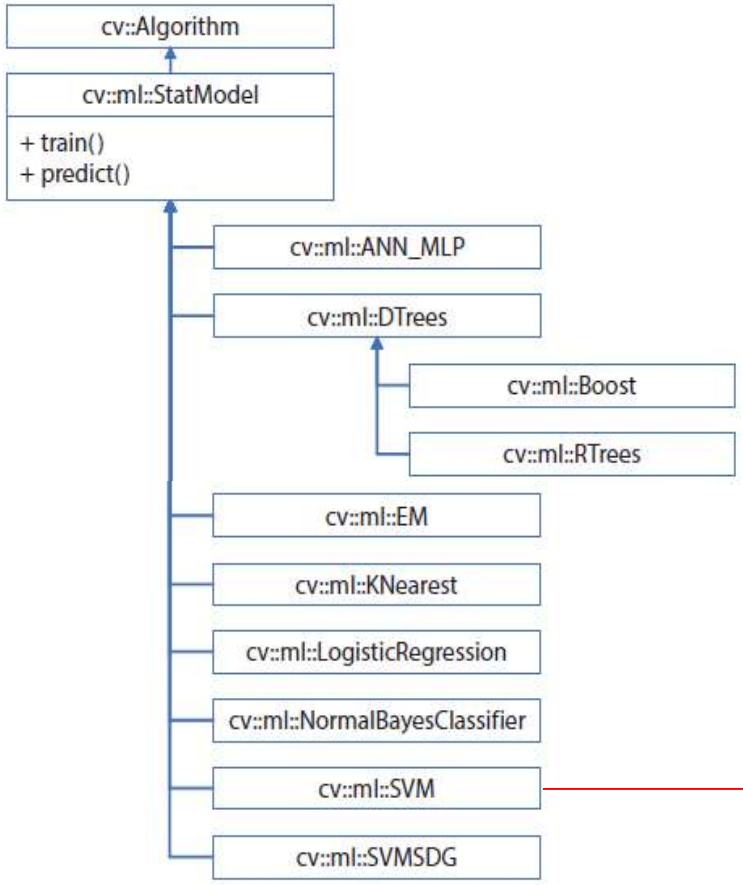
Support Vector Machine
Part 2

OpenCV

- 객체생성




```
static Ptr<SVM> SVM::create();
```

• 반환값 SVM 객체를 참조하는 Ptr 스마트 포인터 객체

OpenCV

- Type 설정

```
virtual void SVM::setType(int val)
```

- val SVM 타입. SVM::Types 열거형 상수 중 하나를 지정합니다.

SVM::Types 열거형 상수	설명	파라미터
C_SVC	C-서포트 벡터 분류. 일반적인 n-클래스 분류 문제에서 사용됩니다.	C
NU_SVC	v-서포트 벡터 분류. C_SVC와 비슷하지만 Nu 값 범위가 0~1 사이로 정규화되어 있습니다.	Nu
ONE_CLASS	1-분류 서포트 벡터 머신. 데이터 분포 측정에 사용됩니다.	C, Nu
EPS_SVR	ε-서포트 벡터 회귀	P, C
NU_SVR	v-서포트 벡터 회귀	Nu, C

OpenCV

- Kernel 함수 설정

```
virtual void SVM::setKernel(int kernelType);
```

• kernelType　　커널 함수 종류. SVM::KernelTypes 열거형 상수 중 하나를 지정합니다.

SVM::KernelTypes 열거형 상수	설명	파라미터
LINEAR	선형 커널	
POLY	다항식 커널	Degree, Gamma, Coef0
RBF	방사 기저 함수 커널	Gamma
SIGMOID	시그모이드 커널	Gamma, Coef0
CHI2	지수 카이 제곱 커널	Gamma
INTER	히스토그램 교차 커널	

OpenCV

- 파라메터 설정
  - C, Nu, P, Degree, Gamma, Coeff 등
  - 자동으로 찾아 학습하는 기능 제공

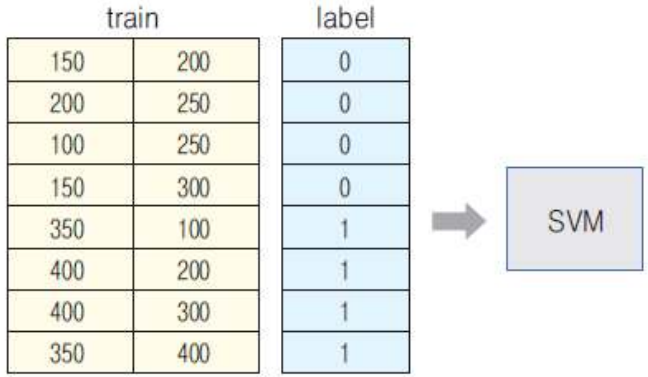
```
virtual bool SVM::trainAuto(InputArray samples,
                            int layout,
                            InputArray responses,
                            int kFold = 10,
                            Ptr<ParamGrid> Cgrid = SVM::getDefaultGridPtr(SVM::C),
                            Ptr<ParamGrid> gammaGrid  = SVM::getDefaultGridPtr(SVM::GAMMA),
                            Ptr<ParamGrid> pGrid      = SVM::getDefaultGridPtr(SVM::P),
                            Ptr<ParamGrid> nuGrid     = SVM::getDefaultGridPtr(SVM::NU),
                            Ptr<ParamGrid> coeffGrid  = SVM::getDefaultGridPtr(SVM::COEF),
                            Ptr<ParamGrid> degreeGrid = SVM::getDefaultGridPtr(SVM::DEGREE),
                            bool balanced = false)
```

• samples	훈련 데이터 행렬
• layout	훈련 데이터 배치 방법. ROW_SAMPLE 또는 COL_SAMPLE를 지정합니다.
• responses	각 훈련 데이터에 대응되는 응답 벡터
• kFold	교차 검증을 위한 부분 집합 개수
• Cgrid	C 탐색 범위
• gammaGrid	gamma 탐색 범위
• pGrid	p 탐색 범위
• nuGrid	nu 탐색 범위
• coeffGrid	coeff 탐색 범위
• degreeGrid	degree 탐색 범위
• balanced	이 값이 true이고 두 클래스 분류 문제인 경우, 전체 훈련 데이터 비율을 고려하여 좀 더 균형 잡힌 교차 검증 부분 집합을 생성합니다.
• 반환값	정상적으로 학습이 완료되면 true를 반환합니다.

