Image Analysis II

Radovan Fusek
Object Detection (Analysis)
Haar, Integral Image, AdaBoost
What is Object Detection?

• It is clear that the images contain many objects of interest. The goal of the object detection systems is to find the location of these objects in the images (e.g. cars, faces, pedestrians).

• For example, the vehicle detection systems are crucial for traffic analysis or intelligent scheduling, the people detection systems can be useful for automotive safety, and the face detection systems are a key part of face recognition systems.
What is Object Detection?

- Output?
  - position of the objects
  - scale of the objects
Problems (Challenges)

- different views - different objects
Problems (Challenges)

- low quality images
Problems (Challenges)

- illumination + low quality

http://vis-www.cs.umass.edu/fddb/
The objects of interest can be described using various image information (e.g. shape, texture, colour). In the area of feature based detectors the image features are the carries of this information.

Many methods for extracting the image features that are able to describe the appearance of objects were presented, especially, the detectors that are based on the histograms of oriented gradients (HOG), Haar features, or local binary patterns (LBP) are dominant and they are considered as the state-of-the-art methods.
Object Detection/Recognition

- Haar
- HOG
- LBP
- SIFT, SURF
- CNNs

Traditional Approaches

KeyPoints

Deep Learning Approach

Practical examples using OpenCV + Dlib (https://opencv.org/, http://dlib.net/)
Sliding Window - Main Idea

Sliding Window - Main Idea

In general, the sliding window technique represents the popular and successful approach for object detection. The main idea of this approach is that the input image is scanned by a rectangular window at multiple scales. The result of the scanning process is a large number of various sub-windows. A vector of features is extracted from each sub-window. The vector is then used as an input for the classifier (e.g. SVM classifier).

During the classification process, some sub-windows are marked as the objects. Using the sliding window approach, the multiple positive detections may appear, especially around the objects of interest.
These detections are merged to the final bounding box that represents the resulting detection.

The classifier that determines each sub-window is trained over the training set that consists of positive and negative images.

The key point is to find what values (features) should be used to effectively encode the image inside the sliding window.
Detection Process

Feature Vector
(gradient, HOG, LBP, ...)

Detection Process

Feature Vector
(gradient, HOG, LBP, ...)

Trainable Classifier
(SVM, ANNs, ...)

Detection Process

• Typically, in the area of feature-based detectors, the detection algorithms consist of two main parts. The extraction of image features is the first part. The second part is created by the trainable classifiers that handle a final classification (object/non-object).

• The extraction of relevant features has a significant influence on the successfulness of detectors. The large number of features slows down the training and detection phases; on the other hand a very small number of features may not be able to describe the properties of object of interest.

• The quality of training set is also equally important.
Generating Training Set

- negative set - without the object of interest
- positive set
  - rotation
  - noise
  - Illumination
  - scale
- Haar
- HOG
- LBP
- SIFT, SURF
- CNNs

Traditional Approaches

KeyPoints

Deep Learning Approach
A Trainable System for Object Detection

CONSTANTINE PAPAGEORGIOUT AND TOMAS POGGIO
Center for Biological and Computational Learning, Artificial Intelligence Laboratory, MIT, Cambridge, MA, USA
cpupg@ai.mit.edu
tp@ai.mit.edu

Robust Real-Time Face Detection

PAUL VIOLA
Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
viola@microsoft.com

MICHAEL J. JONES
Mitsubishi Electric Research Laboratory, 201 Broadway, Cambridge, MA 02139, USA
mjones@merl.com

Histograms of Oriented Gradients for Human Detection

Navneet Dalal and Bill Triggs
INRIA Rhône-Alps, 655 avenue de l’Europe, Montbonnot 38334, France

Related Works
Haar Wavelet-based Descriptors

• The main idea behind the Haar-like features is that the features can encode the differences of mean intensities between the rectangular areas. For instance, in the problem of face detection, the regions around the eyes are lighter than the areas of the eyes; the regions bellow or on top of eyes have different intensities that the eyes themselves.

• These specific characteristics can be simply encoded by one two-rectangular feature, and the value of this feature can be calculated as the difference between the sum of the intensities inside the rectangles.
Haar Wavelet-based Descriptors

• The paper of Viola and Jones contributed to the popularity of Haar-like features. The authors proposed the object detection framework based on the image representation called the integral image combined with the rectangular features, and the AdaBoost algorithm.

• With the use of integral image, the rectangular features are computed very quickly. The AdaBoost algorithm helps to select the most important features.

