Computer Vision Systems Programming VO

Object Category Recognition

Christopher Pramerdorfer

Computer Vision Lab, Vienna University of Technology

Topics

Scene classification using the bag of words model

Fast face detection using boosted Haar features

Convolutional neural networks for large-scale problems

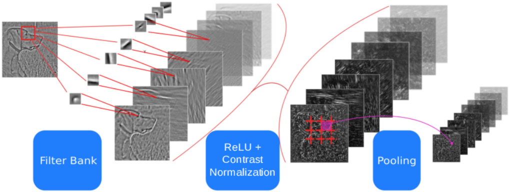


Image adaoted from Kavukcuoglu 2011

Scene Classification

We want to distinguish between c scene categories

- So $w \in \{0, \ldots, c-1\}$ (classification problem)

Street Scenes

Sea Scenes

Image adapted from Prince 2012

Scene Classification
Bag of Visual Words

We represent an image as a collection of visual words

- Images can be compared based on visual word distribution

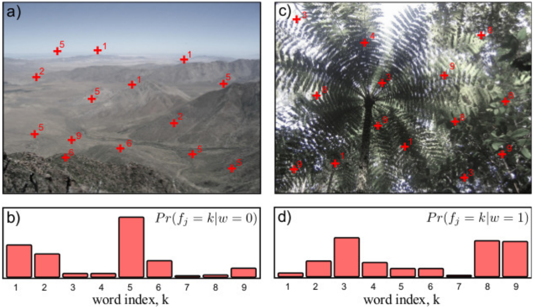


Image from Prince 2012

Scene Classification
Bag of Visual Words

Visual words are learned from an image collection

- Compute (SIFT) keypoints and descriptors for all images
- Cluster descriptors into k clusters using k-means
- k cluster means represent visual words

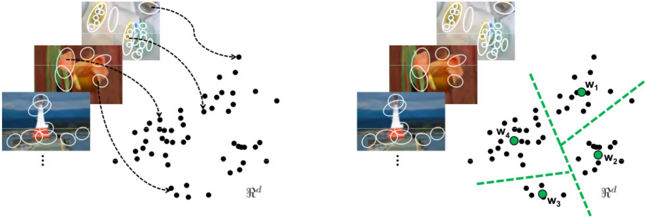
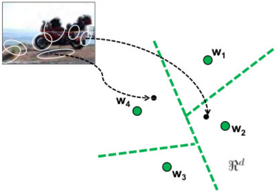


Image from Grauman and Leibe 2011

Visual word distribution $\mathbf{x} \in \mathbb{N}^k$ of image obtained by

- Computing keypoints and descriptors
- Assigning each feature to closest visual word
- Summing up the assignment counts for each visual word



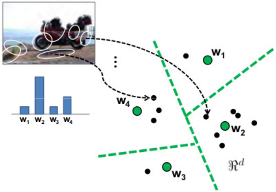


Image from Grauman and Leibe 2011

This image representation is called bag of (visual) words

Now that we have $\mathbf{x}$ we can select and learn a suitable model

- SVMs are often used in the literature
- For a probabilistic alternative see Prince 2012

Many improvements to this model exist

- Better clustering schemes
- Fuzzy assignment to visual words
- Spatial information (constellation model)

Popular and can work well, but no longer state of the art

Scene Classification
Bag of Visual Words Using OpenCV

```
// init SIFT
cv::Ptr<cv::FeatureDetector> kp = cv::FeatureDetector::create("SIFT");
cv::Ptr<cv::DescriptorExtractor> desc = cv::DescriptorExtractor::create("SIFT");

// compute visual words from training data
const int k = 50; // number of visual words
cv::BOWKMeansTrainer trainer(k);

for(const cv::Mat& im : images) { // std::vector of training images
    std::vector<cv::KeyPoint> keypoints; kp->detect(im, keypoints);
    cv::Mat descriptors; desc->compute(im, keypoints, descriptors);
    trainer.add(descriptors);
}

cv::Mat visualWords = trainer.cluster(); // k*128 (SIFT dimension)
```

Scene Classification
Bag of Visual Words Using OpenCV

```
// setup visual word frequency (our x) extractor
cv::Ptr<cv::DescriptorMatcher> fm = cv::makePtr<cv::BFMatcher>(cv::NORM_L2);
cv::BOWImgDescriptorExtractor extractor(desc, fm);
extractor.setVocabulary(visualWords);

// compute x for all training images
cv::Mat xTrain(images.size(), k, CV_32FC1);
for(std::size_t i = 0; i != images.size(); i++) {
    std::vector<cv::KeyPoint> keypoints; kp->detect(images[i], keypoints);
    cv::Mat x; extractor.compute(images[i], keypoints, x);
    xTrain(cv::Rect(0, i, k, 1)) = x;
}

// and corresponding w
cv::Mat wTrain(images.size(), 1, CV_32FC1); // fill me
```

```
// train our model (we use an SVM)
CvSVM svm;
svm.train(xTrain, wTrain);

// now we can predict the class of new images
std::vector<cv::KeyPoint> keypoints; kp->detect(newImage, keypoints);
cv::Mat x; extractor.compute(newImage, keypoints, x);
float w = svm.predict(x); // predicted class label
```

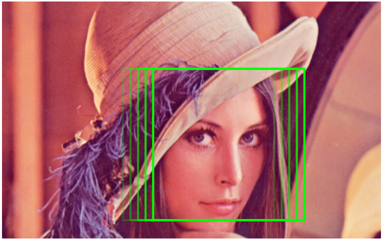
Face Detection

Image from `olympus-europa.com`

Face Detection

We don't know where the faces are so we

- Slide a fixed-size window over the image
- Compute $\Pr(w|\mathbf{x})$ for each window ($w = 1$ if face, 0 if not)



Face Detection
Selecting $\mathbf{x}$

Which are good features for this task?