OpenCV

코드 15-5 SVM 알고리즘을 이용한 2차원 점 분류 [ch15/svmplane]

```
01  #include "opencv2/opencv.hpp"
02  #include <iostream>
03
04  using namespace cv;
05  using namespace cv::ml;
06  using namespace std;
07
08  int main(void)
09  {
10      Mat train = Mat_<float>({ 8, 2 }, {
11          150, 200, 200, 250, 100, 250, 150, 300,
12          350, 100, 400, 200, 400, 300, 350, 400 });
13      Mat label = Mat_<int>({ 8, 1 }, { 0, 0, 0, 0, 1, 1, 1, 1 });
14
15      Ptr<SVM> svm = SVM::create();
16      svm->setType(SVM::Types::C_SVC);
17      svm->setKernel(SVM::KernelTypes::RBF);
18      svm->trainAuto(train, ROW_SAMPLE, label);
19
20      Mat img = Mat::zeros(Size(500, 500), CV_8UC3);
21
```

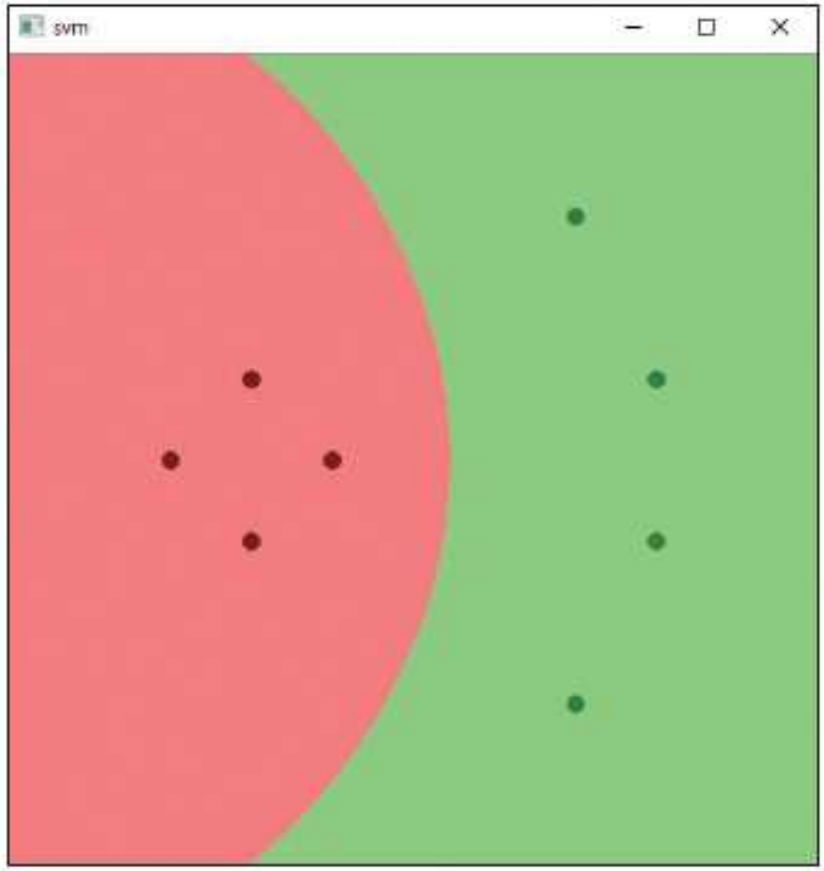


OpenCV

```
22      for (int j = 0; j < img.rows; j++) {
23          for (int i = 0; i < img.cols; i++) {
24              Mat test = Mat_<float>({ 1, 2 }, { (float)i, (float)j });
25              int res = cvRound(svm->predict(test));
26
27              if (res == 0)
28                  img.at<Vec3b>(j, i) = Vec3b(128, 128, 255); // R
29              else
30                  img.at<Vec3b>(j, i) = Vec3b(128, 255, 128); // G
31          }
32      }
33
34      for (int i = 0; i < train.rows; i++) {
35          int x = cvRound(train.at<float>(i, 0));
36          int y = cvRound(train.at<float>(i, 1));
37          int l = label.at<int>(i, 0);
38
39          if (l == 0)
40              circle(img, Point(x, y), 5, Scalar(0, 0, 128), -1, LINE_AA); // R
41          else
42              circle(img, Point(x, y), 5, Scalar(0, 128, 0), -1, LINE_AA); // G
43      }
44
45      imshow("svm", img);
46
47      waitKey();
48      return 0;
49  }
```