• The features are used to train classifiers and the cascade of classifiers is used for reducing the computational time.
Haar Wavelet-based Descriptors

- faces have similar properties
  - eye regions are darker than the upper-cheeks
  - the nose bridge region is brighter than the eyes

https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html
Features

- Rectangular features

\[ F_{\text{Haar}} = E(R_{\text{white}}) - E(R_{\text{black}}) \]
Features

Different sets

Basic Haar set

Haar-like features

Extended Haar features

Borders

Lines

Center-surround

David Gerónimo: Haar-like Features and Integral Image Representation, Master in Computer Vision and Artificial Intelligence, 18th December 2009
Face Detection

- 24x24 sub-window aprox. 160,000 rectangular features
- How speed the computational speed?
  - decrease memory accesses

David Gerónimo: Haar-like Features and Integral Image Representation, Master in Computer Vision and Artificial Intelligence, 18th December 2009
Integral Image

Original image (i)  Integral image (ii)
Integral Image

Original image ($I$)

<table>
<thead>
<tr>
<th>7</th>
<th>7</th>
<th>7</th>
<th>4</th>
<th>8</th>
<th>4</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>

Integral image ($ii$)

<table>
<thead>
<tr>
<th>7</th>
<th>14</th>
<th>21</th>
<th>28</th>
<th>32</th>
<th>40</th>
<th>44</th>
<th>45</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>28</td>
<td>42</td>
<td>57</td>
<td>65</td>
<td>77</td>
<td>82</td>
<td>84</td>
</tr>
<tr>
<td>21</td>
<td>42</td>
<td>64</td>
<td>83</td>
<td>99</td>
<td>119</td>
<td>124</td>
<td>127</td>
</tr>
</tbody>
</table>

$133 = 99 + 109 - 83 + 8$
Integral Image

Original image ($I$) → Integral image ($ii$)
Integral Image

Integral image (ii)

\[ i_i = \text{sum}(A) \]
\[ i_{i_2} = \text{sum}(A) + \text{sum}(B) \]
\[ i_{i_3} = \text{sum}(A) + \text{sum}(C) \]
\[ i_{i_4} = \text{sum}(A) + \text{sum}(B) + \text{sum}(C) + \text{sum}(D) \]

\[ \text{sum}(D) = i_{i_4} + i_{i_1} - i_{i_2} - i_{i_3} \]
Integral Image

Integral image ($ii$)

\[
\begin{array}{cccccc}
7 & 14 & 21 & 28 & 32 & 40 & 44 & 45 \\
14 & 28 & 42 & 57 & 65 & 77 & 82 & 84 \\
21 & 42 & 64 & 83 & 99 & 119 & 124 & 127 \\
28 & 57 & 83 & 109 & 133 & 153 & 159 & 163 \\
35 & 72 & 105 & 138 & 166 & 194 & 201 & 206 \\
42 & 87 & 127 & 167 & 203 & 239 & 247 & 253 \\
\end{array}
\]

Original image ($i$)

\[
\begin{array}{cccccc}
7 & 7 & 7 & 7 & 4 & 8 & 4 & 1 \\
7 & 7 & 7 & 8 & 4 & 4 & 1 & 1 \\
7 & 7 & 8 & 4 & 3 & 3 & 4 & 1 \\
7 & 8 & 4 & 7 & 8 & 4 & 1 & 1 \\
7 & 8 & 7 & 7 & 1 & 8 & 1 & 1 \\
7 & 8 & 7 & 7 & 8 & 8 & 1 & 1 \\
\end{array}
\]

\[54 = 194 + 42 - 77 - 105\]
Feature Selection
Feature Selection

• AdaBoost (Adaptive Boost) is an iterative learning algorithm to construct a “strong” classifier as a linear combination of weighted simple “weak” classifiers

• weak classifier - each single rectangle feature (features as weak classifiers)

• during each iteration, each example/image receives a weight determining its importance
Feature Selection

- AdaBoost starts with a uniform distribution of “weights” over training examples.
Feature Selection

- AdaBoost starts with a uniform distribution of “weights” over training examples.

- Select the classifier with the lowest weighted error (i.e. a “weak” classifier)
AdaBoost starts with a uniform distribution of “weights” over training examples.

- Select the classifier with the lowest weighted error (i.e. a “weak” classifier)

- Increase the weights on the training examples that were misclassified.
AdaBoost starts with a uniform distribution of “weights” over training examples.

Select the classifier with the lowest weighted error (i.e. a “weak” classifier)

Increase the weights on the training examples that were misclassified.
AdaBoost starts with a uniform distribution of “weights” over training examples.

Select the classifier with the lowest weighted error (i.e. a “weak” classifier).

Increase the weights on the training examples that were misclassified.

(Repeat)
Feature Selection

- AdaBoost starts with a uniform distribution of “weights” over training examples.

- Select the classifier with the lowest weighted error (i.e. a “weak” classifier)

- Increase the weights on the training examples that were misclassified.

- (Repeat)
AdaBoost starts with a uniform distribution of “weights” over training examples.

Select the classifier with the lowest weighted error (i.e. a “weak” classifier)

Increase the weights on the training examples that were misclassified.

(Repeat)
AdaBoost starts with a uniform distribution of “weights” over training examples.

Select the classifier with the lowest weighted error (i.e. a “weak” classifier)

Increase the weights on the training examples that were misclassified.

(Repeat)
AdaBoost starts with a uniform distribution of “weights” over training examples.

Select the classifier with the lowest weighted error (i.e. a “weak” classifier)

Increase the weights on the training examples that were misclassified.

