Applied Computer Vision

David Vernon
Carnegie Mellon University Africa

vernon@cmu.edu
www.vernon.eu

Lecture 20

Object Recognition

Histogram of Oriented Gradients (HOG)
Person detection

Histogram of Oriented Gradients

HOG person detector

- HOG uses a global feature to describe a person rather than a collection of local features
  - The entire person is represented by a single feature vector
    (not by many feature vectors representing smaller parts of the person)
- A Support Vector Machine (SVM) was trained to to recognize/classify HOG descriptors of people
  - Positive examples (images with people)
  - Negative examples (images without people)

Credit: Chris McCormick

Histogram of Oriented Gradients

HOG person detector

- Uses a sliding detection window which is moved around the image
- At each position of the detector window, a HOG descriptor is computed for the detection window
- This descriptor is then shown to the trained SVM, which classifies it as either “person” or “not a person”
- To recognize persons at different scales, the image is subsampled to multiple sizes. Each of these subsampled images is searched

Credit: Chris McCormick

Histogram of Oriented Gradients

The HOG person detector was introduced by Dalal and Triggs at the CVPR conference in 2005

- The original paper is available here
- The original training data set is available here

Credit: Chris McCormick

Histogram of Oriented Gradients

Gradient Histograms

- Uses a detection window that is 64 pixels wide by 128 pixels tall

- Below are some of the original images used to train the detector, cropped in to the 64x128 window

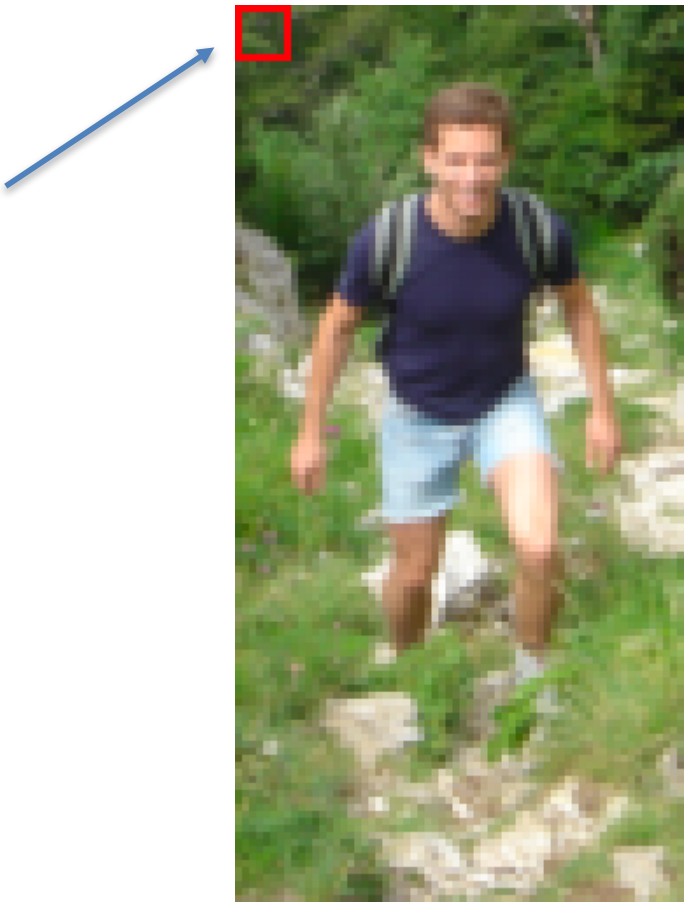
Credit: Chris McCormick

Histogram of Oriented Gradients

Gradient Histograms

- The HOG descriptor is constructed using 8x8 pixel cells within the detection window

- Cells are organized into overlapping blocks (more later)

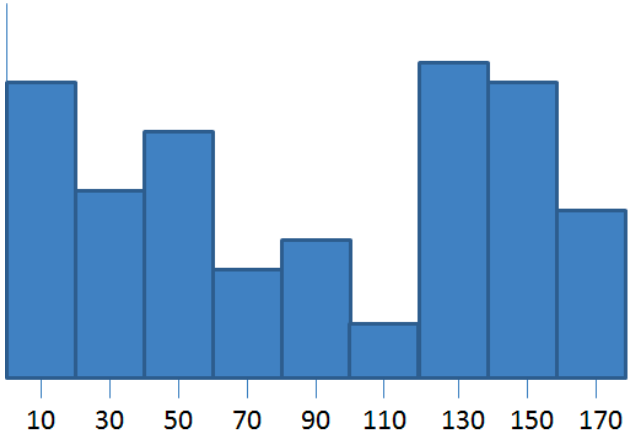


Credit: Chris McCormick

Histogram of Oriented Gradients

Gradient Histograms

- Within each cell, compute the gradient vector at each pixel
- Generate a 9-bin histogram from the 64 gradient vectors (8x8 pixel cell)
- The Histogram ranges from 0 to 180 degrees:* 20 degrees per bin