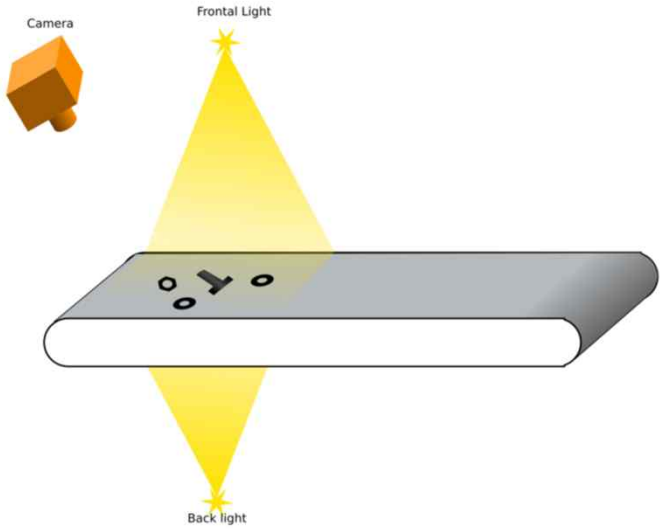
OpenCV

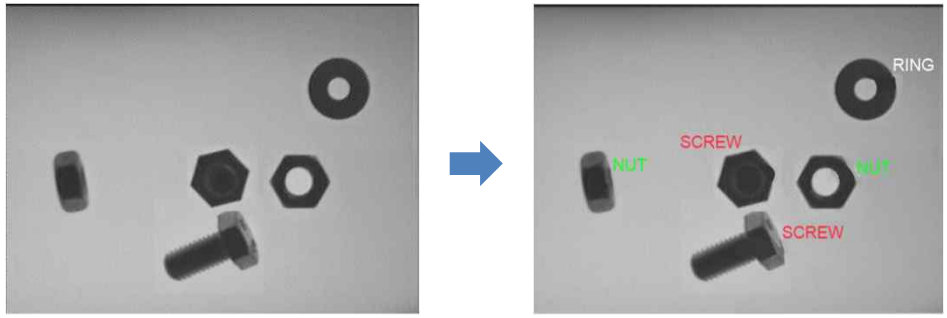
- 실행결과



AOI

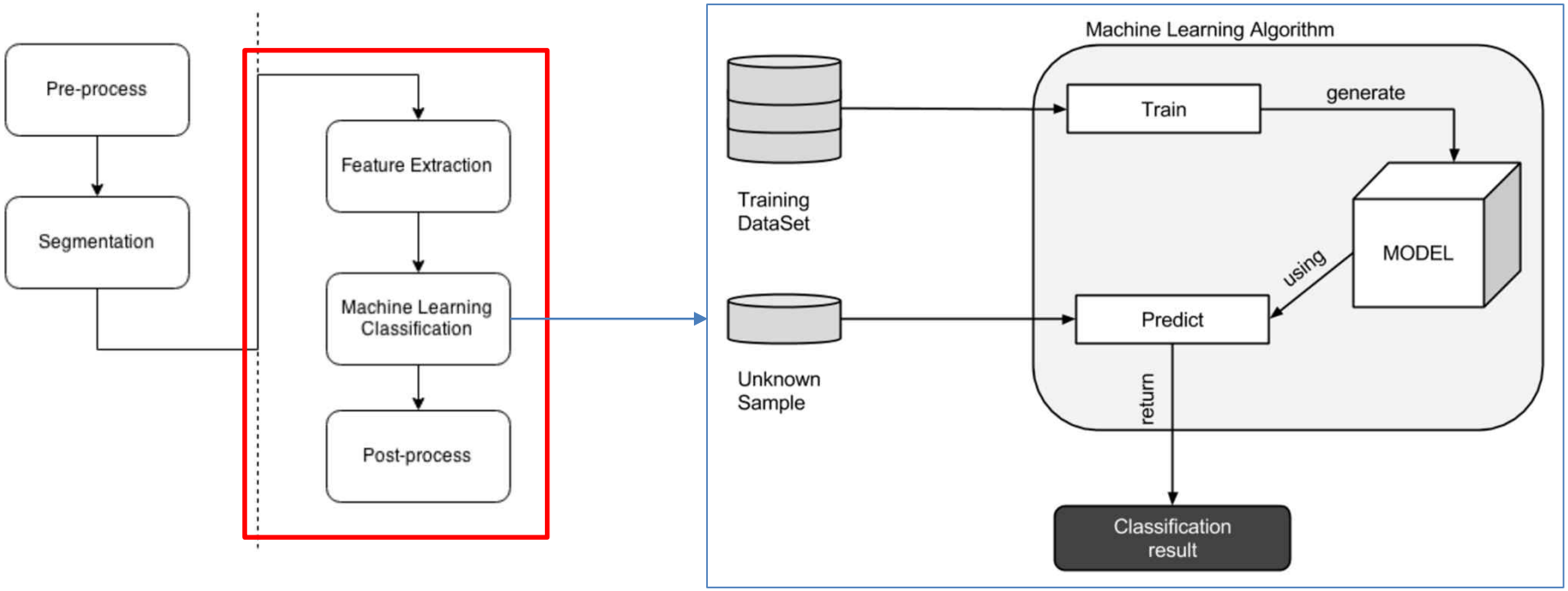
- Object Classification





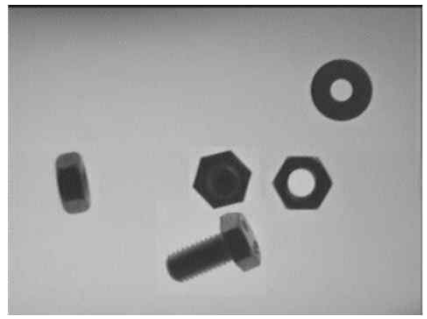
AOI

- Machine Learning Classification



AOI

- Feature Extraction
  - Area
  - Aspect ratio (가로/세로 비율)
  - Number of holes
  - Number of contour size
  - ……



