디지털 영상처리 연구실 연구보고서

김우헌

##레이블링

```
void labeling_basic()
{
    uchar data[] = {
        0, 0, 1, 1, 0, 0, 0, 0,
        1, 1, 1, 1, 0, 0, 1, 0,
        1, 1, 1, 1, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 1, 1, 0,
        0, 0, 0, 1, 1, 1, 1, 0,
        0, 0, 0, 1, 0, 0, 1, 0,
        0, 0, 1, 1, 1, 1, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0,
    };

    Mat src = Mat(8, 8, CV_8UC1, data) * 255;

    Mat labels;
    int cnt = connectedComponents(src, labels);
```

->Connectedcomponets 함수이용 src->labels로 레이블링, cnt반환(라벨링개수)

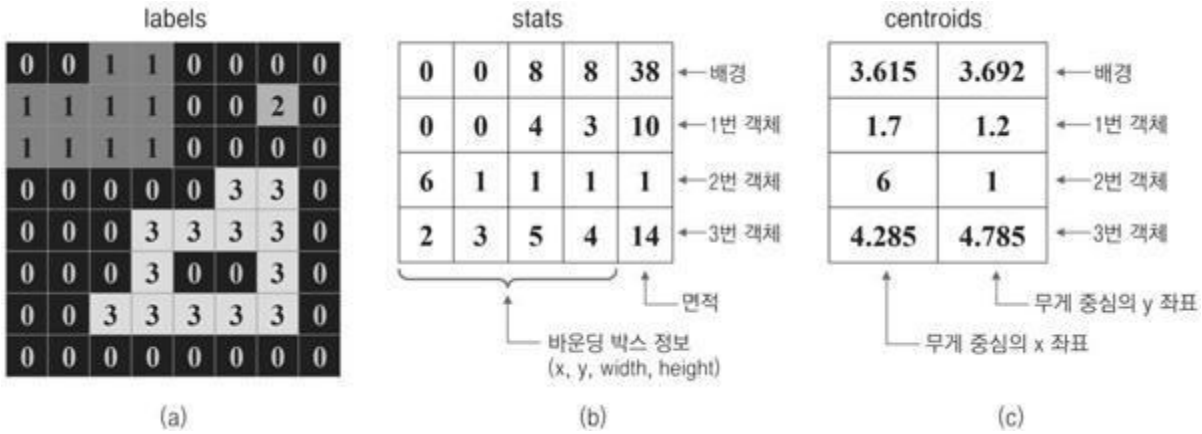
```
src:
[  0,   0, 255, 255,   0,   0,   0,   0;
 255, 255, 255, 255,   0,   0, 255,   0;
 255, 255, 255, 255,   0,   0,   0,   0;
   0,   0,   0,   0,   0, 255, 255,   0;
   0,   0,   0, 255, 255, 255, 255,   0;
   0,   0,   0, 255,   0,   0, 255,   0;
   0,   0, 255, 255, 255, 255, 255,   0;
   0,   0,   0,   0,   0,   0,   0,   0]
labels:
[0, 0, 1, 1, 0, 0, 0, 0;
 1, 1, 1, 1, 0, 0, 2, 0;
 1, 1, 1, 1, 0, 0, 0, 0;
 0, 0, 0, 0, 0, 3, 3, 0;
 0, 0, 0, 3, 3, 3, 3, 0;
 0, 0, 0, 3, 0, 0, 3, 0;
 0, 0, 3, 3, 3, 3, 3, 0;
 0, 0, 0, 0, 0, 0, 0, 0]
number of labels: 4
```

#레이블링 응용

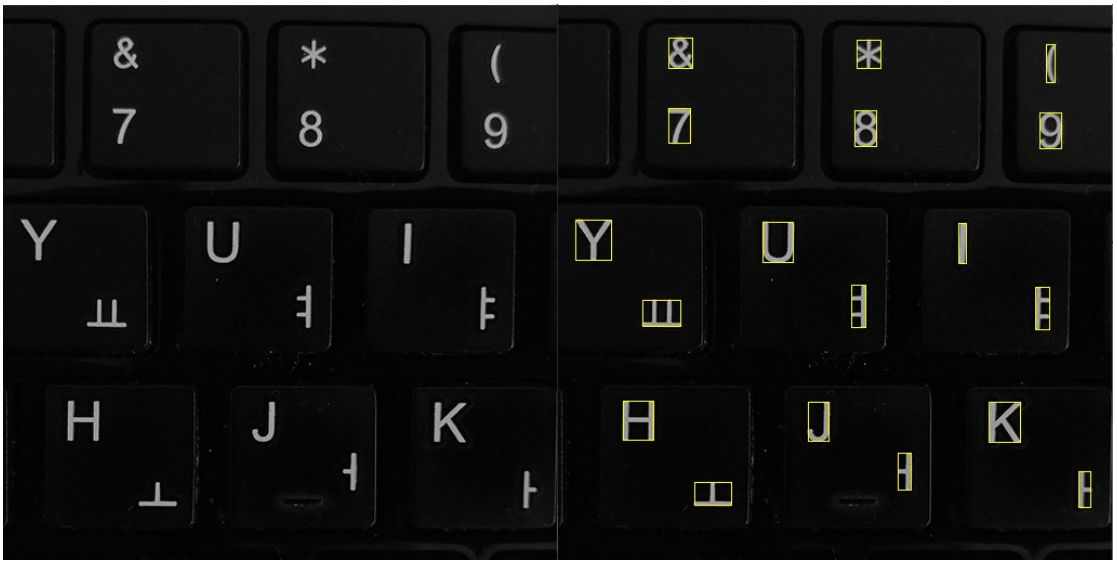
```
threshold(src, bin, 0, 255, THRESH_BINARY | THRESH_OTSU);

Mat labels, stats, centroids;
int cnt = connectedComponentsWithStats(bin, labels, stats, centroids);

Mat dst;
cvtColor(src, dst, COLOR_GRAY2BGR);

for (int i = 1; i < cnt; i++) {
    int* p = stats.ptr<int>(i);

    if (p[4] < 20) continue;

    rectangle(dst, Rect(p[0], p[1], p[2], p[3]), Scalar(0, 255, 255));
}
```