(Repeat)
AdaBoost starts with a uniform distribution of “weights” over training examples.

Select the classifier with the lowest weighted error (i.e. a “weak” classifier)

Increase the weights on the training examples that were misclassified.

(Repeat)
At the end, carefully make a linear combination of the weak classifiers obtained at all iterations.
At the end, carefully make a linear combination of the weak classifiers obtained at all iterations.
Cascade of Classifier

The idea of cascade classifier is to reject the non-face region as soon as possible.
The idea of cascade classifier is reject the non-face region as soon as possible.
The idea of cascade classifier is to reject the non-face region as soon as possible.
The idea of cascade classifier is to reject the non-face region as soon as possible.
The idea of cascade classifier is reject the non-face region as soon as possible.
Cascade of Classifier

The idea of cascade classifier is reject the non-face region as soon as possible.
The idea of cascade classifier is to reject the non-face region as soon as possible.
The idea of cascade classifier is to reject the non-face region as soon as possible.
Haar Features
Face Detection - Evaluation

http://vis-www.cs.umass.edu/fddb/
Face Detection - Evaluation

TP = number of true positives
FP = number of false positives
FN = number of false negatives
TN = number of true negatives

precision = TP/(TP+FP)
sensitivity = TP/(TP+FN)
F1 score (harmonic mean of precision and sensitivity) = 2 × precision × sensitivity/(precision + sensitivity)
Figure 8. *Matching detections and annotations.* In this image, the ellipses specify the face annotations and the five rectangles denote a face detector’s output. Note that the second face from left has two detections overlapping with it. We require a valid matching to accept only one of these detections as the true match, and to consider the other detection as a false positive. Also, note that the third face from the left has no detection overlapping with it, so no detection should be matched with this face. The blue rectangles denote the true positives and yellow rectangles denote the false positives in the desired matching.
• Since Viola and Jones popularized the Haar-like features for face detection, the Haarlike features and their modifications were used in many detection tasks (e.g. pedestrian, eye, vehicle).

• In the area of pedestrian detection, in [1], the authors presented the component-based person detector that is able to detect the occluded people in clustered scenes in static images. The detector uses the Haar-like features to describe the components of people (heads, legs, arms) combined with the SVM classifier. The Viola and Jones detection framework was successfully extended for moving-human detection in [2]. In [3], the authors proposed the method for estimating the walking direction of pedestrian.
The 3D Haar-like features for pedestrian detection were presented in [4]. The authors extend the classical Haar-like features using the volume filters in 3D space (instead of using rectangle filters in 2D space) to capture motion information. The 3D features are then combined with the SVM classifier. To compute the 3D Haar-like features using the integral image like the classical 2D features, the authors introduced Integral Volume that extends 2D integral image to the three dimensions.
Haar Features


The modified version of Haar-like features that more properly reflect the shape of the pedestrians than the classical Haar-like features.


The modified version of Haar-like features that more properly reflect the shape of the pedestrians than the classical Haar-like features.

The modified version of Haar-like features that more properly reflect the shape of the pedestrians than the classical Haar-like features.

Haar Features

The modified version of Haar-like features that more properly reflect the shape of the pedestrians than the classical Haar-like features.


http://mrl.cs.vsb.cz/
- Haar
- HOG
- LBP
- SIFT, SURF
- CNNs

Traditional Approaches

KeyPoints

Deep Learning Approach
Related Works

A Trainable System for Object Detection

CONSTANTINE PAPAGEORGIOU AND TOMAS POGGIN
Center for Biological and Computational Learning, Artificial Intelligence Laboratory, MIT, Cambridge, MA, USA
cpoga@ai.mit.edu
tp@ai.mit.edu

Robust Real-Time Face Detection

PAUL VIOLA
Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
viola@microsoft.com

MICHAEL J. JONES
Mitsubishi Electric Research Laboratory, 201 Broadway, Cambridge, MA 02139, USA
mjones@merl.com

Histograms of Oriented Gradients for Human Detection

Navneet Dalal and Bill Triggs
INRIA Rhône-Alps, 655 avenue de l’Europe, Montbonnot 38334, France

Papageorgiou (2000)


Dalal, Triggs (2005) cit. 10947
Pedestrian Detection - Challenges?
In recent years, the object detectors that are based on edge analysis that provides valuable information about the objects of interest were used in many detection tasks. In this area, the histograms of oriented gradients (HOG) [1] are considered as the state-of-the-art method.