{Ring, Nut, Screw}

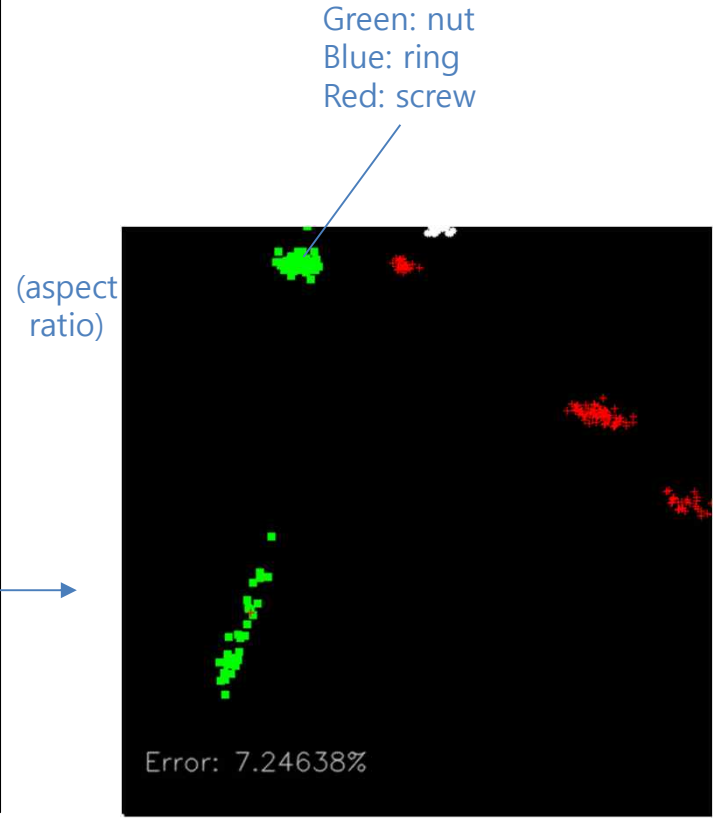
```
vector< vector<float> > ExtractFeatures(Mat img, vector<int>* left=NULL, vector<int>* top=NULL)
{
  vector< vector<float> > output;
  vector<vector<Point> > contours;
  Mat input= img.clone();

  vector<Vec4i> hierarchy;
  findContours(input, contours, hierarchy, RETR_CCOMP, CHAIN_APPROX_SIMPLE);
  // Check the number of objects detected
  if(contours.size() == 0 ){
    return output;
  }
  RNG rng( 0xFFFFFFFF );
  for(int i=0; i<contours.size(); i++){
    Mat mask= Mat::zeros(img.rows, img.cols, CV_8UC1);
    drawContours(mask, contours, i, Scalar(1), FILLED, LINE_8, hierarchy, 1);
    Scalar area_s= sum(mask);
    float area= area_s[0];

    if(area>500){ //if the area is greather than min.
      RotatedRect r= minAreaRect(contours[i]);
      float width= r.size.width;
      float height= r.size.height;
      float ar=(width<height)?height/width:width/height;

      vector<float> row;
      row.push_back(area);
      row.push_back(ar);
      output.push_back(row);
      if(left!=NULL){
          left->push_back((int)r.center.x);
      }
      if(top!=NULL){
          top->push_back((int)r.center.y);
      }
      miw->addImage("Extract Features", mask*255);
      miw->render();
      waitKey(10);
   }
  }
  return output;
}
```

AOI

- Training an SVM model

```
void trainAndTest()
{
  vector< float > trainingData;
  vector< int > responsesData;
  vector< float > testData;
  vector< float > testResponsesData;

  int num_for_test= 20;

  // Get and process the nut images
  readFolderAndExtractFeatures("../data/nut/tuerca_%04d.pgm", 0, num_for_test, trainingData, responsesData, testData, testResponsesData);
  // Get and process the ring images
  readFolderAndExtractFeatures("../data/ring/arandela_%04d.pgm", 1, num_for_test, trainingData, responsesData, testData, testResponsesData);
  // Get and process the screw images
  readFolderAndExtractFeatures("../data/screw/tornillo_%04d.pgm", 2, num_for_test, trainingData, responsesData, testData, testResponsesData);

  cout << "Num of train samples: " << responsesData.size() << 두이;
  cout << "Num of test samples: " << testResponsesData.size() << 두이;

  // Merge all data
  Mat trainingDataMat(trainingData.size()/2, 2, CV_32FC1, &trainingData[0]);  // 2 features per image
  Mat responses(responsesData.size(), 1, CV_32SC1, &responsesData[0]);        // 1 label per image

  Mat testDataMat(testData.size()/2, 2, CV_32FC1, &testData[0]);
  Mat testResponses(testResponsesData.size(), 1, CV_32FC1, &testResponsesData[0]);

  Ptr<TrainData> tdata= TrainData::create(trainingDataMat, ROW_SAMPLE, responses);
```

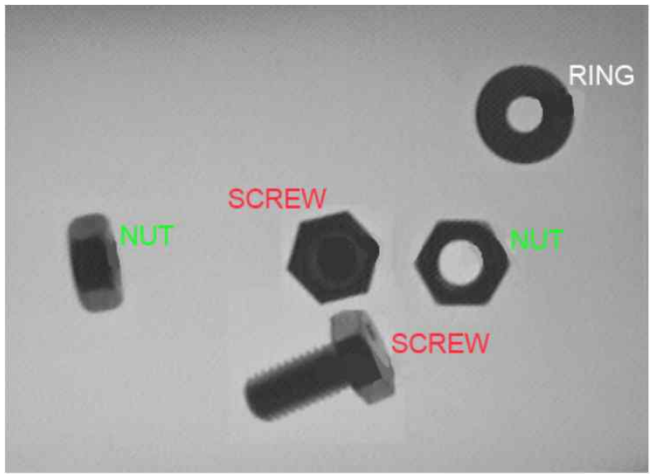
```
svm = cv::ml::SVM::create();      // Ptr<SVM> svm;
svm->setType(cv::ml::SVM::C_SVC);
svm->setNu(0.05);
svm->setKernel(cv::ml::SVM::CHI2);
svm->setDegree(1.0);
svm->setGamma(2.0);
svm->setTermCriteria(TermCriteria(TermCriteria::MAX_ITER, 100, 1e-6));
svm->train(tdata);

  if(testResponsesData.size()>0){
    cout << "Evaluation" << endl;
    cout << "==========" << endl;
    // Test the ML Model
    Mat testPredict;
    svm->predict(testDataMat, testPredict);
    cout << "Prediction Done" << endl;
    // Error calculation
    Mat errorMat= testPredict!=testResponses;
    float error= 100.0f * countNonZero(errorMat) / testResponsesData.size();
    cout << "Error: " << error << "₩%" << endl;
    // Plot training data with error label
    plotTrainData(trainingDataMat, responses, &error);

  }else{
    plotTrainData(trainingDataMat, responses);
  }
}
```



