Template Matching

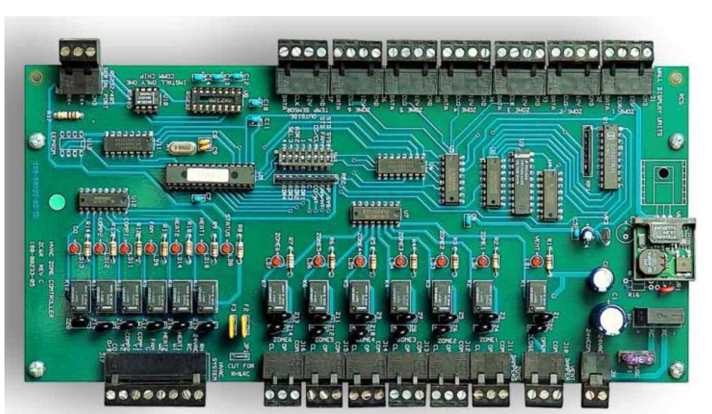
- So far, the classifiers were based on a large set of example patterns.
- All the variability of the patterns were learned from a training set using statistical methods.
- Sometimes, the designer of the classifier knows the variations that the patterns might undergo.
- Then, it is more efficient and more accurate to design a classifier using this knowledge.

Template Matching in Images

2

- Where are the resistors?
- How many are they?
- Are they correctly positioned?

defects detection in assembly line



Template Matching in Images

3

Problem specificities:

- Rigid object -> One example is enough.
- The circuit board is always photographed
  - from the same viewpoint -> No perspective
  - with the same illumination -> No lighting variation.

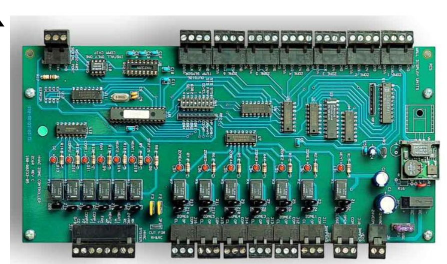
Hence, we may use a simple technique called Template Matching.

Template Matching in Images

4

Reference pattern  : $r(i,j)$ $i=0,\ldots,M\text{-}1$ $j=0,\ldots,N\text{-}1$

Test image: $t(i,j)$ $i=0,\ldots,I\text{-}1$ $j=0,\ldots,J\text{-}1$



Goal: detect the MxN sub-images within $t(i,j)$ that match $r(i,j)$.

Strategy: superimpose r on the test image and translate it at all possible location (x,y) and compute the mismatch:

$$D(x,y) = \sum_{i=x}^{x+M-1} \sum_{j=y}^{y+N-1} \left\| t(i,j) - r(i-x, j-y) \right\|^2 \qquad x=0,\ldots,I\text{-}1 \quad y=0,\ldots,J\text{-}1$$

Example

t is the threshold function: $$t(x,\theta)=\begin{cases}1 & \text{if } x\geq\theta \\ 0 & \text{if } x<\theta\end{cases}$$

$$t(\ \max(D)-D(x,y),\quad \theta\ )=$$

Cross Correlation

Problem: computing $D(x,y)$ is slow.

$$D(x,y)=\sum_{i=x}^{x+M-1}\sum_{j=y}^{y+N-1}\left\|t(i,j)-r(i-x,j-y)\right\|^2$$

$$=\sum_{i=x}^{x+M-1}\sum_{j=y}^{y+N-1}\left\|t(i,j)\right\|^2+\underbrace{\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\left\|r(i,j)\right\|^2}_{\text{does not depend on (x,y)}}-2\sum_{i=x}^{x+M-1}\sum_{j=y}^{y+N-1}t(i,j)r(i-x,j-y)$$

If $\sum_{i=x}^{x+M-1}\sum_{j=y}^{y+N-1}\left\|t(i,j)\right\|^2$ does not vary much on the image

then minimizing $D(x,y)$ is the same

as maximizing $c(x,y)$: $$c(x,y)=\sum_{i=x}^{x+M-1}\sum_{j=y}^{y+N-1}t(i,j)r(i-x,j-y)$$

Fast Cross Correlation

$$c(x,y) = \sum_{i=x}^{x+M-1} \sum_{j=y}^{y+N-1} t(i,j) r(i-x, j-y)$$

is the cross-correlation between $t(i,j)$ and $r(i,j)$.