In HOG, a sliding window is used for detection. The window is divided into small connected cells in the process of obtaining HOG descriptors. The histograms of gradient orientations are calculated in each cell. It is desirable to normalize the histograms across a large block of image. As a result, a vector of values is computed for each position of window. This vector is then used for recognition, e.g. by the Support Vector Machine classifier.
Basic Steps:

- In HOG, a sliding window is used for detection.
- The window is divided into small connected cells.
- The histograms of gradient orientations are calculated in each cell.
- Support Vector Machine (SVM) classifier.
Histories of Oriented Gradients (HOG)

- For gradient computation, the image without Gaussian smoothing is filtered with the [1, 0, -1] kernel to compute the horizontal and vertical derivatives.
- Then the derivatives are used to compute the magnitude of the gradient and orientation.

\[
D_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad D_y = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}
\]

\[
I_x = I \ast D_x \quad \text{and} \quad I_y = I \ast D_y
\]

magnitude of the gradient is \( |G| = \sqrt{I_x^2 + I_y^2} \)

orientation of the gradient is given by: \( \theta = \arctan \frac{I_y}{I_x} \)
In the next step, the image is divided into the cells and the cell histograms are constructed. The histogram bins are spread over 0 to 180 degrees or 0 to 360 degrees. The corresponding histogram bin is found for each pixel inside the cell. Each pixel contributes a weighted vote for its corresponding bin. The pixel contribution can be the gradient magnitude.

Next step represents contrast normalization. For this purpose, the cells are grouped into the large blocks (i.e. $2 \times 2$ cells are considered as blocks). The histograms are normalized within the blocks (e.g. using L2-norm). In the paper, the two main block geometries are presented; rectangular and circular.

The final HOG descriptor is represented by histogram vectors of all blocks within the detection window.
Dalal and Trigs experimented with the size of detection window and they suggested the rectangular window with the size $64 \times 128$ pixels.

They also tried to reduce the size of the window to $48 \times 112$ pixels. Nevertheless, they obtained the best detection result with the size $64 \times 128$ pixels.

Histograms of Oriented Gradients (HOG)
Histomgrams of Oriented Gradients (HOG)

Blocks, Cells:

- 8 x 8 cell
- 16 x 16 block – overlap
- normalization within the blocks

**Final Vector:** Collect HOG blocks into vector
The classical HOG descriptors suffer from the large number of features, which causes that the training and detection phases can be time consuming. The sufficient amount of training data is also needed to find a separating hyperplane by the SVM classifier.

Sometimes, it is desirable to use the methods for the dimensionality reduction of feature vector. In addition the that, the classical HOG descriptors are not rotation invariant.

These shortcomings became the motivation for creating many variations of HOG-based detectors. Many methods and applications based on HOG were presented in recent years.
In [1], the authors applied the principal component analysis (PCA) to the HOG feature vector to obtain the PCA-HOG vector. This vector contains the subset of HOG features and the vector is used as an input for the SVM classifier. Their method was used for pedestrian detection with the satisfactory results.

Felzenszwalb et al. proposed the part-based detector that is based on HOG. In this method, the objects are represented using the mixtures of deformable HOG part models and these models are trained using a discriminative method (see following image). This method obtained excellent performance for object detection tasks [2, 3].

An example of person detection using a part model. The model is defined by the coarse global template that covers the entire object and higher resolution part templates. The templates represent the histogram of oriented gradient [2].

Practical Example – Detection + Recognition

Consider the following problem: Find and recognize two following lego kits
OpenCV (Open Source Computer Vision Library: [http://opencv.org](http://opencv.org)) is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms. The document describes the so-called OpenCV 2.x API, which is essentially a C++ API, as opposed to the C-based OpenCV 1.x API. The latter is described in opencv1x.pdf.

OpenCV has a modular structure, which means that the package includes several shared or static libraries. The following modules are available:

- **Core functionality** - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.
- **Image processing** - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.
- **video** - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.
- **calib3d** - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
- **features2d** - salient feature detectors, descriptors, and descriptor matchers.
- **objdetect** - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).
- **highgui** - an easy-to-use interface to simple UI capabilities.
- **Video I/O** - an easy-to-use interface to video capturing and video codecs.
- **gpu** - GPU-accelerated algorithms from different OpenCV modules.
- ... some other helper modules, such as FLANN and Google test wrappers, Python bindings, and others.

The further chapters of the document describe functionality of each module. But first, make sure to get familiar with the common API concepts used thoroughly in the library.
Detection step
HOG+SVM (OpenCV)

```c
// Set up training data
int labels[4] = {1, -1, -1, -1};
Mat labelsMat(4, 1, CV_32SC1, labels);

float trainingData[4][2] = {{501, 10}, {255, 10}, {501, 255}, {10, 501}};
Mat trainingDataMat(4, 2, CV_32FC1, trainingData);

// Set up SVM's parameters
SVM::Params params;
params.svmType = SVM::C_SVC;
params.kernelType = SVM::LINEAR;
params.termCrit = TermCriteria(TermCriteria::MAX_ITER, 100, 1e-6);

// Train the SVM
Ptr<SVM> svm = StatModel::train<SVM>(trainingDataMat, ROW_SAMPLE, labelsMat, params);
```