->레이블링을 위해 otsu이진화

->connectedcomponentsWithStats 함수(기존의 레이블링함수에 with stats추가)로 라벨링 상태를 stats,centroids에 저장

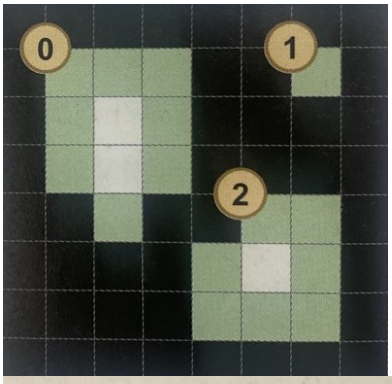


->stats정보를 이용하여 rectangle



#외곽선검출

->객체의 외곽선은 vector<Point> 배열로 나타낼 수 있고 여러 개의 그룹들이 존재하기 때문에 vector<vector<Point>>로 전체 이미지에서의 배열을 나타낼 수 있습니다.



vector<vector<Point>> contours;로 선언시

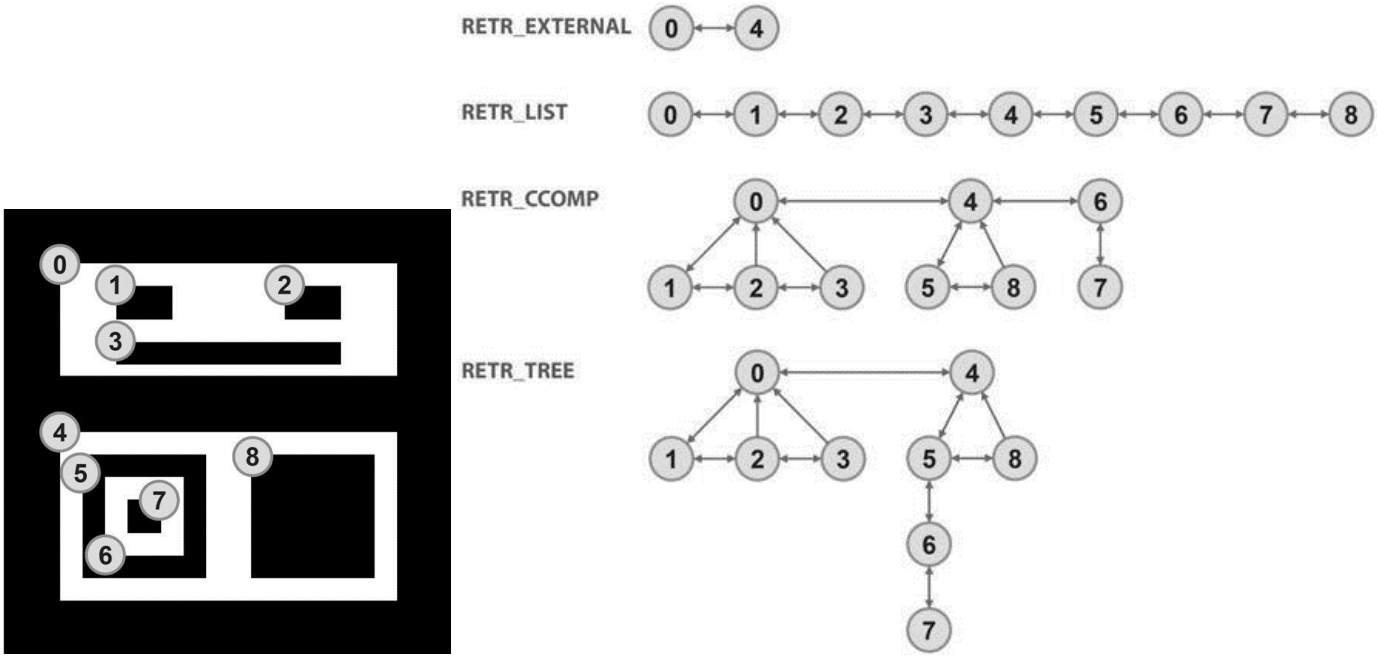
```
contours[0] : [1, 1], [1, 2], [1, 3], [2, 4], [3, 3], [3, 2], [3, 1], [2, 1]
contours[1] : [6, 1]
contours[2] : [5, 4], [4, 5], [4, 6], [5, 6], [6, 6], [6, 5], [6, 4]
```

