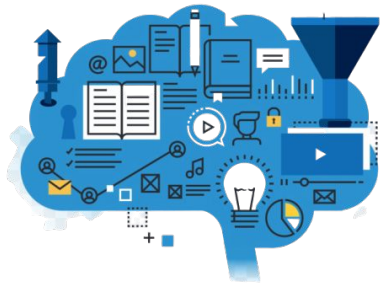
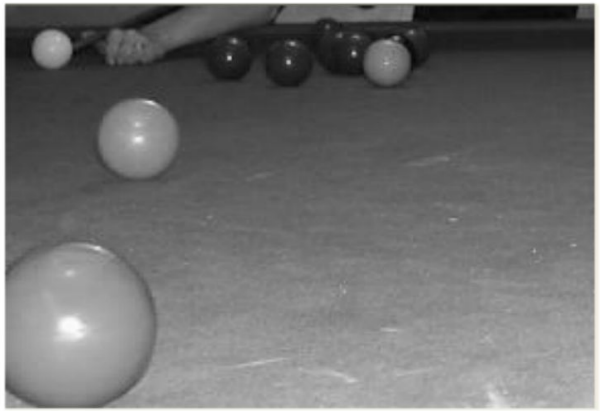


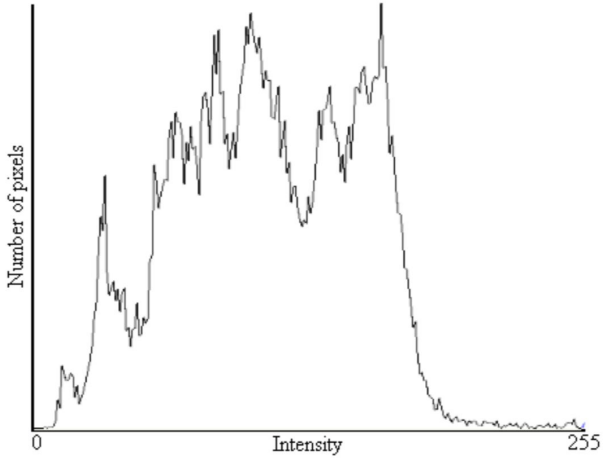
Computer Vision

Histograms

1D Histograms

- Global information
- Not unique
- Could be useful for classification





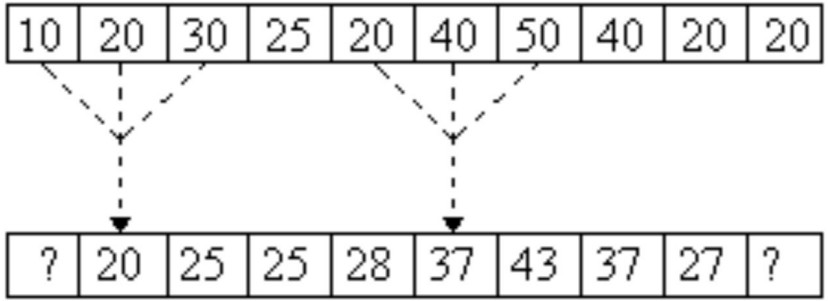
1D Histograms in OpenCV

```
Mat  display_image;
MatND  histogram;
const int* channel_numbers = { 0 };
float channel_range[] = { 0.0, 255.0 };
const float* channel_ranges = channel_range;
int number_bins = 64;
calcHist( &gray_image, 1, channel_numbers, Mat(), histogram, 1, &number_bins, &channel_ranges );

// drawing using cv::line()…
```

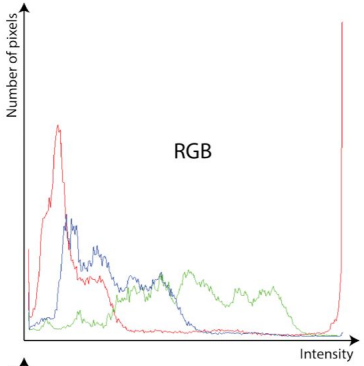
1D Histograms - Smoothing

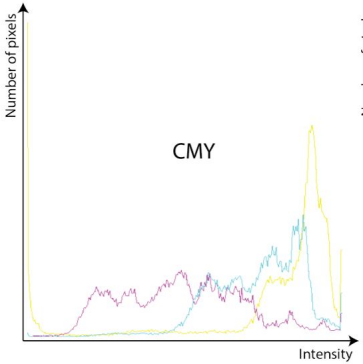
- Local min- and maximums
- How can we deal with noise?
  - Smooth the histogram