Credit: Chris McCormick

*Dalal and Triggs used "unsigned gradients" so that orientations only ranged from 0 to 180 degrees instead of 0 to 360

Histogram of Oriented Gradients

Gradient Histograms

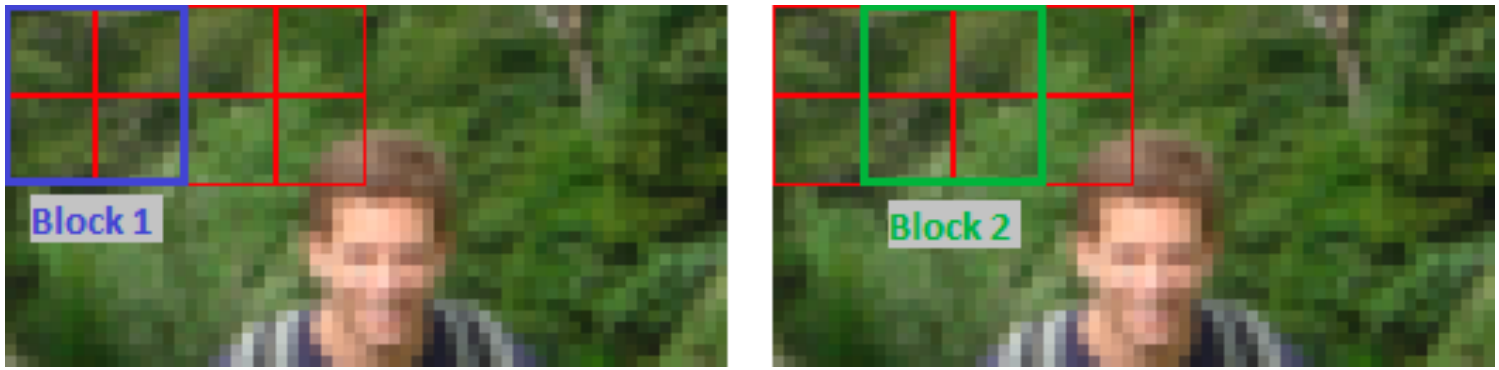
- The contribution of each gradient vector to the histogram is weighted by the magnitude of the vector
  - Stronger gradients have a bigger impact on the histogram
- The contribution is split between the two closest bins
  - For example, if a gradient vector has an angle of 85 degrees
    - Add 1/4th of its magnitude to the bin centered at 70 degrees
    - Add 3/4ths of its magnitude to the bin centered at 90

Credit: Chris McCormick

Histogram of Oriented Gradients

Gradient Histograms

- Normalize histogram
  - Rather than normalize each histogram individually, the cells are first grouped into blocks of 2x2 cells and normalized based on all histograms in the block
  - The blocks have 50% overlap

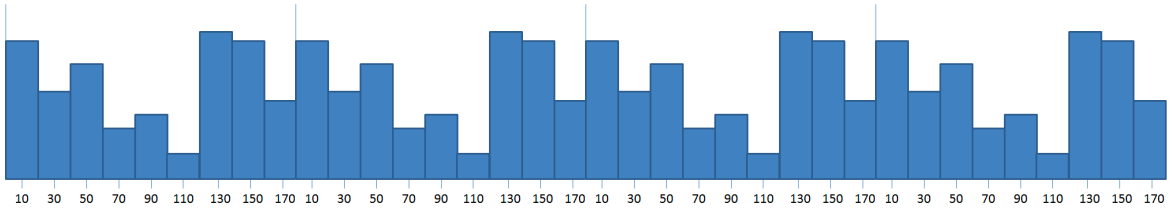


Credit: Chris McCormick

Histogram of Oriented Gradients

Gradient Histograms

- Normalize histogram
  - Block normalization is performed by concatenating the histograms of the four cells within the block into a vector with 36 components (4 histograms x 9 bins per histogram)
  - Divide this vector by its magnitude to normalize it



