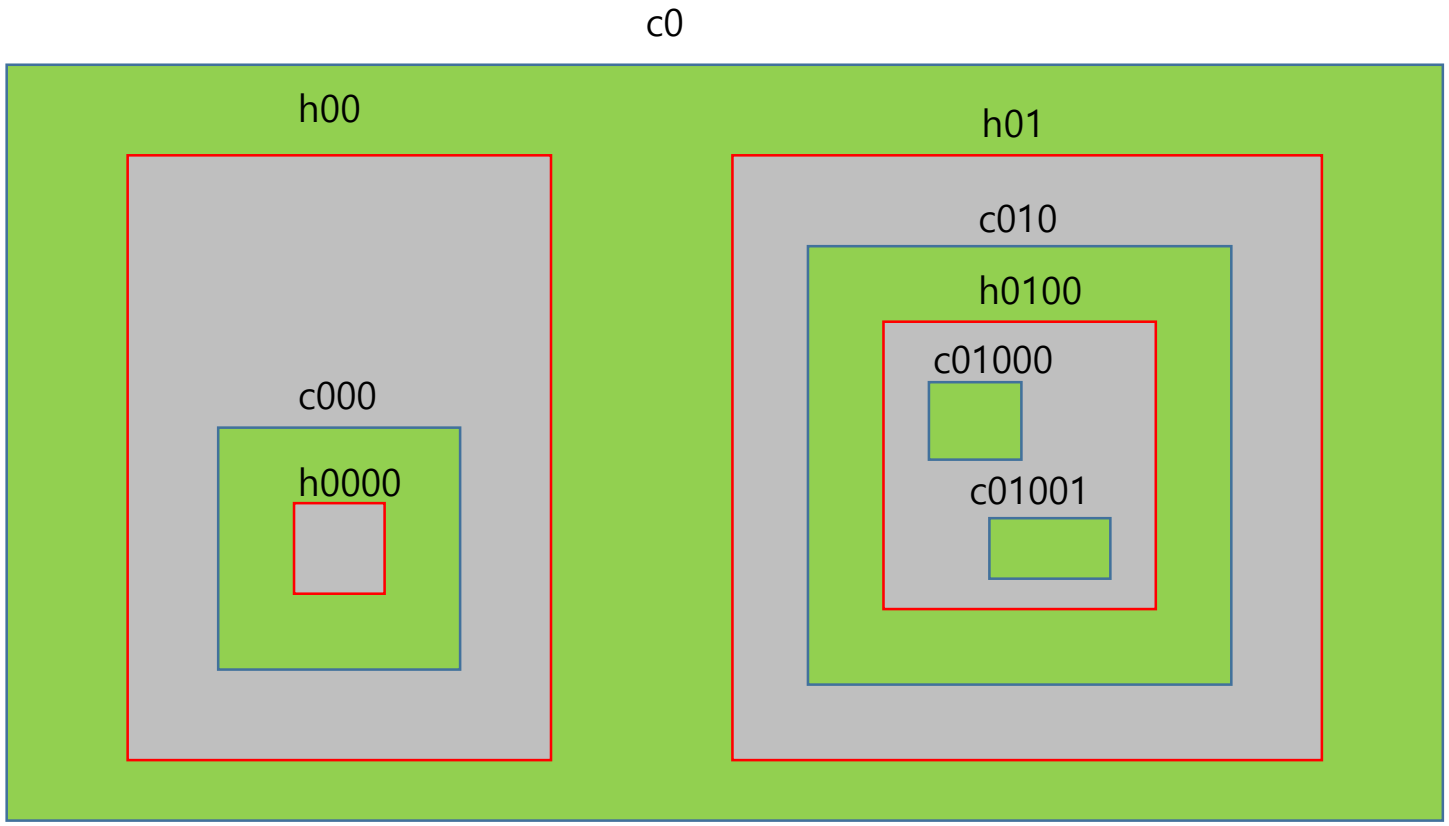
```
contours[1], hierarchy[2]  findContour(
            InputOutputArray image[3],  // 8비트 1채널 영상 (이진영상 취급)
            int   mode,          // 윤곽선의 검색 모드
            int   method,        // 윤곽선의 근사 방법
            Point offset=Point()     // 좌표 이동
)
```

1) 검출된 전체 윤곽선(contours)는 vector<vector<Point>> 자료형이며,
개별 윤곽선(contour)는 vector<Point> 자료형이다.