https://docs.opencv.org/3.1.0/d1/d73/tutorial_introduction_to_svm.html
Avenger
Negative Set
```cpp
int blockSize = 16;
int cellSize = 8;
int strideSize = 8;
int winSize = 64;

//HOGDescriptor hog;
HOGDescriptor my_hog(
    cv::Size(winSize,winSize), //winSize
    cv::Size(blockSize,blockSize), //blocksize
    cv::Size(strideSize,strideSize), //blockStride,
    cv::Size(cellSize,cellSize), //cellSize,
    9, //nbins,
);

//SVM
Ptr<SVM> svm = StatModel::load<SVM>( classifierName );
std::vector< float > hog_detector;
//get the support vectors
get_svm_detector( svm, hog_detector );
//set SVM
my_hog.setSVMDetector( hog_detector );
std::vector<Rect> positivesAll;
my_hog.detectMultiScale( frameGray, positivesAll, 0,
    Size(0,0), Size(0,0), 1.1, 4);

https://github.com/opencv/opencv/blob/master/samples/cpp/train_HOG.cpp
```
- Haar
- HOG
- LBP
- SIFT, SURF
- CNNs

Traditional Approaches

KeyPoints

Deep Learning Approach
Related Works

Face Description with Local Binary Patterns: Application to Face Recognition
Timo Ahonen, Student Member, IEEE, Abdennour Hadid, and Matti Pietikäinen, Senior Member, IEEE

Face Detection Based on Multi-Block LBP Representation
Lun Zhang, Ruifeng Chu, Shiming Xiang, Shengcai Liao, Stan Z. Li
Center for Biometrics and Security Research & National Laboratory of Pattern Recognition
Institute of Automation, Chinese Academy of Sciences
95 Zhongguancun Donglu, Beijing 100080, China

Face detection using simplified Gabor features and hierarchical regions in a cascade of classifiers
Li Xiaohua1,2, Kin-Man Lam2,3, Shen Lansun1, Zhou Jiliu3
1Department of Computer Science, Shandong University, Chengdu 610064, China
2Centre for Signal Processing, Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong, Kowloon, Hong Kong
3Signal and Information Processing Lab, Beijing University of Technology, Beijing 100022, China
LBP - Local Binary Patterns

- Were introduced by Ojala et al. for the texture analysis.

- The local binary patterns (LBP) were introduced by Ojala et al. [2, 3] for the texture analysis. The main idea behind LBP is that the local image structures (micro patterns such as lines, edges, spots, and flat areas) can be efficiently encoded by comparing every pixel with its neighboring pixels. In the basic form, every pixel is compared with its neighbors in the $3 \times 3$ region. The result of comparison is the 8-bit binary number for each pixel; in the 8-bit binary number, the value 1 means that the value of center pixel is greater than the neighbor and vice versa. The histogram of these binary numbers (that are usually converted to decimal) is then used to encode the appearance of region.
LBP - Local Binary Patterns

- The important properties of LBP are the resistance to the lighting changes and a low computational complexity.

- Due to their properties, LBP were used in many detection tasks, especially in facial image analysis [1, 4].

https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html#local-binary-patterns.histograms
LBP - Local Binary Patterns


LBP - Local Binary Patterns

- Robust to monotonic changes in illumination
LBP - Local Binary Patterns

LBP - Local Binary Patterns

1. Sample

2. Difference

3. Threshold

\[ 1 \times 1 + 1 \times 2 + 1 \times 4 + 1 \times 8 + 0 \times 16 + 0 \times 32 + 0 \times 64 + 0 \times 128 = 15 \]

LBP - Local Binary Patterns

'Uniform' patterns (P=8)

Examples of 'nonuniform' patterns (P=8)
LBP - Local Binary Patterns

Spot

Spot/Flat

Line

Edge

Corner

https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html#local-binary-patterns-histograms
LBP - Local Binary Patterns
LBP - Local Binary Patterns

In [1], LBP were used for solving the face detection problem in low-resolution images. In this approach, the $19 \times 19$ face images are divided into the 9 overlapping regions in which the LBP descriptors are computed. Additionally, the LBP descriptors are extracted from the whole $19 \times 19$ image. The descriptors are then used to create the feature vector, and the SVM classifier with a polynomial kernel is used for the final classification.

LBP - Local Binary Patterns

LBP - Local Binary Patterns

Multi-block local binary patterns (MB-LBP) for face detection and recognition were proposed in [1, 2]. In this method, the authors encode the rectangular regions by the local binary pattern operator and the Gentle AdaBoost is used for feature selection. Their results showed that MBLBP are more distinctive than the Haar-like features and the original LBP features.


LBP - Local Binary Patterns


LBP - Local Binary Patterns

The paper of Tan and Triggs [2] proposed the face recognition method with robust preprocessing based on the difference of Gaussian image filter combined with LBP in which the binary LBP code is replaced by the ternary code to create local ternary patterns (LTP).

LBP were also successfully used for the facial expression analysis. The coarse-to-fine classification scheme with LBP combined with the k-nearest neighbor classifier that carries out the final classification was proposed in [1].

The comprehensive study of facial expression recognition using LBP was proposed in [78], the survey of facial image analysis using LBP was presented in [38].