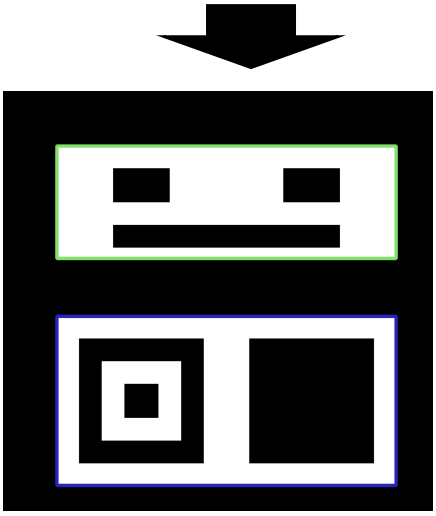
```
vector<vector<Point>> contours;
findContours(src, contours, RETR_LIST, CHAIN_APPROX_NONE);

Mat dst;
cvtColor(src, dst, COLOR_GRAY2BGR);

for (int i = 0; i < contours.size(); i++) {
    Scalar c(rand() & 255, rand() & 255, rand() & 255);
    drawContours(dst, contours, i, c, 2);
}
```

->findcontours함수를 사용해서 외곽선들을 contours 에 저장 RETR_LIST는 계층구조를 설정하는 요소, CHAIN_APPROX_NONE는 모든 외곽선 점들의 좌표를 저장

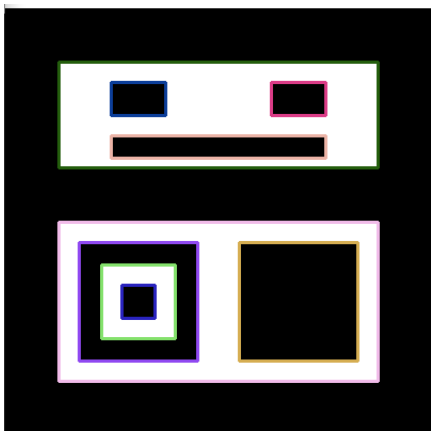
->drawcontours 함수를 이용하여 저장된 외곽선들을 모두 RANDOM한 COLOR로 DISPLAY.