2) 계층구조(hierarchy)는 vector<Vec4i> 자료형으로, 순서대로 next, prev, 1st child, parent
윤곽선을 나타냄(대응하는 윤곽선이 없으면 음수 값을 가짐)

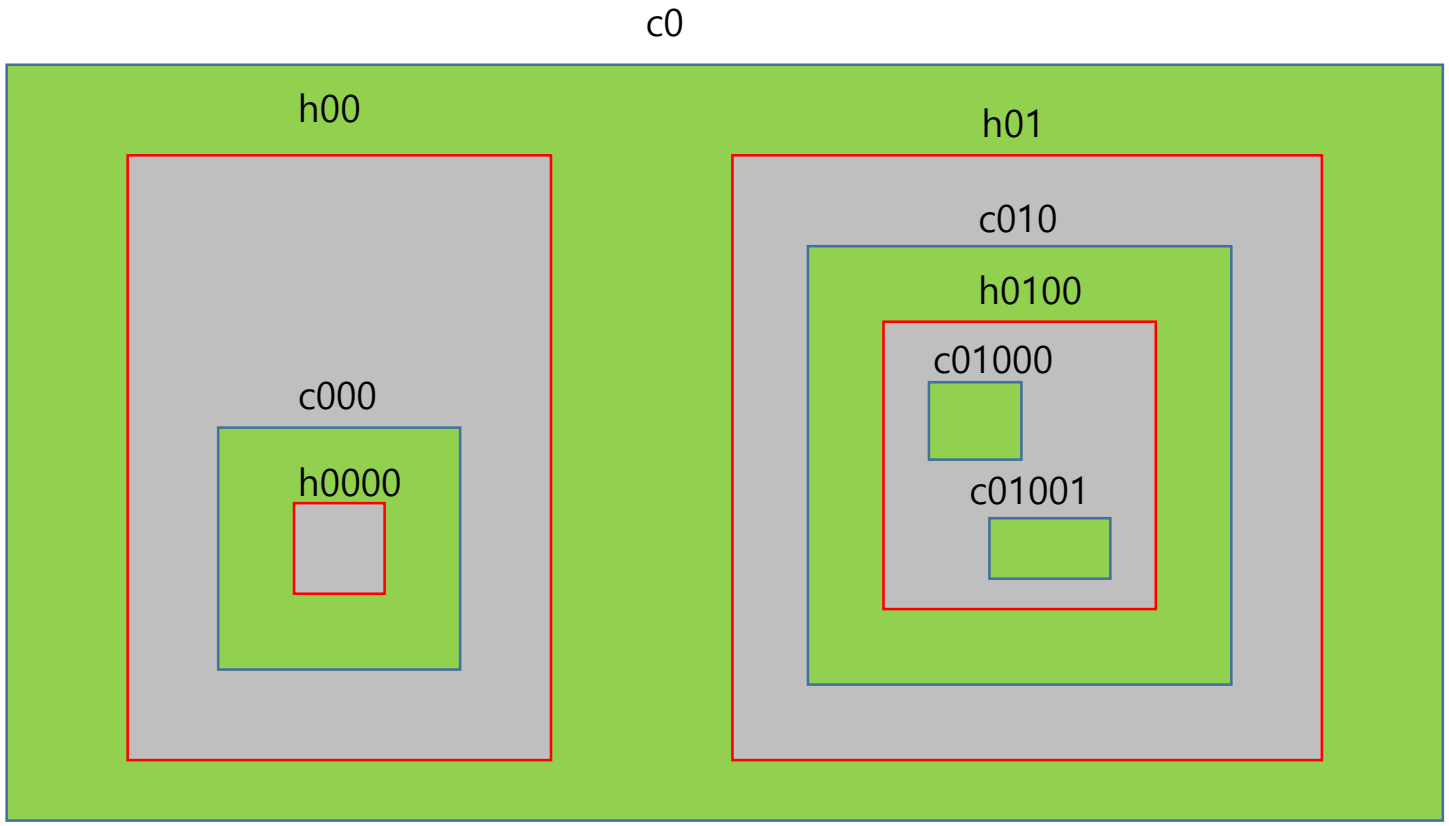
3) 0이 아닌 값은 1로 취급하여 이진영상으로 취급, 윤곽선을 검출하는 과정에서 수정됨