Do you recognize this formula?

This is actually the formula of a convolution: $c(x,y) = t(x,y) \otimes r(x,y)$

An efficient way to compute a convolution is via the Convolution Theorem:

$$c(x,y) = \underbrace{\mathrm{IDFT}}_{\text{the 2 sums are gone.}}\left[\mathrm{DFT}\left[t(x,y)\right] \underbrace{\times}_{\text{normal product}} \mathrm{DFT}\left[r(x,y)\right]\right]$$

Normalized Cross Correlation

Now what if $t(i,j)$ cannot be assumed to be constant over the image?

Then we cannot neglect the term $\sum_{i=x}^{x+M-1} \sum_{j=y}^{y+N-1} \left\| t(i,j) \right\|^2$

In this case, instead of using the cross-correlation, the normalized cross-correlation is used:

$$c_N(x,y) = \frac{\sum_{i=x}^{x+M-1} \sum_{j=y}^{y+N-1} t(i,j) r(i-x, j-y)}{\sqrt{\sum_{i=x}^{x+M-1} \sum_{j=y}^{y+N-1} \left\| t(i,j) \right\|^2 \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \left\| r(i,j) \right\|^2}}$$

Normalized Cross Correlation

$$c_N(x,y) = \frac{\sum_{i=x}^{x+M-1} \sum_{j=y}^{y+N-1} t(i,j)\,r(i-x,\,j-y)}{\sqrt{\sum_{i=x}^{x+M-1} \sum_{j=y}^{y+N-1} \|t(i,j)\|^2 \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \|r(i,j)\|^2}}$$

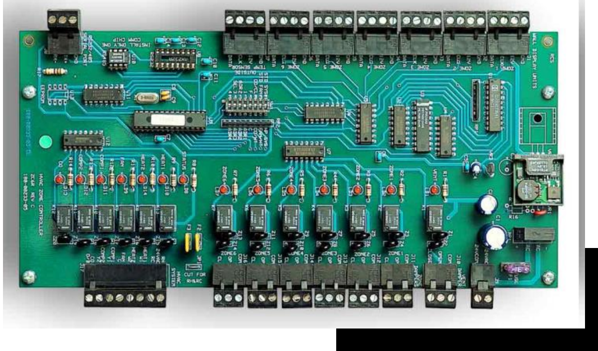
This formula may be cumbersome, to simplify it, the normalized cross correlation of vectors a and b is:

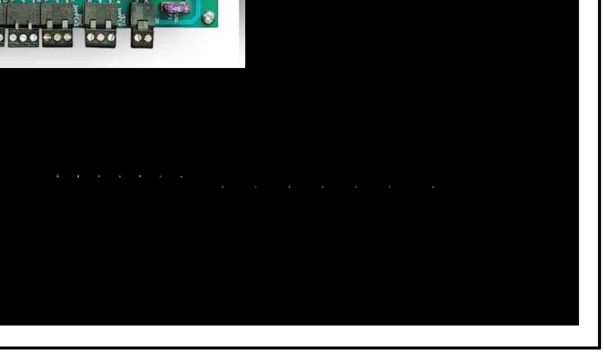
$$c_N = \frac{a^T b}{\|a\|\|b\|}$$

Cauchy-Schwarz inequality: $\left|a^T b\right| \leq \|a\|\|b\|$

Hence: $-1 \leq c_N \leq 1$ and c_N=1 only if a=αb with α positive scalar.

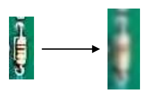
Normalized Cross Correlation Result





Blurring the Reference Pattern

To allow for small displacements (rotation or perspective variation) of the object in the input image, it helps to blur the reference pattern.