**계층구조??



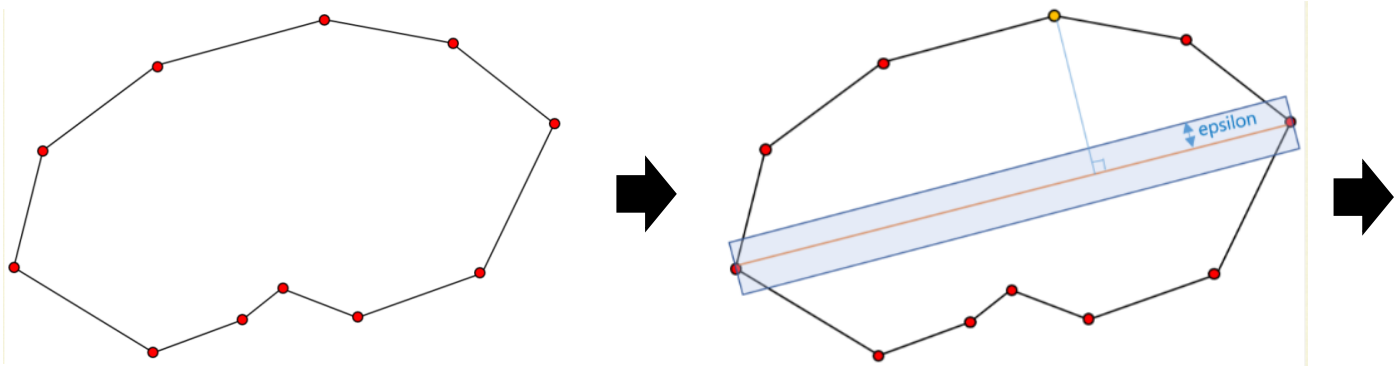


->EXTERNAL

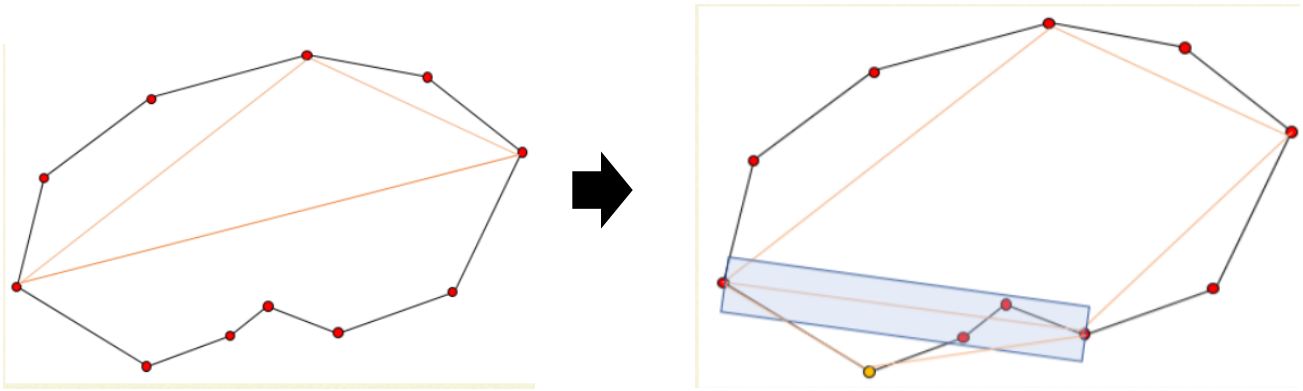


->LIST

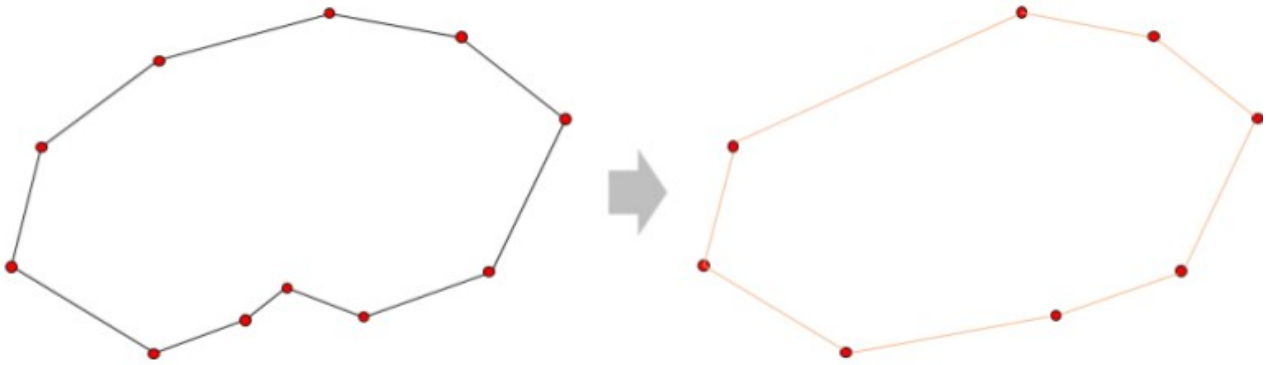
->더글라스-포이커 알고리즘을 사용한 외곽선 단순화.



->임계치보다 큰 거리(직선-꼭짓점)에 있는 꼭짓점 선택



--→ 결과



->코드구현

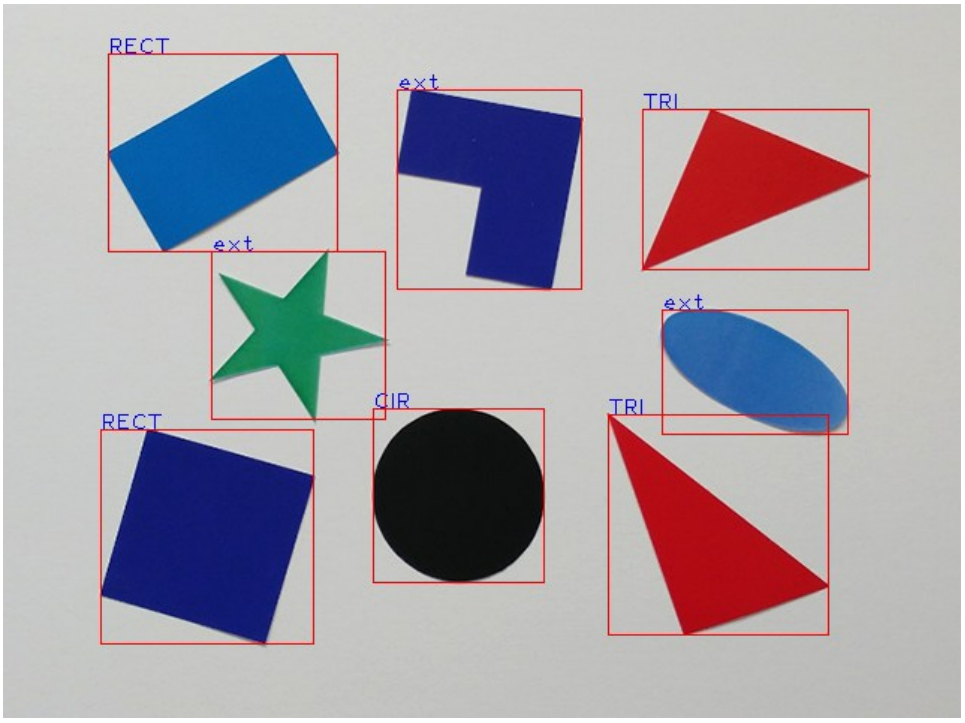
```
vector<vector<Point>> contours;
findContours(bin, contours, RETR_EXTERNAL, CHAIN_APPROX_NONE);

for (vector<Point> pts : contours) {
    if (contourArea(pts) < 400)
        continue;

    vector<Point> approx;
    approxPolyDP(pts, approx, arcLength(pts, true) * 0.02, true);

    int vtc = (int)approx.size();

    if (vtc == 3) {
       setLabel(img, pts, "TRI");
    }
    else if (vtc == 4) {
        setLabel(img, pts, "RECT");
    }
    else {
        double len = arcLength(pts, true);
        double area = contourArea(pts);
        double ratio = 4. * CV_PI * area / (len * len);

        if (ratio > 0.85) {
            setLabel(img, pts, "CIR");
        }else  setLabel(img, pts, "ext");
    }
}
```