- Haar
- HOG
- LBP
- SIFT, SURF
- CNNs

**Traditional Approaches**

**KeyPoints**

**Deep Learning Approach**
KeyPoints

The most of the previously mentioned methods for object description were based on the fact that the descriptors were extracted over the whole image (sliding window) that was usually divided into the overlap or non-overlap regions. Inside these regions, the descriptors were calculated and combined to the final feature vector that was used as an input for the classifier.

In this lecture, we present the state-of-the-art descriptors that are based on the fact that the regions (within which the descriptors are extracted) are selected using the keypoint detectors.
KeyPoints - SIFT

One of the most popular descriptors based on the interest points was proposed by David Lowe [1, 2, 3]. The method is called scale invariant feature transform (SIFT).

The idea of the SIFT descriptor is that the interesting points (keypoints) of the objects can be extracted to provide the key information about the objects. The gradient magnitude and orientation are computed around the keypoint location; the histograms are then summarized over subregions (see following image). The keypoints are extracted from the reference image (that contains the object of interest) and also from the target image (that possibly contains the object of interest). The extracted keypoints are matched to find similarity between the images.
KeyPoints - SIFT

- Keypoints are invariant in the terms of:
  - Scale
  - Orientation
  - Position
  - Illumination
  - Partially Occlusion
KeyPoints - SIFT

An example of SIFT keypoint descriptor in which the gradient orientation and gradient magnitude around each interest point are used [3].


The speeded up robust feature (SURF) descriptor by Bay et al. [1, 2] is also one of the widely used keypoint descriptors. In this method, the Hessian matrix-based measure is used to find the points of interest. The sum of the Haar-wavelet responses within the neighborhood of interest point is calculated. The authors also use the fast calculation via the integral image thanks to which SURF is faster than SIFT.


A very fast method called binary robust independent elementary features (BRIEF) was proposed by Calonder et al. [1]. The authors reported that the method outperforms SURF in the term of speed, and the recognition rate in many cases. In BRIEF, a binary string that contains the results of intensity differences of pixels are used and the descriptor similarity is evaluated using the Hamming distance. In [2], the authors proposed another binary descriptor with rotation and noise invariant properties called oriented fast and rotated BRIEF (ORB).


Leutenegger et al. [1] proposed binary robust invariant scalable keypoints (BRISK). The method provides both scale and rotation invariance. BRISK is a binary descriptor like BRIEF and ORB, it means that the binary string that represents a region around the keypoint is composed. In BRISK, a concentric circle pattern of points near to the keypoint is used (see following image). In this pattern, the blue circles represent the sampling locations and Gaussian blurring is computed to be less sensitive to noise; the radius of red circles denotes a standard deviation of blurring kernel. The standard deviation of the Gaussian kernel is increased with the increasing distance from the feature center to avoid aliasing effects. The final descriptor is determined by the comparison of sample points.
KeyPoints – BRISK

KeyPoints – FREAK

In [1], the authors proposed the fast retina keypoint (FREAK) descriptor that also uses the binary strings. The method is biologically inspired by a human visual system; more exactly by the retina. In this paper, the authors proposed a retinal sampling pattern; The following image shows the topology of this pattern. The pattern is divided into the areas (foveal, fovea, parafoveal, and perifoveal) similar to the human retina. In this pattern, the pixels are overlapped and concentrated near to the center. The binary strings is computed by comparing the point pairs of image intensities within the pattern.

KeyPoints – FREAK

FREAK sampling pattern [1]

KeyPoints - Example

The goal is to find image KeyPoints that are invariant in the terms of scale, orientation, position, illumination, partially occlusion.
KeyPoints - Example

https://docs.opencv.org/3.1.0/d6f/tutorial_feature_flann_matcher.html
- Haar
- HOG
- LBP
- SIFT, SURF
- CNNs

Traditional Approaches

KeyPoints

Deep Learning Approach
Recognition

Alien vs. Avenger
CNNs – Main Steps (LeNet)

1. Convolution
2. Non Linearity (ReLU)
3. Pooling or Sub Sampling
4. Classification (Fully Connected Layer)

https://www.clarifai.com/technology
1. Convolution

Input Image

Mask/Filter

https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
1. Convolution

Multiply the image pixels by pixels of the filter, then sum the results.
1. Convolution

Sharpen: $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

Blur: $\begin{bmatrix} 0 & 0.2 & 0 \\ 0.2 & 0.2 & 0.2 \\ 0 & 0.2 & 0 \end{bmatrix}$

Horizontal Motion Blur: $\begin{bmatrix} 0 & 0 & 0 \\ 0.2 & 0.2 & 0.2 \\ 0 & 0 & 0 \end{bmatrix}$

Edge detect: $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

http://dimitroff.bg/image-filtering-your-own-instagram/
1. Convolution

http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/
1. Convolution

- Before training, we have many filters/kernels
  - Filter values are randomized
  - Depth of this conv. layer corresponds to the number of filters we use for the convolution operation

- The filters are learned during the training

http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/
2. Non Linearity (ReLU)