Deformable Templates

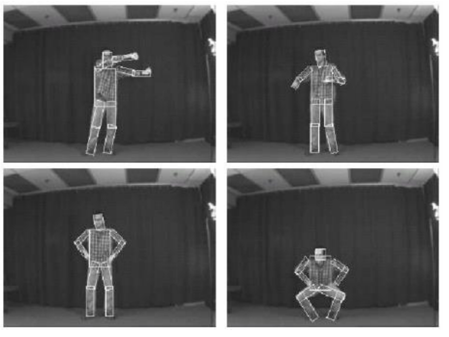
Template Matching was concerned with:

- rigid objects,
- viewed from the same angle,
- cannot handle occlusion
- with the same illumination

Deformable Template is a method that allow the object to deform:

- **flexible** objects,
- some **viewpoint variations** are allowed,
- some **occlusion** is allowed
- same illumination

Examples of Objects that can Deform



The relative location of the limbs depends on the **gesture** of the person.

The relative location of eyes, nose and mouth depends on the **person** and on the **viewpoint**.

Parts based Object Representation

Template Matching with a single template would not work on these examples.

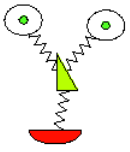
These examples are characterized by:

- The object is constituted by different **parts**.
- The appearance of each part is somewhat constant.
- The relative position of each part varies.

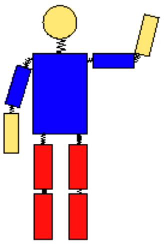
We want to localize the object by localizing each of its parts.

Part based Object Representation

15



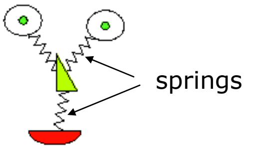
A face object is represented by the appearance of the eyes, nose and mouth, and a shape model that code how these parts can deform.

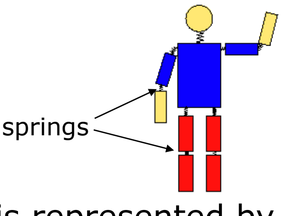


A body object is represented by the appearance of the head, the torso and each limbs, and a shape model that code how these parts can deform.

The Problem as Flexible Model

16





Here, the shape of an object is represented by "springs" connecting certain pair of parts.

This can be modeled as a Probabilistic Graphical Model where a part is a node and a spring is an edge:

Graph:

$G=(V,E)$

$V = \{v_1,\dots,v_n\}$ are the parts

$(v_i, v_j) \in E$ are the edges connecting the parts.

Part based Cost Function

We want to localize an object by finding the parts that simultaneously:

- minimize the appearance mismatch of each part,

and

- minimize the deformation of the spring model.

$m_i(l_i)$: cost of placing part i at location $l_i = (x_i, y_i)^T$

$d_{ij}(l_i,l_j)$: deformation cost.

Optimal location for the object is $L^* = \left(l_1^*, \ldots, l_n^*\right)$ where

$$L^* = \arg\min_L \left(\underbrace{\sum_{i=1}^{n} m_i(l_i)}_{\text{appearance cost}} + \underbrace{\sum_{(v_i,v_j)\in E} d_{ij}(l_i,l_j)}_{\text{deformation cost}} \right)$$

Part based Cost Function

$$L^* = \arg\min_L \left(\sum_{i=1}^{n} m_i(l_i) + \sum_{(v_i,v_j)\in E} d_{ij}(l_i,l_j) \right)$$

It would not be optimal to first detect each part then to combine them. Why?

Because detecting a single part separately, is a more difficult problem, as it involves less information.

This is why the cost function is minimized over all possible locations for all parts taking into account both appearance and deformation.