-> findcontours함수를 통해 외곽선 검출

->for문을 통해 외곽선 그룹별로 approxpolydp 함수를 통해 외곽선 단순화

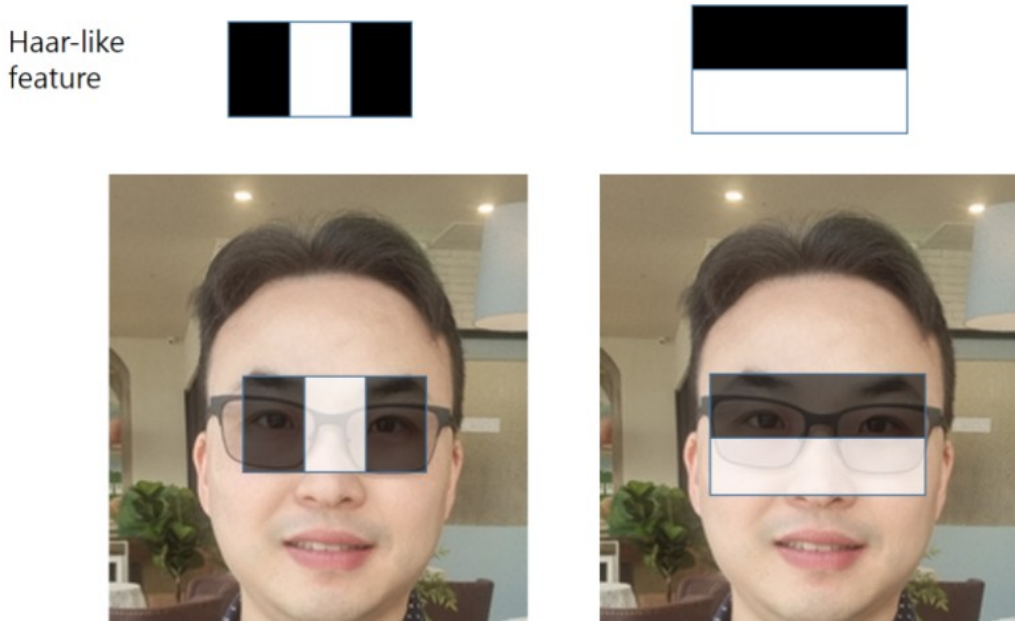
->단순화된 외곽선 (꼭지점개수)vtc 별로 다각형 구분 (삼각형,사각형,원,나머지)

->R=4*파이*A/P^2 공식에 의해 R이 1에 수렴할수록 원에 근접



##캐스케이드 분류기와 얼굴검출

->비올라-존스가 유사-하르 필터 로 특징을 추출하여 얼굴여부를 판별하는 것을 기초로 합니다.



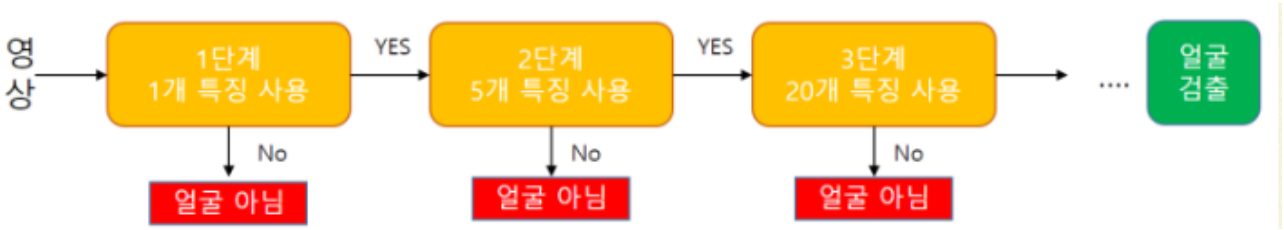
->검정부분과 흰부분에 해당하는 밝기 값을 빼서 일정값 이상인 것들을 찾는 방식