AOI

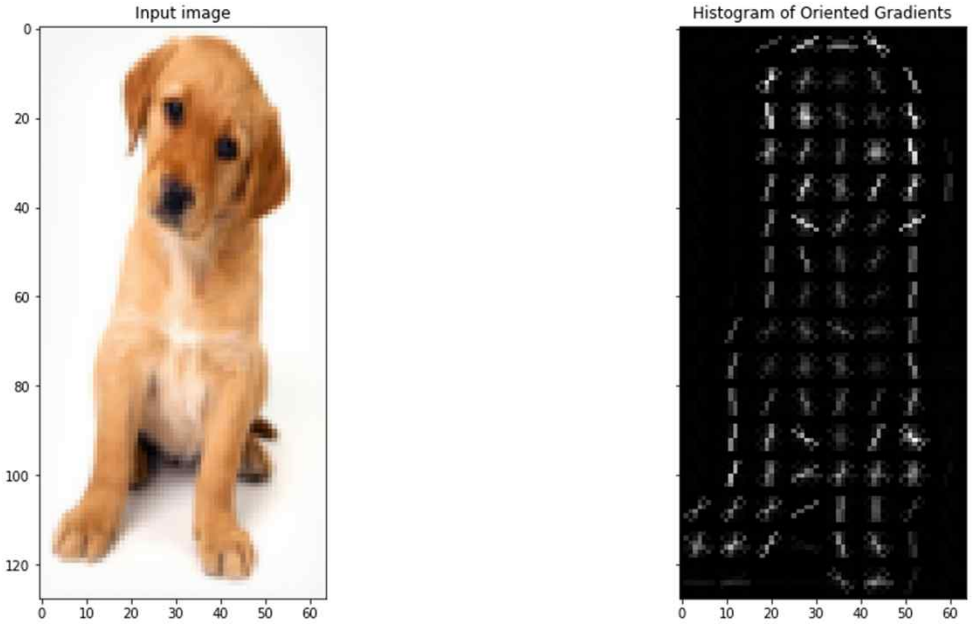
- Input image prediction

```
int main( int argc, const char** argv )
{
.....
  // Preprocess image
  Mat pre= preprocessImage(img);

 // Extract features
  vector<int> pos_top, pos_left;
  vector< vector<float> > features= ExtractFeatures(pre, &pos_left, &pos_top);

  for(int i=0; i< features.size(); i++){
    Mat trainingDataMat(1, 2, CV_32FC1, &features[i][0]);
    float result= svm->predict(trainingDataMat);

   stringstream ss;
   Scalar color;
    if(result==0){
      color= green; ss << "NUT";
    }
    else if(result==1){
      color= blue; ss << "RING" ;
    }
    else if(result==2){
      color= red; ss << "SCREW";
    }

    putText(img_output, ss.str(), Point2d(pos_left[i], pos_top[i]),
      FONT_HERSHEY_SIMPLEX, 0.4, color);
}
```



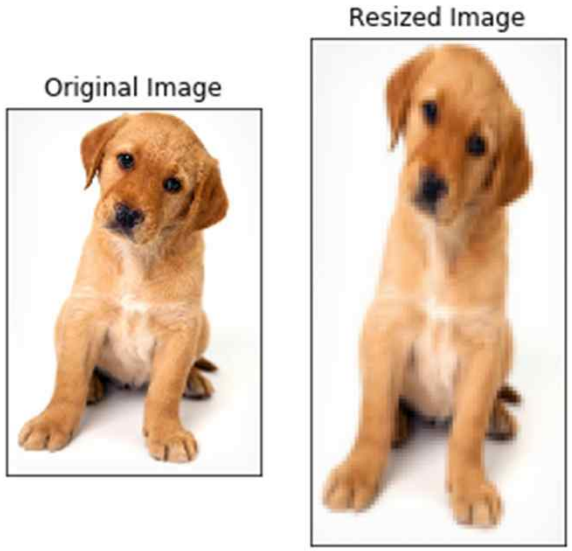
HOG

- HOG (Histogram of Oriented Gradients)
  - N.Dalal & B.Triggs, CVPR 2005
  - Local feature descrptor
  - SVM 과 결합하여 Object detection 에 많이 사용



HOG

1) Preprocess
   - Resize the image => width : height = 1: 2

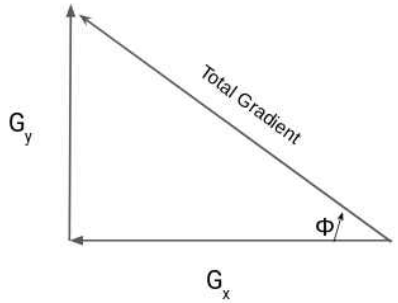


64 x 128

HOG

2) Calculation gradients (X & Y diection)

121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45



- Change in X direction(G_x) = 89 − 78 = 11
- Change in Y direction(G_y) = 68 − 56 = 8

- Gradient magnitude $\sqrt{[(G_x)^2+(G_y)^2]}$ =13.6
- Gradient orientation $\Phi = \text{atan}(Gy / Gx)$ =36(deg)

Sobel operator

HOG

3) Calculation of HOG

121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45

Gradient orientation=36(deg)

Frequency						1									
Angle	1	2	3	4 ...	35	36	37	38	39....	175	176	177	178	179	180

Frequency		1							
Bin	0	20	40	60	80	100	120	140	160

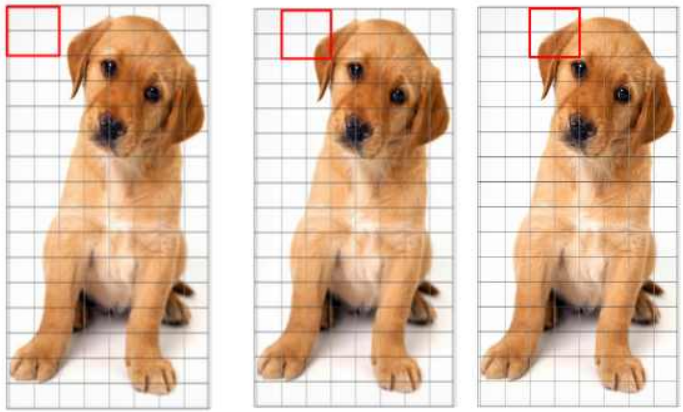
9 x 1 matrix

HOG

4) Calculate HOG in cells
   - 전체 영상을 여러 개의 Cell 로 분할
   - 각 cell 에 대하여 histogram 을 9 x 1 matrix 로 구함