Part based Cost Function

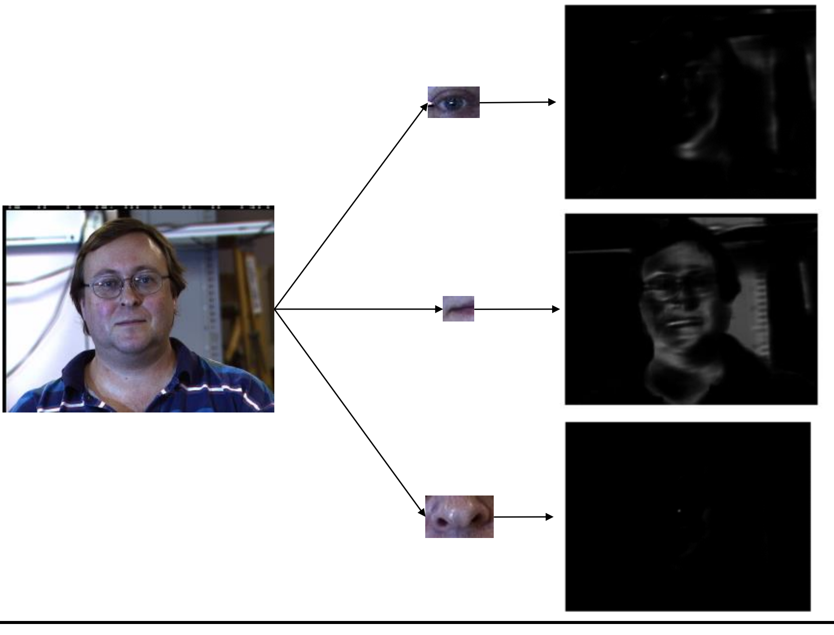
$$L^* = \arg\min_L \left(\sum_{i=1}^{n} m_i(l_i) + \sum_{(v_i,v_j)\in E} d_{ij}(l_i, l_j) \right)$$

$m_i(l_i)$: cost of placing part i at location l_i.

This can be done by template matching for example.

Template Matching is not the best choice as it is computationally expensive.

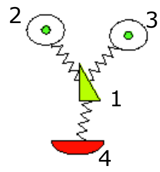
Template Matching for each Part



Deformation Cost

Now, the question is: how to combine these appearance results, using the shape information, in order to find the global minimum of the cost function?

$$\sum_{(v_i,v_j)\in E} d_{ij}(l_i,l_j) = ?$$



$$\sum_{(v_i,v_j)\in E} d_{ij}(l_i,l_j) = d_{12}(l_1,l_2) + d_{13}(l_1,l_3) + d_{14}(l_1,l_4)$$

e.g. using the Mahalanobis Distance

mean displacement of part 2 from part 1

$$d_{12}(l_1,l_2) = (l_2 - \overline{l_2} - l_1)^T \Sigma_{12}^{-1} (l_2 - \overline{l_2} - l_1)$$

says where part 2 is likely to be located given the location of part 1.

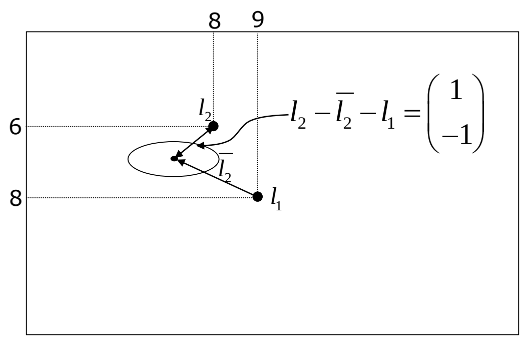
covariance matrix computed on a training set.

Deformation Cost Computation

Example of computation of the deformation:

Given $l_1 = \begin{pmatrix} 9 \\ 8 \end{pmatrix}$ what is the cost of having $l_2 = \begin{pmatrix} 8 \\ 7 \end{pmatrix}$

$$d_{12}(l_1,l_2) = (l_2 - \overline{l_2} - l_1)^T \Sigma_{12}^{-1} (l_2 - \overline{l_2} - l_1) \; = 1.5$$



with the mean and the covariance fixed:

$$\overline{l_2} = \begin{pmatrix} -2 \\ -1 \end{pmatrix}$$

$$\Sigma_{12} = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$$

Efficient Implementation

$$L^* = \arg\min_L \left(\sum_{i=1}^{n} m_i(l_i) + \sum_{(v_i,v_j)\in E} d_{ij}(l_i,l_j) \right)$$

Finding the **global minimum** of this cost function requires computing it for all possible positions of l_i and l_j. If h is the number of pixel, this algorithm needs $O(h^2)$ evaluations. This is far too inefficient.