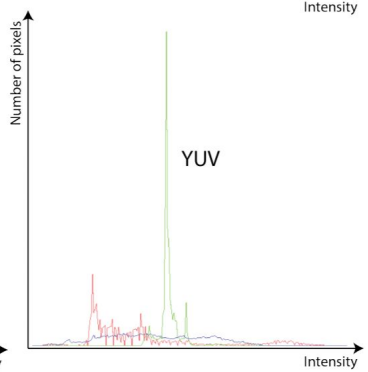


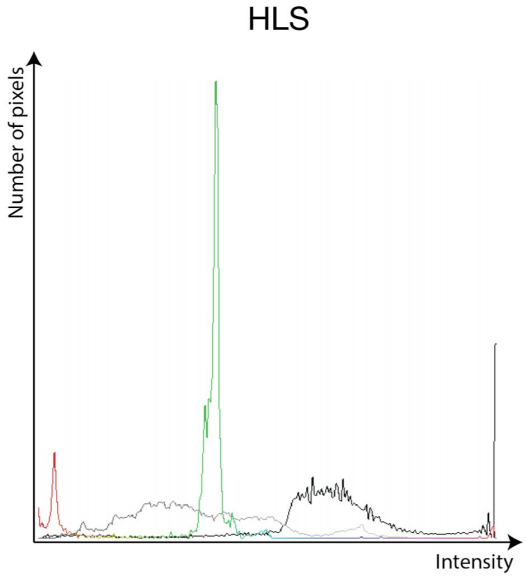

```
MatND smoothed_histogram = histogram[channel].clone();
for(int i = 1; i < histogram[channel].rows - 1; ++i)
  smoothed_histogram[channel].at<float>(i) =
     (histogram.at<float>(i-1) + histogram.at<float>(i) + histogram.at<float>(i+1)) / 3;
```

1D Histograms - Color Histograms









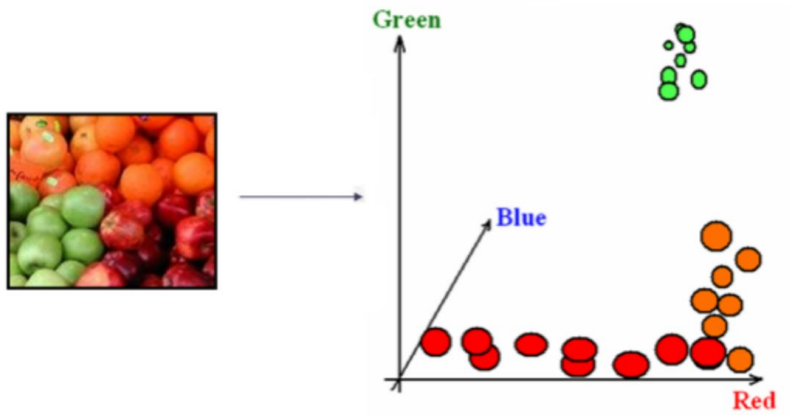
1D Color Histograms in OpenCV

```
MatND* histogram = new MatND[image.channels() ];
vector<Mat> channels(image.channels() );
split( image, channels );
const int* channel_numbers = { 0 };
float channel_range[] = { 0.0, 255.0 };
const float* channel_ranges = channel_range;
int number_bins = 64;
for (int chan=0; chan < image.channels(); chan++)
        calcHist( &(channels[chan]), 1, channel_numbers, Mat(),
            histogram[chan], 1, &number_bins, &channel_ranges );

// draw using cv::line()…
```

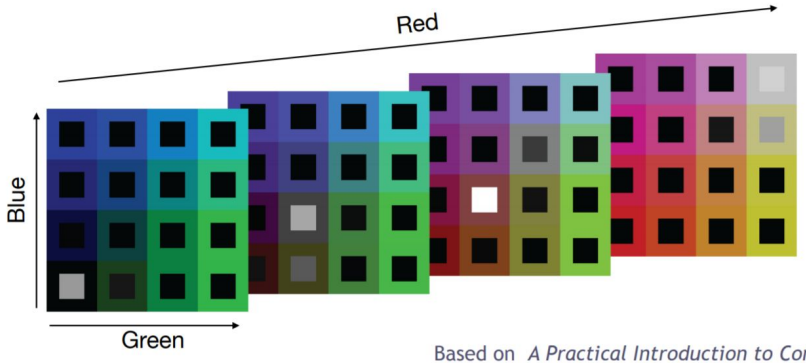
3D Histograms

- Channels are not independent
- Better discrimination comes from considiring all channels simultaneously