- ◆ 눈,눈썹,입술등은 어둡고 미간,볼,이마는 밝은 특징 이용

->수많은 필터로 인하여 시간이 오래 걸리는 단점이 존재

- ◆ 에이다 부스트 알고리즘,적분 영상을 통해 18만개 필터->6000개필터로 선별
- ◆ 그래도 많은 연산량

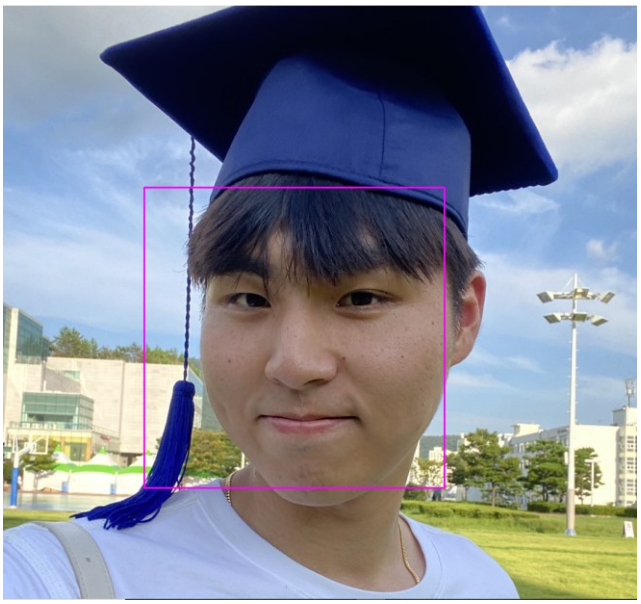
#캐스케이드 구조사용



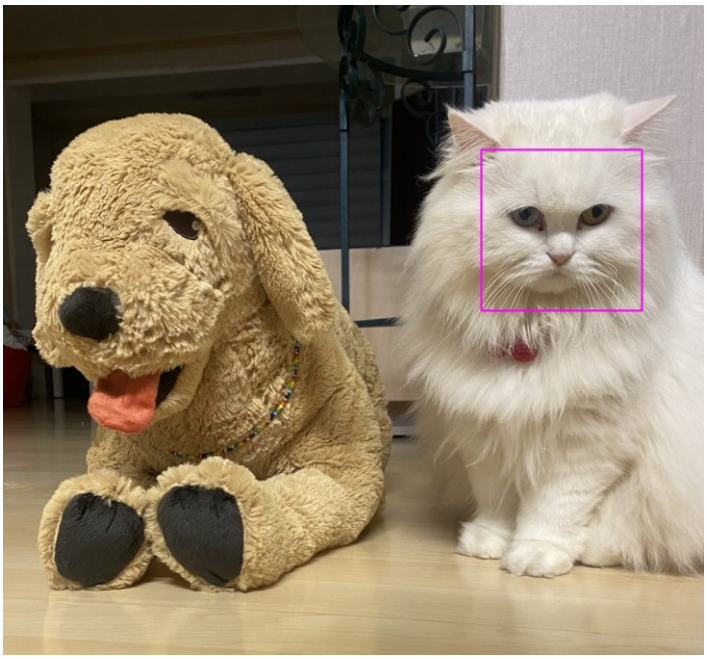
->CascadeClassifier 클래스 사용

```
CascadeClassifier classifier("haarcascade_frontalface_default.xml");
vector<Rect> faces;
classifier.detectMultiScale(src, faces);

for (Rect rc : faces) {
    rectangle(src, rc, Scalar(255, 0, 255), 2);
}
```