"Pictorial Structures for Object Recognition"
Felsenszwalb et al. in *Intl. Journal of Computer Vision, Jan. 2005.*

It is shown that it can be computed in $O(nh)$ which is much much better.

A Bayes Framework for Deformable Templates matching.

"Pictorial Structures for Object Recognition"
Felsenszwalb et al. in *Intl. Journal of Computer Vision, Jan. 2005.*

Statistical Framework

We want to maximize the posterior: $p(L|I,\theta)$

$L = (l_1, l_1, \ldots, l_n)^T$: 2D position of n parts in the image.

I : input image

θ : model parameters (modeling appearance and shape)

Bayes Theorem: $\boxed{p(L|I,\theta) \propto p(I|L,\theta)\, p(L|\theta)}$

$p(I|L,\theta)$: likelihood of seeing a particular image given that an object is at some position. This is the appearance model.

$p(L|\theta)$: prior probability that an object is at a particular position. This is the shape model.

Image Likelihood

$$p(L|I,\theta) \propto p(I|L,\theta)\, p(L|\theta)$$

If the n parts are image patches that do *not overlap*, then we may assume that they are *statically independent*:

$$p(I|L,\theta) \propto \prod_i^n p(I|l_i,\theta) \quad \text{where} \quad l_i = (x_i, y_i)^T \text{ and } L = (l_1, \ldots, l_n)$$

Hence, the full posterior is:

$$\boxed{p(L|I,\theta) \propto \left(\prod_i^N p(I|l_i,\theta) \right) p(L|\theta)}$$

probability that part i is at location l_i, depends on the image and on each part individually (independent).

probability of a shape configuration.

Cost Function

Maximizing the posterior $p(L|I,\theta)$ is equivalent to minimizing its negative logarithm:

$$L^* = \arg\max_L \left(\prod_i^N p(I|l_i,\theta) \right) p(l_1,\ldots,l_n|\theta)$$

$$\boxed{L^* = \arg\min_L \left(-\left(\sum_{i=1}^n \ln p(I|l_i,\theta) \right) - \ln p(l_1,\ldots,l_n|\theta) \right)}$$

Learning Model Parameters

θ are the model parameters. It regroups two kinds of parameters:

- Appearance parameters, denoted by u,
- shape parameters, denoted by c

$$\theta = (u, c)$$

We need to learn them from a training set of m labeled examples:

$I^1,\ldots,I^m$ and $L^1,\ldots,L^m$

Learning Model Parameters

We want to find the Maximum Likelihood estimate of θ, i.e. the value θ^* that maximizes:

$$p(I^1,\ldots,I^m,L^1,\ldots,L^m|\theta)=\prod_{k=1}^{m}p(I^k,L^k|\theta) \longleftarrow \text{assuming .... ?}$$

Recall that $p(I,L|\theta)=p(I|L,\theta)\,p(L|\theta)$ hence:

$$\theta^* = \arg\max_{\theta}\prod_{k=1}^{m}p(I^k|L^k,\theta)\prod_{k=1}^{m}p(L^k|\theta) \qquad \theta=(u,c)$$

$$\theta^* = \arg\max_{u,c}\prod_{k=1}^{m}p(I^k|L^k,u)\prod_{k=1}^{m}p(L^k|c)$$

Hence,

$$u^* = \arg\max_{u}\prod_{k=1}^{m}p(I^k|L^k,u)$$

$$c^* = \arg\max_{c}\prod_{k=1}^{m}p(L^k|c)$$

Estimating Appearance Parameters

$$u^* = \arg\max_{u}\prod_{k=1}^{m}p(I^k|L^k,u)$$

Recall that we assumed the image likelihood of the n parts to be *independent*: $p(I|L,\theta)\propto\prod_{i}^{n}p(I|l_i,\theta)$

$$u^* = \arg\max_{u}\prod_{k=1}^{m}\prod_{i=1}^{n}p(I^k|l_i^k,u_i)$$

$$= \arg\max_{u}\prod_{i=1}^{n}\prod_{k=1}^{m}p(I^k|l_i^k,u_i)$$

Hence, we can independently solve for each part:

$$u_i^* = \arg\max_{u_i}\prod_{k=1}^{m}p(I^k|l_i^k,u_i)$$

Estimating Appearance Parameters

31

Now, we need to choose a model for $p(I|l_i, u_i)$

Any model learnt on the lecture about *Density Estimation* can be used: Gaussian, Mixture of Gaussians, non-parametric model, etc.