- ReLU is used after every Convolution operation
- The goal of this step is to replace all negative pixels by zero in the feature map

![Image showing the effect of ReLU](http://mloss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf)
3. Pooling

(Subsampling or downsampling)

- The goal of this step is to reduce the dimensionality of each feature map but preserve important informations

- Operations: e.g. Sum, Average, Max
3. Pooling

(Subsampling or downsampling)

- Common way is a pooling layer with filters of size 2x2 applied with a stride of 2

![Pooling Diagram]

• Common way is a pooling layer with filters of size 2x2 applied with a stride of 2
Convolution layers and Pooling layers can be repeated any number of times in a single ConvNet.
4. Classification

- Multi Layer Perceptron
- The number of filters, filter sizes, architecture of the network etc. are fixed and do not change during training process.
- Only the values of the filter matrix and connection weights get updated.

[Diagram of a convolutional neural network]

http://cs231n.github.io/convolutional-networks/
CovNet Architectures

- LeNet (1990s)
- AlexNet (2012)
- ZF NET (2013)
- GoogLeNet (2014)
- VGGNet (2014)
- ResNets (2015)
- DenseNet (2016)

https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
Dlib contains a wide range of machine learning algorithms. All designed to be highly modular, quick to execute, and simple to use via a clean and modern C++ API. It is used in a wide range of applications including robotics, embedded devices, mobile phones, and large high performance computing environments. If you use dlib in your research please cite:
```cpp
using net_type = loss_multiclass_log<
    fc<10,
    relu<fc<84,
    relu<fc<120,
    max_pool<2,2,2,2,relu<con<16,5,5,1,1,
    max_pool<2,2,2,2,relu<con<6,5,5,1,1,
    input<matrix<unsigned char>>
  >>>>>>>>>>>>>>
```

http://dlib.net/dnn_introduction_ex.cpp.html
using net_type = loss_multiclass_log<
    fc<10,
    relu<fc<84,
    relu<fc<120,
    max_pool<2,2,2,2,relu<con<16,5,5,1,1,
    max_pool<2,2,2,2,relu<con<6,5,5,1,1,
    input<matrix<unsigned char>>
    >>>>>>>>>>>>>>>>>;

http://dlib.net/dnn_introduction_ex.cpp.html
using net_type = loss_multiclass_log<
fcc10,
relu<fc<84,
relu<fc<120,
max_pool<2,2,2,2,relu<con<16,5,5,1,1,
max_pool<2,2,2,2,relu<con<6,5,5,1,1,
input<matrix<unsigned char>>

http://dlib.net/dnn_introduction_ex.cpp.html
Recognition step CNNs (Dlib)

using net_type = loss_multiclass_log<
    fc<10,
    relu<fc<84,
    relu<fc<120,
    max_pool<2,2,2,2,relu<con<16,5,5,1,1,
    max_pool<2,2,2,2,relu<con<6,5,5,1,1,
    input<matrix<unsigned char>>
>>>;

6 conv. filters
5x5 filter size
1x1 stride
+ReLU

http://dlib.net/dnn_introduction_ex.cpp.html
using net_type = loss_multiclass_log<
    fc<10,
    relu<fc<84,
    relu<fc<120,
    max_pool<2,2,2,2,relu<con<16,5,5,1,1,
    max_pool<2,2,2,2,relu<con<6,5,5,1,1,
    input<matrix<unsigned char>>
    >>>>>>>>>>>>>>;

Recognition step CNNs
(Dlib)

MAX POOLING
2x2 window
2x2 stride

http://dlib.net/dnn_introduction_ex.cpp.html
Recognition step CNNs (Dlib)

Using

\[
\text{net\_type} = \text{loss\_multiclass\_log}\left<
\begin{array}{c}
\text{fc<10}, \\
\text{relu<fc<84}, \\
\text{relu<fc<120}, \\
\text{max\_pool<2,2,2,2}, \\
\text{relu<con<16,5,5,1,1}, \\
\text{max\_pool<2,2,2,2}, \\
\text{relu<con<6,5,5,1,1}, \\
\text{input<matrix<unsigned\_char>>}
\end{array}\right>
\]

16 conv. filters
5x5 filter size
1x1 stride
+ReLU

http://dlib.net/dnn_introduction_ex.cpp.html
**Recognition step CNNs (Dlib)**

Using `net_type = loss_multiclass_log<
fc<10,
relu<fc<84,
relu<fc<120,
max_pool<2,2,2,2,relu<con<16,5,5,1,1,
max_pool<2,2,2,2,relu<con<6,5,5,1,1,
input<matrix<unsigned char>>
>>>>>>>>>>>>>>;'>

Max pooling
2x2 window
2x2 stride

Input Image

Convolution + ReLU

Pooling

Convolution + ReLU

Pooling

Fully Connected

http://dlib.net/dnn_introduction_ex.cpp.html
using net_type = loss_multiclass_log<

```
fc<10, relu<fc<84, relu<fc<120,
max_pool<2,2,2,2,relu<con<16,5,5,1,1,
max_pool<2,2,2,2,relu<con<6,5,5,1,1,
input<matrix<unsigned char>>
```