findContour mode = RETR_EXTERNAL



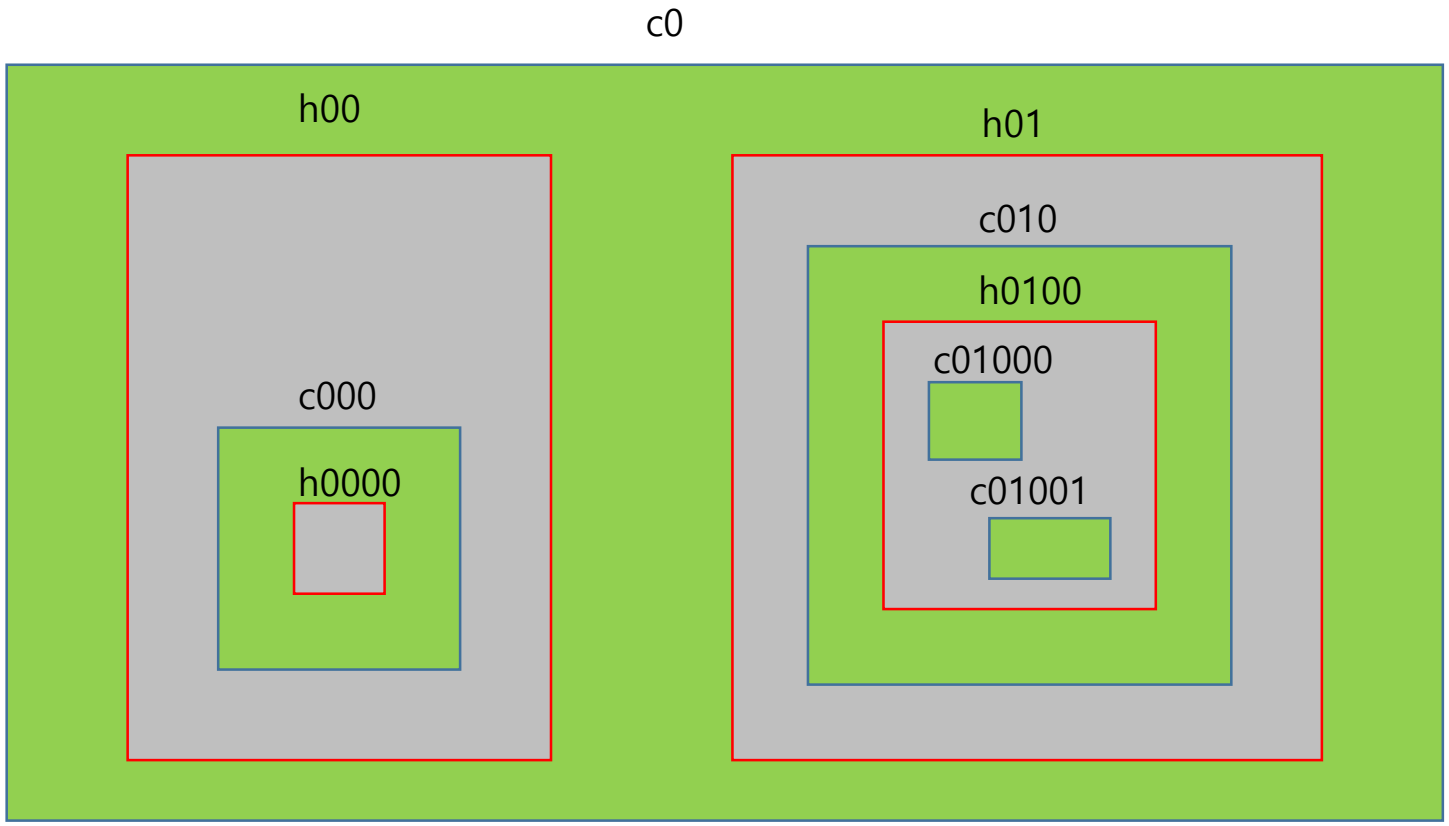
first_contour = c0

findContour mode = RETR_LIST



first_contour = c01001-c01000-h0100-h0000-c010-c000-h01-h00-c0

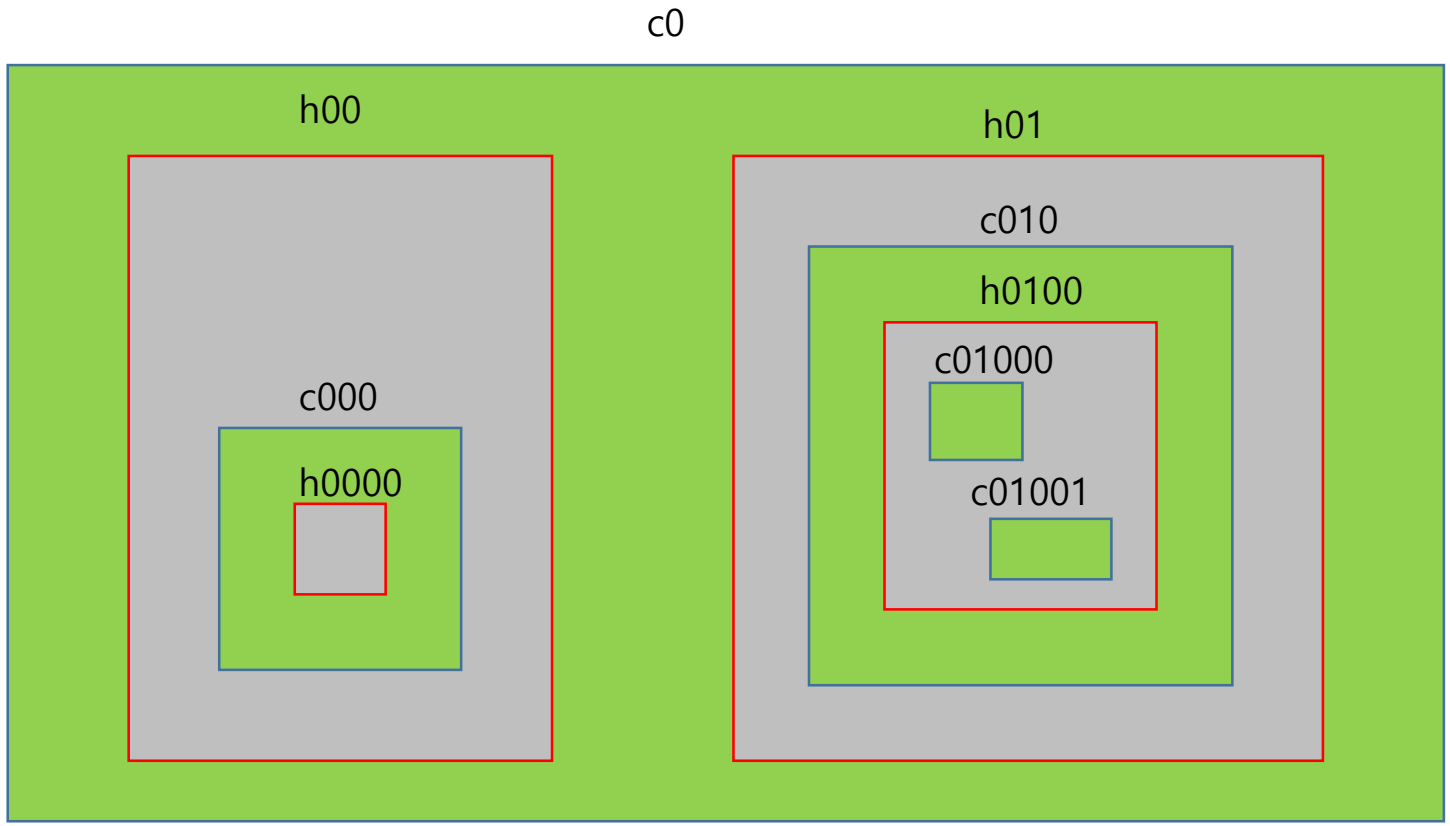
findContour mode = RETR_CCOMP

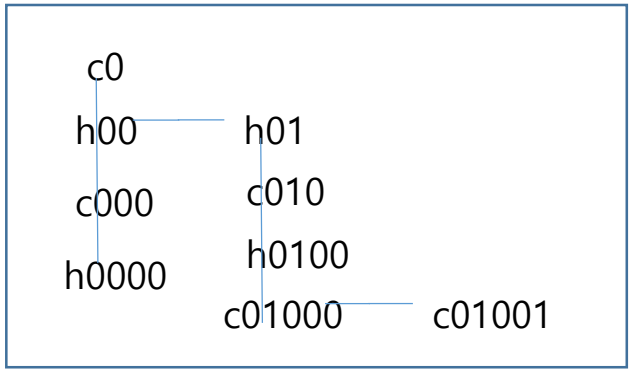


first_contour = c01001-c01000------c010----c000------c0

h0100 h0000 h01---h00

findContour mode = RETR_TREE





숫자와 문자 객체 검출

❖윤곽선 그리기

```
void  drawContour(
          InputOutputArray image,
          InputOutputArrays contours,
          int  contourIdx[1],        // 그릴 윤곽선 인덱스
          const Scalar& color,  int  thickness=1,  int  lineType[2]=8,
          InputArray hierarchy[3]=noArray(),   // 그릴 윤곽선의 최대 깊이