Here, for simplicity we model a patch of the image centered at the position l_i with a Gaussian model with a unit covariance matrix:

$$p(I|l_i, u_i) = N(\mu_i, Id)$$

We have learnt that the ML estimate is: $\mu_i = \frac{1}{m}\sum_{k=1}^{m} I_{l_i}$

where I_{l_i} is the patch of the image I centered at l_i

Gaussian Appearance Model

32

$$p(I|l_i, u_i) = N(\mu_i, Id)$$

Recall that $L^* = \arg\min_L \left(-\left(\sum_{i=1}^{n} \underline{\ln p(I|l_i, \theta)} \right) - \ln p(l_1, \ldots, l_n|\theta) \right)$

number of pixel in patch i.

$$-\ln p(I|l_i, u_i) = \frac{1}{2}\left\|I_{l_i} - \mu_i\right\|^2 + \frac{d_i}{2}\ln 2\pi$$

Hence, using a Gaussian appearance model with an identity covariance matrix is the same as doing template matching on each part separately.

Shape Model

Likewise we need to choose a model for the shape configuration prior $p(L|c)$

Again, any model learnt on the lecture about *Density Estimation* can be used: Gaussian, Mixture of Gaussians, non-parametric model, etc.

We have seen that the shape model can be learnt independently from the appearance model:

$$c^* = \arg\max_c \prod_{k=1}^{m} p(L^k|c)$$

Gaussian Shape Model

For instance, we can choose a Gaussian model, for which

$$c = (\mu_L, \Sigma_L)$$

$$\Rightarrow p(L|c) = N(\mu_L, \Sigma_L)$$

We have learnt that the ML estimate are:

$$\mu_L = \frac{1}{m}\sum_{k=1}^{m} L^k \quad \text{and} \quad \Sigma_L = \frac{1}{m}\sum_{k=1}^{m}(L^k - \mu_L)(L^k - \mu_L)^T$$

and its negative logarithm is:

$$-\ln p(L|\mu_L, \Sigma_L) = \frac{1}{2}(L - \mu_L)^T \Sigma_L^{-1}(L - \mu_L) + n\ln 2\pi + \frac{1}{2}\ln|\Sigma_L|$$

Algorithm for 3 parts and *h* pixels

35

$$L^* = \arg\min_L \left(-\left(\sum_{i=1}^{n} \ln p(I \mid l_i, \theta) \right) - \ln p(l_1, \ldots, l_n \mid \theta) \right)$$

```
best_cost = Infinity;
for l1 = 1 to h,  pI_l1[l1] = log of image likelihood of part 1 in l1 ; endfor
for l2 = 1 to h,  pI_l2[l2] = log of image likelihood of part 2 in l2 ; endfor
for l3 = 1 to h,  pI_l3[l3] = log of image likelihood of part 3 in l3 ; endfor

for l1 = 1 to h
    for l2 = 1 to h
        for l3 = 1 to h
            pL = log of probability of configuration (l1, l2, l3)
            cost = -pI_l1[l1] - pI_l2[l2] - pI_l3[l3] - pL
            best_cost = min(cost, best_cost)
        endfor
    endfor
endfor
```

n nested loops !!!

Very slow !

Prior Shape Model

36

$$p(L \mid \theta) = p(l_1, l_2, l_3 \mid \theta)$$
$$= p(l_3 \mid l_2, l_1, \theta)\, p(l_2, l_1 \mid \theta)$$
$$= p(l_3 \mid l_2, l_1, \theta)\, p(l_2 \mid l_1, \theta)\, p(l_1 \mid \theta)$$

Problem: It is very time consuming to evaluate $p(L \mid \theta)$

This is due to $p(l_3 \mid l_2, l_1, \theta)$.   Why?