3D Histograms

- Reduce Quantisation
  - 6 bits
  - 4 bits
  - 2 bits



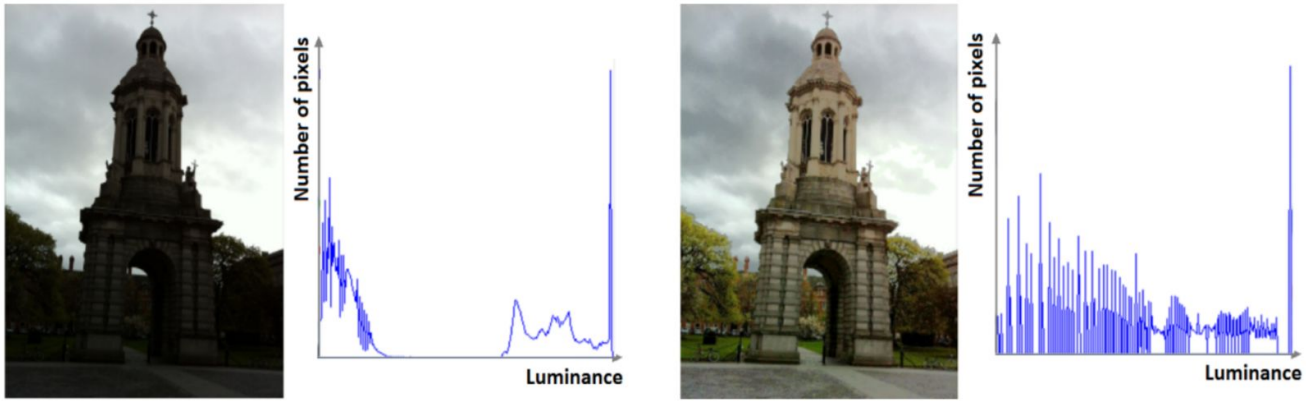
Based on *A Practical Introduction to Computer Vision with OpenCV* by Kenneth Dawson-Howe © Wiley & Sons Inc. 2014

3D Histograms in OpenCV

```
MatND histogram;
int channel_numbers[] = { 0, 1, 2 };
int* number_bins = new int[image.channels()];
for (ch=0; ch < image.channels(); ch++)
        number_bins[ch] = 64;
float ch_range[] = { 0.0, 255.0 };
const float* channel_ranges[] = {ch_range, ch_range, ch_range};
calcHist( &image, 1, channel_numbers, Mat(), histogram,
                     image.channels(), a_number_bins, channel_ranges );
```

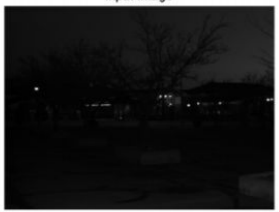
Equalisation

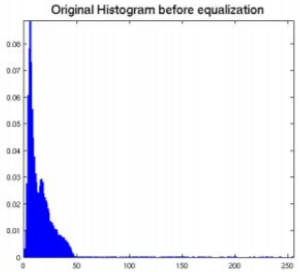
- If an image has insufficient contrast
- Human can distinguish 700-900 greyscales
- Evenly distribute the greyscales
- Normally equalise only the greyscales / luminance

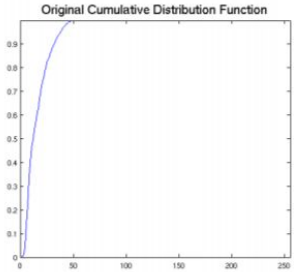


Equalisation in OpenCV

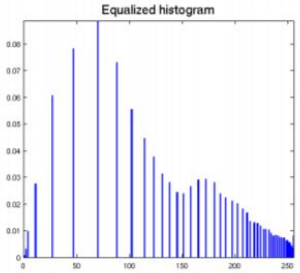
Input Image

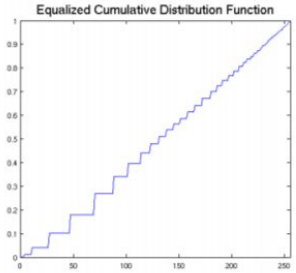






Output Image



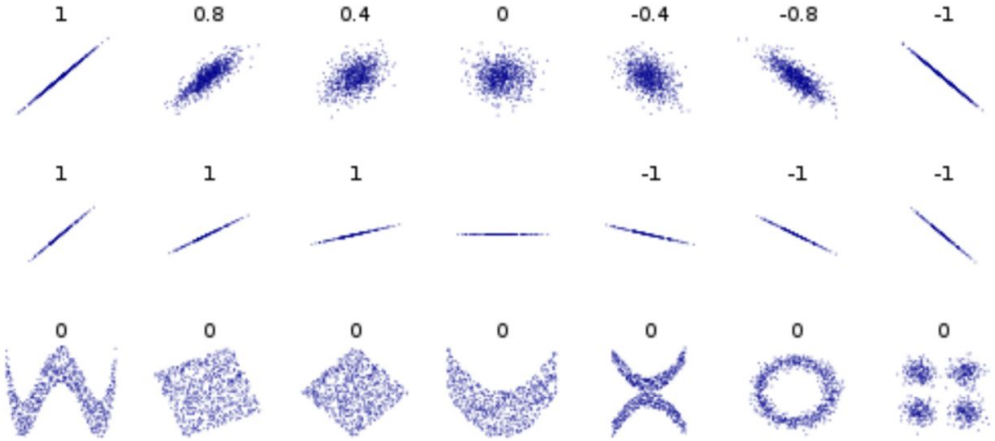



```
split(hls_image, channels);
vector<Mat> channels( hls_image.channels() );
equalizeHist( channels[1], channels[1] );
merge( channels, hls_image );
```