          int  maxLevel=INT_MAX,
          Point  offset=Point()              // 좌표 이동
)
```

1) 음수이면, 모든 윤곽선을 그린다.

2) 8이면 8연결, 4이면 4연결, 그리고 -1이면 anti-aliased line

3) 그릴 윤곽선의 최대 깊이 (윤곽선에 계층구조가 존재할 때만 유효)

검출된 개별 숫자/문자 영상 생성

❖숫자와 문자 사각형들 벡터(obejct_rects)에 저장

- cv::findContours() 함수로 객체 외각선 검출시
  - 검출 사각형들 위치 정렬되어 있지 않음
    →각 사각형이 몇 번째 숫자/문자 인지 확인 불가

- 각 사각형을 x좌표 순으로 정렬 필요
  - sort_rects() 함수로 구현: cv::sortIdx() 함수 이용

```python
def classify_numbers(cells, nknn, tknn, K1, K2, object_rois):
    if len(cells) != 7:
        print("검출된 숫자(문자)가 7개가 아닙니다.")
        reutrn

    texts =  "가나다라마거너더러머버서어저고노도로모보" \
             "소오조구누두루무부수우주아바사자바하허호"          # 학습 문자 집합
    numbers = [find_number(cell) for cell in cells]          # 숫자 객체 찾기
    datas = [place_middle(num, (40,40)) for num in numbers]  # 중심 배치
    datas = np.reshape(datas, (len(datas), -1))              # 1행 데이터 변환

    idx = np.argsort(object_rois, axis=0).T[0]               # x좌표 정렬 인덱스
    text = datas[idx[2]].reshape(1,-1)
    _, resp1, _, _ = nknn.findNearest(datas, K1)             # 숫자 k-NN 분류
    _, [[resp2]], _, _ = tknn.findNearest(text, K2)          # 문자 k-NN 분류

    resp1 = res1.flatten().astype('int')                     # 전개 및 정수변환
    results = resp1[idx].astype('str')
    results[2] = texts[int(resp2)]

    print("정렬 인덱스:", idx)
    print("숫자 객체 결과:", resp1)
    print("문자 객체 결과:", int(resp2))
    print("분류 결과: ", " ".join(results))
```