Fully connected layer
120 neurons
84 neurons
10 outputs/classes
multiclass classification

Recognition step CNNs (Dlib)

http://dlib.net/dnn_introduction_ex.cpp.html
Recognition step CNNs (Dlib)

1 // network instance
2 net_type net;
3
4 // mini-batch stochastic gradient descent
5 //dnn_trainer<net_type> trainer(net, sgd(), {0,1}); //{0,1} - will use two GPU
6 dnn_trainer<net_type> trainer(net);
7 trainer.set_learning_rate(0.01);
8 trainer.set_min_learning_rate(0.0001);
9 trainer.set_mini_batch_size(160);
10 trainer.set_iterations_without_progress_threshold(500);
11 trainer.set_max_num_epochs(100);
12 trainer.beVerbose();
13 //train
14 trainer.train(train_images, train_labels);
15 // save
16 serialize("LeNet.dat") << net;
1 //Load image using OpenCV
2 Mat frame;
3 frame = imread( "my_img.png", 1 );
4 cvtColor( frame, frame, COLOR_BGR2GRAY );
5 medianBlur(frame, frame, 5);

7 //OpenCV Mat to Dlib
8 cv_image<unsigned char> cimg(frame);
9 matrix<unsigned char> dlibFrame = dlib::mat(cimg);

11 //prediction using CNN
12 unsigned long predict_label = net(frame);
Exercise

DETECTING FREE/OCCUPIED PLACES IN PARKING LOTS

Motivation:
The vehicle detection systems using images have been very useful in the recent years. Especially nowadays in the cities, the increasing number of vehicles brings a major problem. The car detection systems can be important, especially for drivers who are looking for vacant spaces in the parking lots, for traffic analysis, for intelligent scheduling, for smart cities and so on.

Input Data:
The training data with a basic template (C++/OpenCV) can be found in the following link:

http://mrl.cs.vsb.cz/data/parking.zip
Exercise

description of template:

- training and testing data are in the “testImages” and “trainImages” folders

- each image is named as free_xx.png or full_xx.png (the name of the images represents the state of parking space)

- functions for loading training/testing images are already implemented - `train_parking()`, `test_parking()`

- **the training and prediction steps are missing** - You can use any available libraries to solve this detection task. The use of the provided main.cpp template is not required.
Exercise

Output:

If you successfully run the template, you obtain this output. It means that the accuracy of the detector is approx. 32%. The accuracy is low because each parking space is labeled as occupied - line 82 in main.cpp. The goal is to implement better prediction approach.
Exercise

Hints:

Since we want to label each parking space as free (0) or occupied (1), this recognition problem can be solved using classical binary classifiers (SVM, neural networks). To train the classifiers, you can use the provided training data in the “trainImages” folder. As the input for the classifiers, you can use the whole image or you can use feature extraction approaches (e.g. histograms of oriented gradients, local binary patterns). Alternatively, you can skip the training process and use simple color or gradient information for example. In that case, you can use only the test_parking() function without the training.

The provided template is based on the OpenCV library [https://opencv.org/](https://opencv.org/)
Installation in Linux: [https://www.learnopencv.com/install-opencv3-on-ubuntu/](https://www.learnopencv.com/install-opencv3-on-ubuntu/)
Installation in Windows: [https://www.learnopencv.com/install-opencv3-on-windows/](https://www.learnopencv.com/install-opencv3-on-windows/)
Installation in MacOS: [https://www.learnopencv.com/install-opencv3-on-macos/](https://www.learnopencv.com/install-opencv3-on-macos/)
Simple install for Windows without cmake using NuGet:
[https://www.nuget.org/packages/opencv.win.native/320.1.1-vs141](https://www.nuget.org/packages/opencv.win.native/320.1.1-vs141)
Exercise

Hints:

Alternatively, you can install OpenCV from the Ubuntu or Debian repository:

```
sudo apt-get install libopencv-dev python3-opencv
```

You can find the several tutorials in the following link: [https://docs.opencv.org/3.4.2/d9/df8/tutorial_root.html](https://docs.opencv.org/3.4.2/d9/df8/tutorial_root.html)

**Dlib** library represents another option how to solve this detection problem

- Installation in Windows: [https://www.learnopencv.com/install-dlib-on-windows/](https://www.learnopencv.com/install-dlib-on-windows/)
- Installation in Linux: [https://www.learnopencv.com/install-dlib-on-ubuntu/](https://www.learnopencv.com/install-dlib-on-ubuntu/)
- Installation in MacOS: [https://www.learnopencv.com/install-dlib-on-macos/](https://www.learnopencv.com/install-dlib-on-macos/)

You can follow this tutorial: [http://dlib.net/dnn_introduction_ex.cpp.html](http://dlib.net/dnn_introduction_ex.cpp.html)

You can also use **Keras**, **Caffe**, **TensorFlow**, etc.