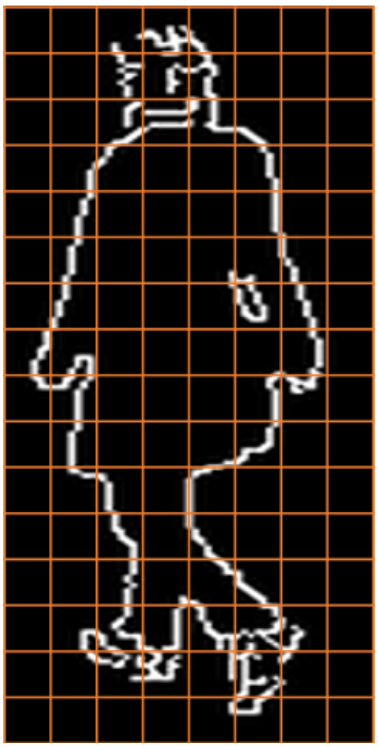
-> CascadeClassifier를 생성한후 LOAD함수를 이용



-> haarcascade_frontalcatface.xml 을 로드 하여 고양이 얼굴검출



##HOG 알고리즘

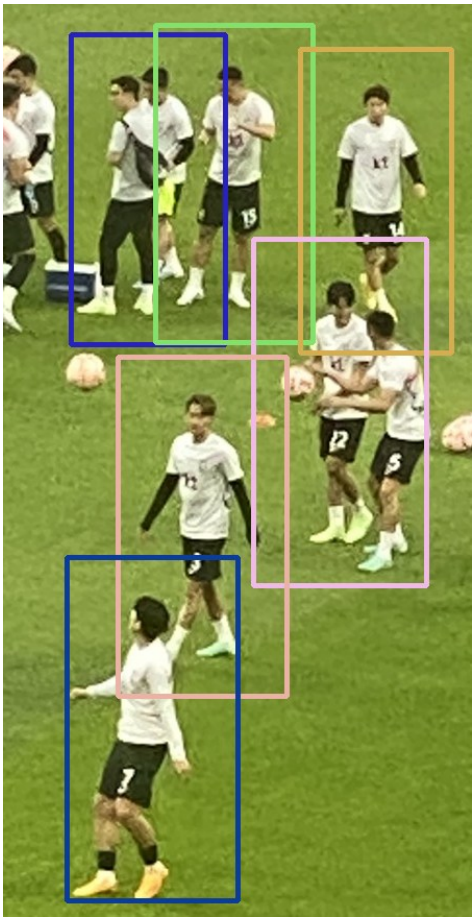


->64*128 영상 -> 8X8크기 단위로 분할->그래디언트방향(0~180)을 9개의 빈으로 군집화

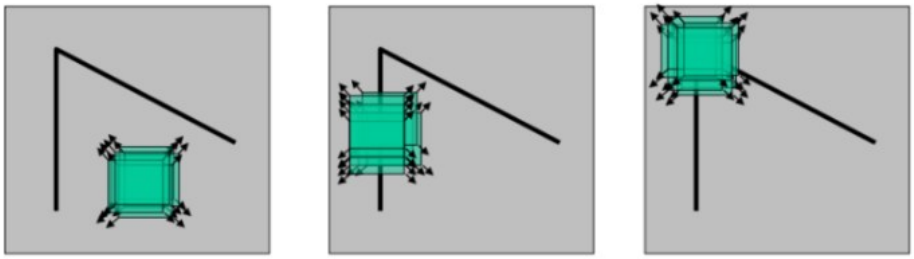
->4개의 단위를 1개의 박스-> 1개의 박스에 36개의 파라메타->105*36=3780의 실수값 이 특징벡터 역할

```
HOGDescriptor hog;
hog.setSVMDetector(HOGDescriptor::getDefaultPeopleDetector());

 vector<Rect> detected;
 hog.detectMultiScale(src, detected,0,Size(), Size(),1.03,1.4);

 for (Rect r : detected) {
     Scalar c = Scalar(rand() % 256, rand() % 256, rand() % 256);
     rectangle(src, r, c, 3);
 }
```

-> HOGDescriptor 객체 생성후 HOGDescriptor::getDefaultPeopleDetector() 보행자 검출을 위해 훈련된 분류기 계수를 setSVMDetector 함수에 입력하고 detectMultiScale을 통한 객체 검출 실시