Histogram Comparison

Histogram Comparison - Correlation

- $$D_{Correlation}(h_1, h_2) = \frac{\sum_i (h_1(i) - \overline{h_1})(h_2(i) - \overline{h_2})}{\sqrt{\sum_i (h_1(i) - \overline{h_1})^2 \sum_i (h_2(i) - \overline{h_2})^2}}$$



Histogram Comparison in OpenCV

```
  normalize( histogram1, histogram1, 1.0);
  normalize( histogram2, histogram2, 1.0);
  double matching_score = compareHist( histogram1,
                                histogram2, CV_COMP_CORREL);

We can also use Chi-Square (CV_COMP_CHISQR),
Intersection (CV_COMP_INTERSECT) or
Bhattacharyya (CV_COMP_BHATTACHARYYA) metrics or
alternatively can use the Earth Mover's Distance function
(EMD())
```

Histogram Back Projection

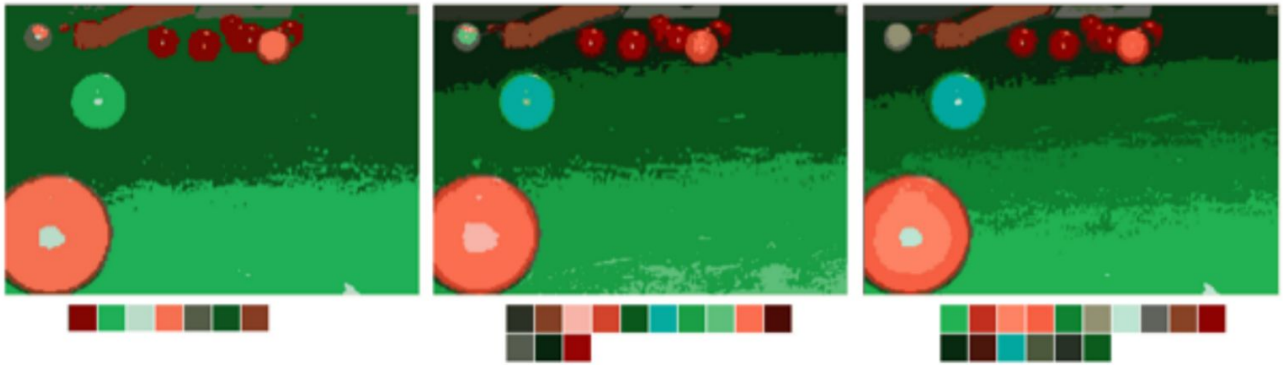
- Better approach to selecting colours (Based on samples)
- Histogram the samples
- Normalize the histogram
- Back project the normalized histogram onto an image
- Results in a probability image which indicated the similarity between the image and the sample set

Histogram Back Projection in OpenCV

```
calcHist( &hls_samples_image, 1, channel_numbers, Mat(),
   histogram,image.channels(),number_bins,channel_ranges);
normalize( histogram, histogram, 1.0);
Mat probabilities = histogram.BackProject( hls_image );
```

k means Clustering

- Different values of k
- Not all clusters end up with patterns
- More exemplars generally gives a better representation



k means Clustering in OpenCV

```
// Store the image pixels as an array of samples
Mat samples(image.rows*image.cols, 3, CV_32F);
float* sample = samples.ptr<float>(0);
for(int row=0; row<image.rows; row++)
  for(int col=0; col<image.cols; col++)
    for (int channel=0; channel < 3; channel++)
         samples.at<float>(row*image.cols+col,channel) =
                              (uchar) image.at<Vec3b>(row,col)[channel];
// Apply k-means clustering, determining the cluster
// centres and a label for each pixel.
   ….
```

k means Clustering in OpenCV

```
Mat labels, centres;
kmeans(samples, k, labels, TermCriteria( CV_TERMCRIT_ITER|
                  CV_TERMCRIT_EPS, 0.0001, 10000), iterations,
                   KMEANS_PP_CENTERS, centres );
// Use centres and label to populate result image
Mat& result_image = Mat( image.size(), image.type() );
for(int row=0; row<image.rows; row++)
  for(int col=0; col<image.cols; col++)
    for (int channel=0; channel < 3; channel++)
      result_image.at<Vec3b>(row,col)[channel] =
                           (uchar) centres.at<float>( *(labels.ptr<int>(

row*image.cols+col )), channel);
```