Credit: Chris McCormick

Histogram of Oriented Gradients

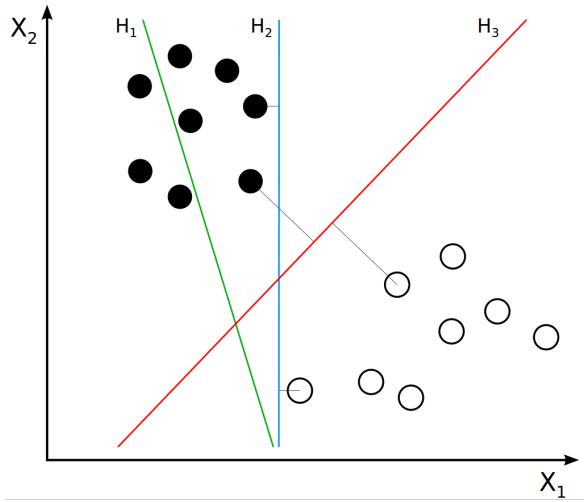
Final Descriptor

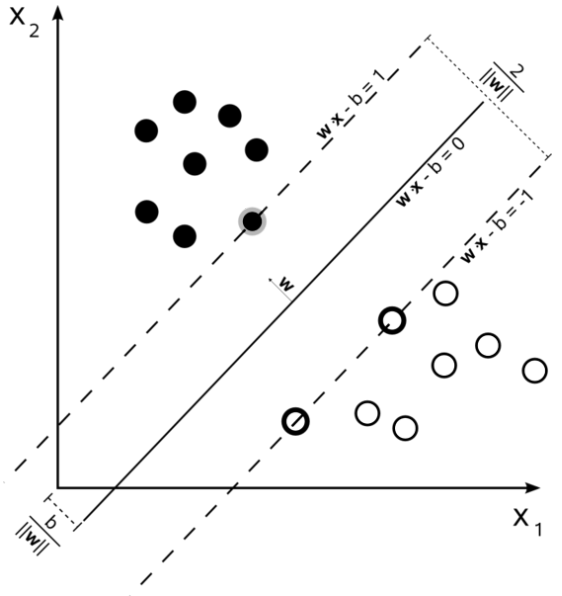
- The 64 x 128 pixel detection window is divided into 105 blocks:
  - 7 blocks horizontally
  - 15 blocks vertically
- 36 values per block:
  - 4 cells each with a 9-bin histogram for each cell
- HOG descriptor has 3,780 values
  - 7 blocks x 15 blocks x 4 cells per block x 9-bins per histogram

Credit: Chris McCormick

Histogram of Oriented Gradients

Support Vector Machine classification





- Maximum margin classifier: H3 separates classes “better” than H2
- Training examples defining separating hyperplane: support vectors
- If not linearly separable: use kernel trick to map to higher-dimensional space

Credit: Markus Vincze, Technische Universität Wien

Demos

The following code is taken from the personDetection project in the lectures directory of the ACV repository

See:

```
personDetection.h
personDetectionImplementation.cpp
personDetectionApplication.cpp
```

```
/*
  Example use of openCV to detect people (pedestrians) using HOG features
  -----------------------------------------------------------------------
  Implementation file

  David Vernon
  24 October 2017
*/

#include "personDetection.h"

void personDetection(char *filename) {

   char inputWindowName[MAX_STRING_LENGTH]         = "Input Image";
   char outputWindowName[MAX_STRING_LENGTH]        = "Person Image";
   Mat inputImage;
   Mat personImage;
   vector<Rect> people;
   Rect currentPerson;
   int scaleFactor;
   HOGDescriptor hog;    // Histogram of Oriented Gradients for people detection

   namedWindow(inputWindowName,      CV_WINDOW_AUTOSIZE);
   namedWindow(outputWindowName,     CV_WINDOW_AUTOSIZE);

   inputImage = imread(filename, CV_LOAD_IMAGE_COLOR); // Read the file
   if (!inputImage.data) {                             // Check for invalid input
      printf("Error: failed to read image %s\n",filename);
      prompt_and_exit(-1);
   }

   printf("Press any key to continue ...\n");
```

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

    /* the people in the original image are too small so we scale the input image */
    scaleFactor = 3;
    resize(inputImage,personImage,Size(inputImage.cols*scaleFactor, inputImage.rows*scaleFactor));

    hog.detectMultiScale(personImage, people);

    for (int count = 0; count < (int)people.size(); count++ ) {
        currentPerson = people[count];
        rectangle(personImage, currentPerson, cv::Scalar(0,0,255), 2);
    }

    imshow(inputWindowName,  inputImage );
    imshow(outputWindowName, personImage);

    do{
        waitKey(30);
    } while (!_kbhit());

    getchar(); // flush the buffer from the keyboard hit

    destroyWindow(inputWindowName);
    destroyWindow(outputWindowName);
}
```

Reading

R. Szeliski, *Computer Vision: Algorithms and Applications*, Springer, 2010.

Section 14.1.2 Pedestrian detection