검출 객체(숫자/문자) 영상의 인식

❖검출 객체 영상 분류하여 숫자 및 문자 인식

- classify_numbers() 함수로 구현
  - 10장 3절 숫자 인식 예제에서 사용한 함수 사용

    find_number() 함수 – 숫자객체 검출

    place_middle() 함수 – 숫자 중앙배치 및 1행 데이터 생성

예제 12.3.1 번호판 숫자 인식 프로그램 완성 – 08.classify_numbers.cpp

```
01  from plate_preprocess import *                          # 전처리 및 후보 영역 검출 함수
02  from plate_candidate import *
03  from plate_classify import *                            # k-NN 학습 및 분류 함수 임포트
04
05  car_no = int(input("자동차 영상 번호(0~15): "))
06  image, morph = preprocessing(0)                         # 전처리
07  candidates = find_candidates(morph)                     # 후보 영역 검색
08
09  fills = [refine_candidate_img(image, size) for size, _, _ in candidates]
10  new_candi = [find_candidates(fill) for fill in fills]
11  new_candi = [cand[0] for cand in new_candi if cand]
12  candidate_imgs = [rotate_plate(image, cand) for cand in new_candi]
13
14  svm = cv2.ml.SVM_load("SVMTrain.xml")                   # 학습된 데이터 적재
15  rows = np.reshape(candidate_imgs, (len(candidate_imgs), -1))   # 1행 데이터들로 변환
16  _, results = svm.predict(rows.astype('float32'))        # 분류 수행
17  result = np.where(results.flatten() == 1)[0]            # 1인 값의 위치 찾기
18
19  plate_no = result[0] if len(result)>0 else -1           # 번호판 판정
20
21  K1, K2 = 10, 10
22  nknn = kNN_train("images/train_numbers.png", K1, 10, 20)    # 숫자 학습
23  tknn = kNN_train("images/train_texts.png", K2, 40, 20)      # 문자 학습
```

```python
if plate_no >= 0:
    plate_img = preprocessing_plate(candidate_imgs[plate_no]) # 번호판 전처리
    cells_roi = find_objects(cv2.bitwise_not(plate_img))
    cells = [plate_img[y:y+h, x:x+w] for x, y, w, h in cells_roi]    # 셀(숫자/문자) 영상 생성

    classify_numbers(cells, nknn, tknn, K1, K2, cells_roi)      # 숫자/문자 분류

    pts = np.int32(cv2.boxPoints(candidates[plate_no]))
    cv2.polylines(image, [pts], True,  (0, 255, 0), 2)          # 번호판 표시

    color_plate = cv2.cvtColor(plate_img, cv2.COLOR_GRAY2BGR) # 번호판 영상 컬러로
    for x, y, w, h in cells_roi:                                # 숫자/문자 사각형 표시
        cv2.rectangle(color_plate, (x,y), (x+w, y+h), (0, 0, 255), 1)

    h, w = color_plate.shape[:2]
    image[0:h, 0:w] = color_plate                               # 번호판 원본 영상에 복사
else:
    print("번호판 미검출")

cv2.imshow("image", image)
cv2.waitKey(0)
```

```
Run: 08.classify_number
C:\Python\python.exe D:/source/chap12/08.classify_number.py
자동차 영상 번호 (0~20): 1
정렬 인덱스: [5 0 6 4 3 2 1]
숫자 분류 결과: [3 8 3 4 0 2 2]
문자 분류 결과: 21
분류결과: 2 3 오 0 4 3 8
```