5) Calculate HOG in blocks
   - 4 개의 cell 을 하나의 block 으로 구성
   - 각 cell 의 histogram 을 merge 하여 block 당 36 x 1 matrix 로 구함

cell



block

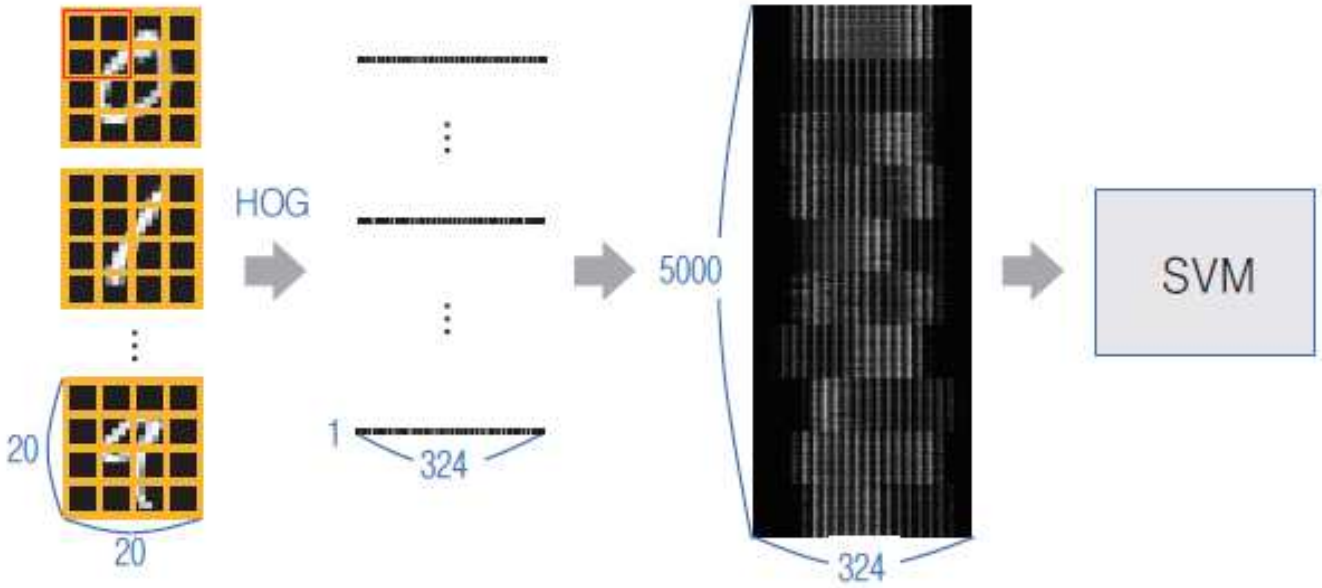
Total features = 7 x 15 x 36 x 1

HOG

- 필기체 숫자인식
  - 영상크기: 20 x 20
  - cell 크기: 5 x 5
  - Block 크기: 10 x 10

  ⇒ Block 수량: 3 x 3 = 9

  ⇒ 1 영상 당 9 x 36 x 1 = 324 features



OpenCV

- HOGDescriptor Class

```
HOGDescriptor::HOGDescriptor(Size _winSize, Size _blockSize, Size _blockStride,
            Size _cellSize, int _nbins, int _derivAperture = 1, double _winSigma = -1,
            HOGDescriptor::HistogramNormType _histogramNormType = HOGDescriptor::L2Hys,
            double _L2HysThreshold = 0.2, bool _gammaCorrection = false,
            int _nlevels = HOGDescriptor::DEFAULT_NLEVELS,
            bool _signedGradient = false)
```

• _winSize	검출 윈도우 크기
• _blockSize	블록 크기
• _blockStride	블록 이동 크기
• _cellSize	셀 크기
• _nbins	히스토그램 빈 개수
• _derivAperture	현재 사용되지 않습니다.
• _winSigma	가우시안 블러를 위한 표준 편차
• _histogramNormType	현재 사용되지 않습니다.
• _L2HysThreshold	L2-Hys 정규화 임계값
• _gammaCorrection	감마 보정 수행 여부
• _nlevels	검출 윈도우 증가 최대 횟수. 기본값은 64입니다.
• _signedGradient	그래디언트 방향 부호 사용 여부

```
HOGDescriptor hog(Size(20, 20), Size(10, 10), Size(5, 5), Size(5, 5), 9);
```

코드 15-6 SVM 알고리즘을 이용한 필기체 숫자 인식 [ch15/svmdigits]

```
01  #include "opencv2/opencv.hpp"
02  #include <iostream>
03
04  using namespace cv;
05  using namespace cv::ml;
06  using namespace std;
07
08  Ptr<SVM> train_hog_svm(const HOGDescriptor& hog);
09  void on_mouse(int event, int x, int y, int flags, void* userdata);
10
11  int main()
12  {
13  #if _DEBUG
14      cout << "svmdigits.exe should be built as Release mode!" << endl;
15      return 0;
16  #endif
17
18      HOGDescriptor hog(Size(20, 20), Size(10, 10), Size(5, 5), Size(5, 5), 9);
19
20      Ptr<SVM> svm = train_hog_svm(hog);
21
22      if (svm.empty()) {
23          cerr << "Training failed!" << endl;
24          return -1;
25      }
26
27      Mat img = Mat::zeros(400, 400, CV_8U);
28
29      imshow("img", img);
30      setMouseCallback("img", on_mouse, (void*)&img);
31
32      while (true) {
33          int c = waitKey();
34
```

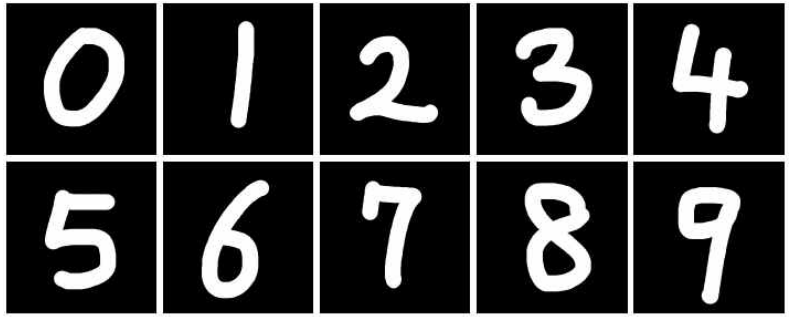
```
35          if (c == 27) {
36              break;
37          } else if (c == ' ') {
38              Mat img_resize;
39              resize(img, img_resize, Size(20, 20), 0, 0, INTER_AREA);
40
41              vector<float> desc;
42              hog.compute(img_resize, desc);
43
44              Mat desc_mat(desc);
45              int res = cvRound(svm->predict(desc_mat.t()));
46              cout << res << endl;
47
48              img.setTo(0);
49              imshow("img", img);
50          }
51      }
52
53      return 0;
54  }
```