##해리스 코너 검출



"flat" region: no change in all directions

"edge": no change along the edge direction

"corner": significant change in all directions

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

->E(u,v)의 값이 크면 코너라고 판별

$$R = \det(M) - k(\text{trace}(M))^2$$

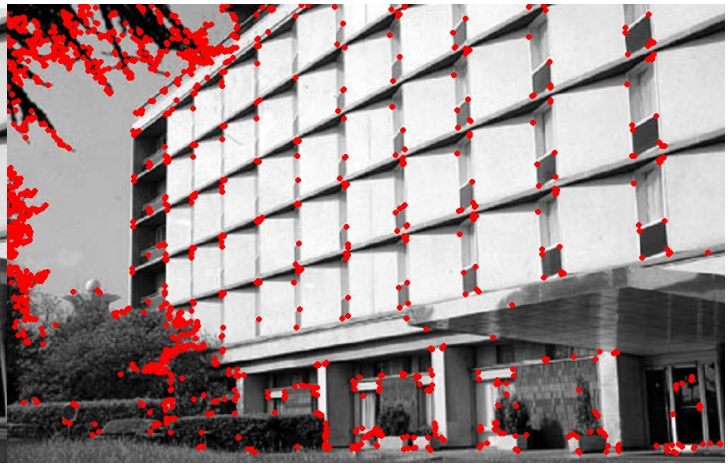
->테일러급수,고윳값분석을 통하여 코너 응답함수R유도

->R이 0보다 충분히 큰양수 ->코너,R=0 은 FLAT 한 영역, R<0일경우 EDGE라고 판단

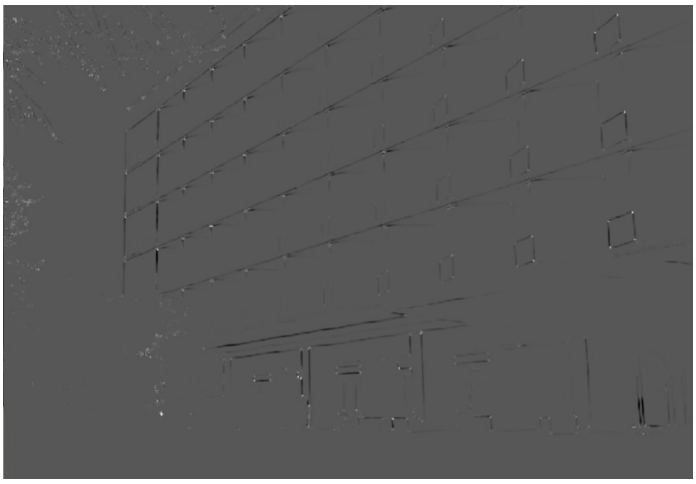
```
Mat src = imread("building.jpg", IMREAD_GRAYSCALE);
Mat harris;
cornerHarris(src, harris, 3, 3, 0.04);

Mat harris_norm;
normalize(harris, harris_norm, 0, 255, NORM_MINMAX, CV_8U);

Mat dst;
cvtColor(src, dst, COLOR_GRAY2BGR);

for (int j = 1; j < harris.rows - 1; j++) {
    for (int i = 1; i < harris.cols - 1; i++) {
        if (harris_norm.at<uchar>(j, i) > 120) {
            if (harris.at<float>(j, i) > harris.at<float>(j - 1, i) &&
                harris.at<float>(j, i) > harris.at<float>(j + 1, i) &&
                harris.at<float>(j, i) > harris.at<float>(j, i - 1) &&
                harris.at<float>(j, i) > harris.at<float>(j, i + 1)) {
                circle(dst, Point(i, j), 5, Scalar(0, 0, 255), 2);
            }
        }
    }
}
```

-> cornerharris 함수를 통해 R값 얻음

k=0.04



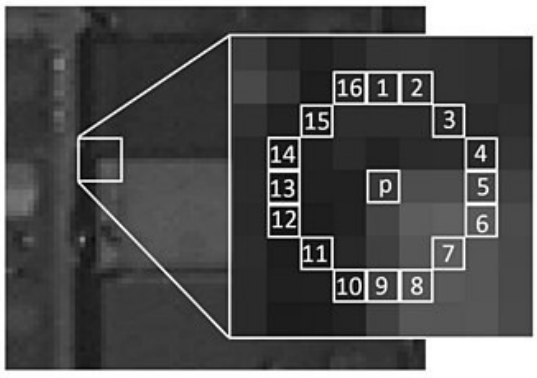
k=0.06



->0~255값으로 정규화된 R값

#FAST코너검출

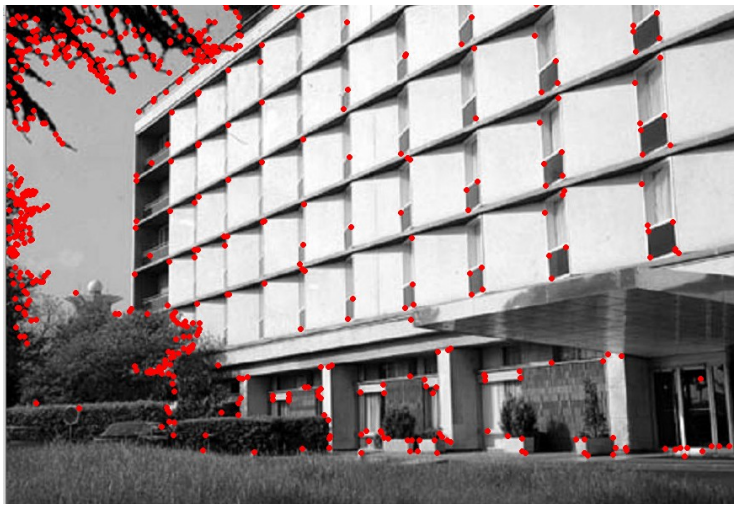
->해리스 코너검출의 느린 연산속도 개선방법



->주변 16개의 픽셀보다 충분이 밝거나 충분히 어두운 픽셀이 9개 이상 연속으로 존재하면 코너로 정의

```
Mat src = imread("building.jpg", IMREAD_GRAYSCALE);

if (src.empty()) {
    cerr << "Image load failed!" << endl;
    return;
}

vector<KeyPoint> keypoints;
FAST(src, keypoints, 60, true);

Mat dst;
cvtColor(src, dst, COLOR_GRAY2BGR);

for (KeyPoint kp : keypoints) {
    Point pt(cvRound(kp.pt.x), cvRound(kp.pt.y));
    circle(dst, pt, 5, Scalar(0, 0, 255), 2);
}
```

-> FAST함수를 통하여 keypoint에 포인터 저장



->thres=60 을 통한 코너 검출