Must be fast to compute (many windows)

Must be robust to illumination, so we use gradient information

Different approaches to encoding gradient information

- Compute gradients, pool orientations in blocks (e.g. SIFT)
- Use a collection of Gabor filters

Face Detection
Selecting $\mathbf{x}$

We want something faster

- So we use a “blocky” approximation of Gabor filters
- Difference between rectangular subwindows (Haar features)
- Can be computed in constant time using integral images

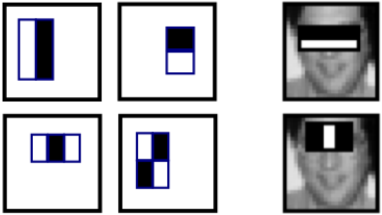


Image adapted from Prince 2012

Computing a Haar feature yields a scalar f_i

We define $\mathbf{x} = (x_1, \ldots, x_I)$ with $x_i = \text{heaviside}(f_i - t_i)$

I is very large (but finite)

- Different Haar features, subwindow locations, t_i
- We thus learn which features x_i work best

We model $\Pr(w|\mathbf{x})$ as a weighted sum of a feature subset

$$\Pr(w|\mathbf{x}) \propto a = \phi_0 + \sum_k \phi_k \, x_k \qquad (1 \leq k \leq I)$$

And learn the parameters $\boldsymbol{\theta} = (\phi_0, \phi_k, x_k)$ from training samples

- For each x_k in a large precomputed set
- Find optimal ϕ_0, ϕ_k
- Add best x_k to sum and repeat

This incremental approach is called boosting

We stop adding features at some point

- If the classification error no longer decreases
- After a specified maximum number of iterations

We end up with K good features, $K \ll I$

- For prediction we compute only these K features
- We stop if $P(w = 0) > t$ after processing $J \ll K$ features

If we don't care about probabilities we choose $w = \text{heaviside}(a)$

If we do we use logistic regression, $\Pr(w|\mathbf{x}) = \text{Bern}_w(\text{sig}(a))$

- We model w as a Bernoulli distribution
- Pass a through a logistic sigmoid to map it to $[0, 1]$
- Called logitboost in this context

This method was proposed in Viola and Jones 2001

Very efficient, ideal for digital cameras

Trades off efficiency for accuracy

- Features capture gradients coarsely, no color information
- More powerful but slower methods exist

Not invariant to scale changes (fixed-size window)

- Repeat detection at different image scales

Detect faces using a pretrained model

OpenCV also supports training

```
# detect faces using a pretrained cascade
image = cv2.imread('faces.jpg', cv2.IMREAD_GRAYSCALE)
cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
faces = cascade.detectMultiScale(image) # should tune parameters
```

Deep Learning

Selecting good features $\mathbf{x}$ for object recognition is challenging

- Why we previously learned $\mathbf{x}$

Learned features were low-level

- Based on SIFT descriptors or Haar wavelets

We want task-specific high-level features

- Virtually impossible to design manually
- So we learn them as well

Deep Learning

We learn these features hierarchically

- Model consists of layers
- The higher up the layer, the higher-level the feature
- Features in layer n are based on those in layer $n-1$

This results in a *deep* model, hence deep learning

At the same time we learn to predict $\mathbf{w}$

Deep Learning

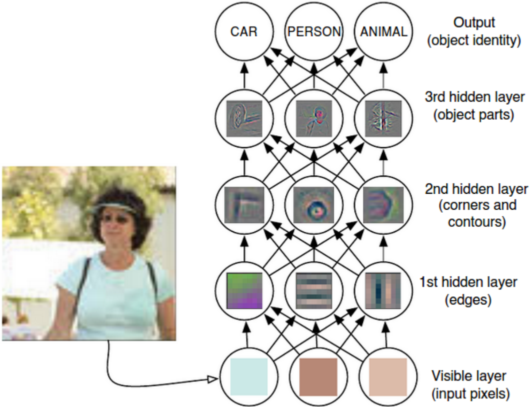


Image from Bengio, Goodfellow, and Courville 2015

Bibliography I

Bengio, Yoshua, Ian Goodfellow, and Aaron Courville (2015). *Deep Learning (Draft)*. MIT Press.

Grauman, Kristen and Bastian Leibe (2011). *Visual object recognition*. Morgan & Claypool.

Kavukcuoglu, Koray (2011). *Learning feature hierarchies for object recognition*. PhD thesis.

Prince, S.J.D. (2012). *Computer Vision: Models Learning and Inference*. Cambridge University Press.

Viola, Paul and Michael Jones (2001). *Rapid object detection using a boosted cascade of simple features*.