Ptr<SVM> train_hog_svm(const HOGDescriptor& hog)

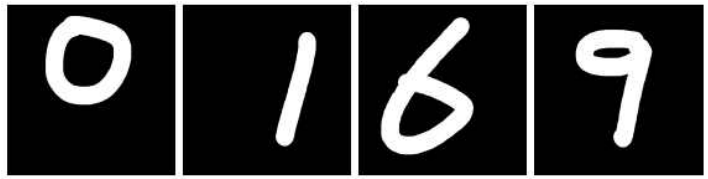
```
57  {
58      Mat digits = imread("digits.png", IMREAD_GRAYSCALE);
59
60      if (digits.empty()) {
61          cerr << "Image load failed!" << endl;
62          return 0;
63      }
64
65      Mat train_hog, train_labels;
66
67      for (int j = 0; j < 50; j++) {
68          for (int i = 0; i < 100; i++) {
69              Mat roi = digits(Rect(i * 20, j * 20, 20, 20));
70
71              vector<float> desc;
72              hog.compute(roi, desc);
73
74              Mat desc_mat(desc);
75              train_hog.push_back(desc_mat.t());
76              train_labels.push_back(j / 5);
77          }
78      }
79
80      Ptr<SVM> svm = SVM::create();
81      svm->setType(SVM::Types::C_SVC);
82      svm->setKernel(SVM::KernelTypes::RBF);
83      svm->setC(2.5);
84      svm->setGamma(0.50625);
85      svm->train(train_hog, ROW_SAMPLE, train_labels);
86
87      return svm;
88  }
```

HOG & SVM

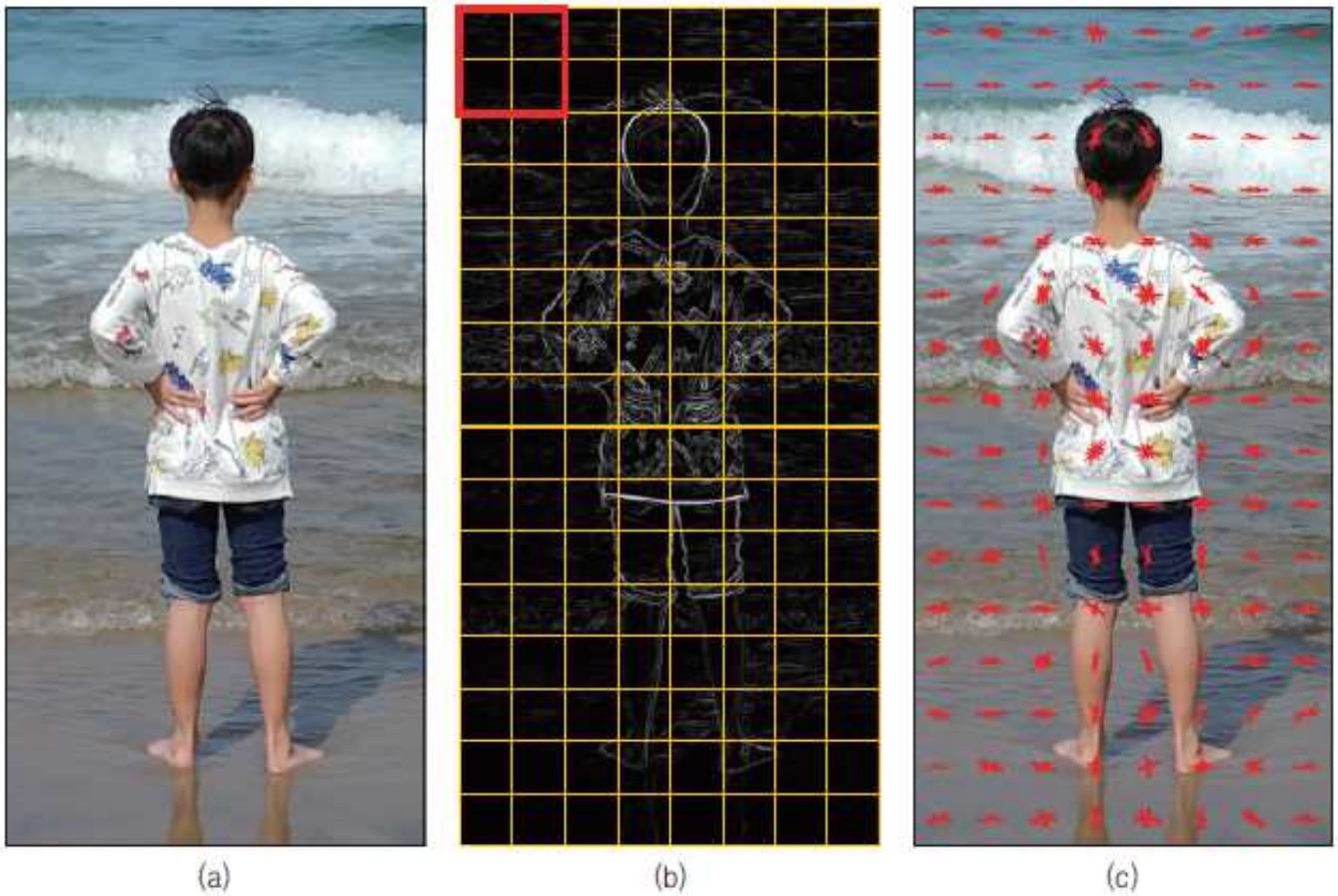
- 성공



- 실패



HOG & 보행자 검출



HOG & 보행자 검출

- HOGDescriptor::getDefaultPeopleDetector()
  - 미리 계산된 보행자 검출용 HOG Descriptor 정보 반환

```
static std::vector<float> HOGDescriptor::getDefaultPeopleDetector();
```

- 반환값 보행자 검출을 위해 훈련된 분류기 계수

- HOGDescriptor::setSVMDetector(InputArray svmdetector)
  - SVM classifier 를 위한 계수 등록

```
virtual void HOGDescriptor::setSVMDetector(InputArray svmdetector);
```

- svmdetector 선형 SVM 분류기를 위한 계수

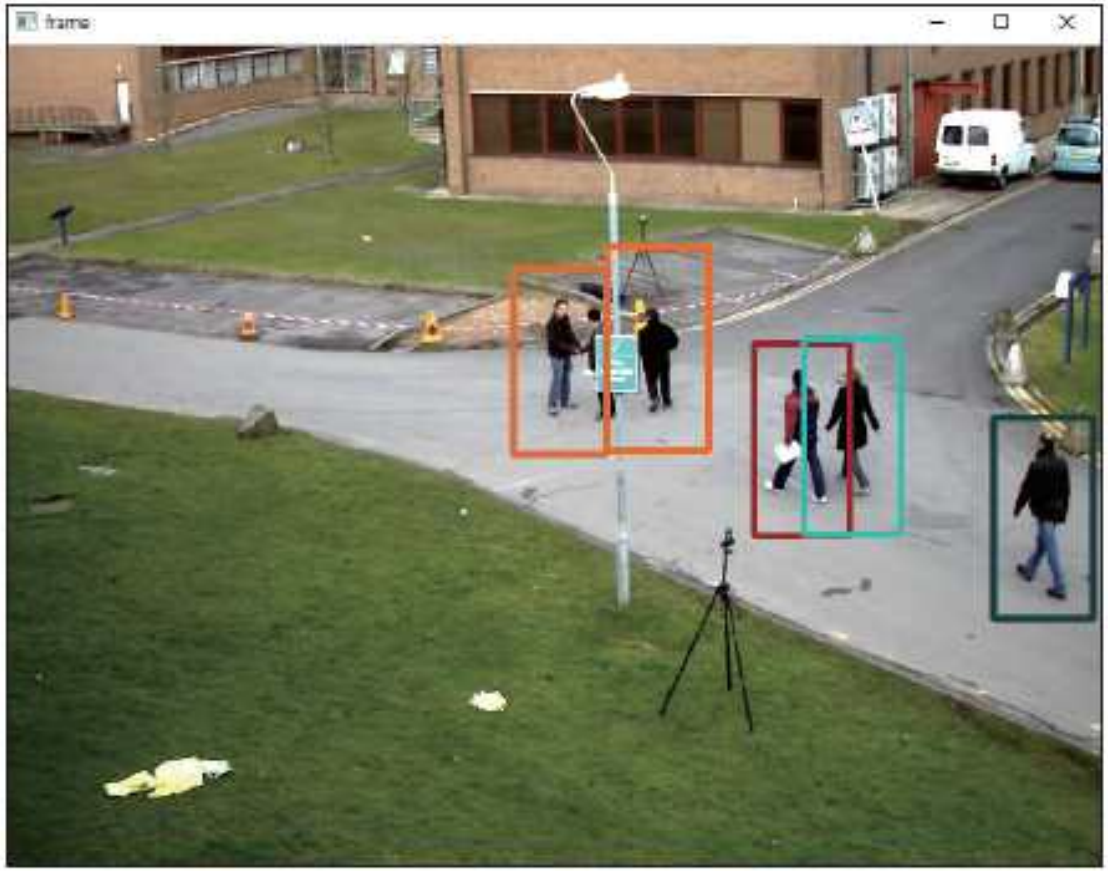
HOG & 보행자 검출

- HOGDescriptor::detectMultiScale()
  - 입력 영상에서 다양한 사각형 (Rect) 영역을 검출하여 반환

```
virtual void HOGDescriptor::detectMultiScale(InputArray img,
                                             std::vector<Rect>& foundLocations,
                                             std::vector<double>& foundWeights,
                                             double hitThreshold = 0,
                                             Size winStride = Size(),
                                             Size padding = Size(),
                                             double scale = 1.05,
                                             double finalThreshold = 2.0,
                                             bool useMeanshiftGrouping = false) const;
virtual void HOGDescriptor::detectMultiScale(InputArray img,
                                             std::vector<Rect>& foundLocations,
                                             double hitThreshold = 0,
                                             Size winStride = Size(),
                                             Size padding = Size(),
                                             double scale = 1.05,
                                             double finalThreshold = 2.0,
                                             bool useMeanshiftGrouping = false) const;
```

• img	입력 영상. CV_8UC1 또는 CV_8UC3
• foundLocations	(출력) 검출된 사각형 영역 정보
• foundWeights	(출력) 검출된 사각형 영역에 대한 신뢰도
• hitThreshold	특징 벡터와 SVM 분류 평면까지의 거리에 대한 임계값
• winStride	셀 윈도우 이동 크기. Size() 지정 시 셀 크기와 같게 설정합니다.
• padding	패딩 크기
• scale	검색 윈도우 크기 확대 비율
• finalThreshold	검출 결정을 위한 임계값
• useMeanshiftGrouping	겹쳐진 검색 윈도우를 합치는 방법 지정 플래그

HOG & 보행자 검출



HOG & 보행자 검출

코드 13-5 보행자 검출 예제 프로그램 [ch13/hog]

```
01  #include "opencv2/opencv.hpp"
02  #include <iostream>
03
04  using namespace cv;
05  using namespace std;
06
07  int main()
08  {
09      VideoCapture cap("vtest.avi");
10
11      if (!cap.isOpened()) {
12          cerr << "Video open failed!" << endl;
13          return -1;
14      }
15
16      HOGDescriptor hog;
17      hog.setSVMDetector(HOGDescriptor::getDefaultPeopleDetector());
18
19      Mat frame;
20      while (true) {
21          cap >> frame;
22          if (frame.empty())
23              break;
24
25          vector<Rect> detected;
26          hog.detectMultiScale(frame, detected);
27
28          for (Rect r : detected) {
29              Scalar c = Scalar(rand() % 256, rand() % 256, rand() % 256);
30              rectangle(frame, r, c, 3);
31          }
32
33          imshow("frame", frame);
34
35          if (waitKey(10) == 27)
36              break;
37      }
38
39      return 0;
40  }
```