Let's assume that there are h pixel positions in the input image. To maximize $p(L \mid \theta)$ over the whole image we must evaluate $p(l_3 \mid l_2, l_1, \theta)$ for all combinations of the 3 parts.

$$\left.\begin{array}{l} \text{For } 3 \text{ parts: } h^3 \text{ evaluations.} \\ \text{For } n \text{ parts: } h^n \text{ evaluations.} \end{array}\right\} \textit{exponential} \text{ time algorithm}$$

Conditional Independence

$$p(L|\theta) = p(l_3|l_2, l_1, \theta)\, p(l_2|l_1, \theta)\, p(l_1|\theta)$$

How can we speed that up?

Answer: assume conditional independence between parts.

Now, let's assume that l_3 and l_2 are *conditionally independent* given l_1. This means that if l_1 is known, then knowing l_2 gives us *no additional information* to estimate l_3. Hence:

$$p(l_3|l_2, l_1, \theta) = p(l_3|l_1, \theta)$$

$$\Rightarrow \quad p(L|\theta) = p(l_3|l_2, l_1, \theta)\, p(l_2|l_1, \theta)\, p(l_1|\theta)$$

$$= p(l_3|l_1, \theta)\, p(l_2|l_1, \theta)\, p(l_1|\theta)$$

Graphical Model

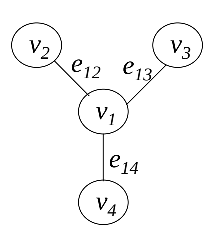
The conditional independence relations can be nicely represented by a Graphical Model where a part is a node and an edge connects two *dependent* parts:

Undirected Graph: $G=(V, E)$

$V = \{v_1,\ldots,v_n\}$ are the parts

$e_{ij} \in E$ are the edges connecting the parts (v_i, v_j) .

$\equiv$



$$p(L|\theta) = p(l_2|l_1, \theta)\, p(l_3|l_1, \theta)\, p(l_4|l_1, \theta)\, p(l_1|\theta)$$

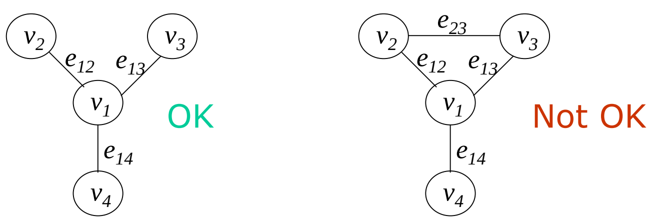
Graphical Model

39

The condition to have a **polynomial** time detection algorithm is that the graph is **acyclic**.

This means that there can be no cycles in the graph,
i.e. no loops,
i.e. there can be no path starting and ending on one node.

Example:



Graphical Model

40

$$p(L|\theta) = \underbrace{p(l_2|l_1,\theta)\,p(l_3|l_1,\theta)\,p(l_4|l_1,\theta)}\,p(l_1|\theta)$$

This encodes **relative** information:
With this, if I tell you where is the nose, you can tell me roughly where should be the eyes (without looking at the image).

This encodes **absolute** information. This tells you where is the tip of the nose on *any* image.

However, we assume the nose could be *anywhere*. Hence, we must model this as a *uniform* PDF.

$$p(L|\theta) \propto p(l_2|l_1,\theta)\,p(l_3|l_1,\theta)\,p(l_4|l_1,\theta)\,\cancel{p(l_1|\theta)}$$

$$p(L|\theta) \propto \prod_{(v_i,v_j)\in E} p(l_j|l_i,\theta)$$

constant

Part based Cost Function

41

We want to find the object configuration L^* that maximizes the posterior:

$$L^* = \arg\max_L \prod_i^n p(I \mid l_i, \theta) \prod_{(v_i, v_j) \in E} p(l_j \mid l_i, \theta)$$

This is the same as minimizing its negative logarithm:

$$L^* = \arg\min_L \left(-\sum_{i=1}^n \ln p(I \mid l_i, \theta) - \sum_{(v_i, v_j) \in E} \ln p(l_j \mid l_i, \theta) \right)$$

probability that part i is at location l_i, depends on the image and on each part independently.

probability of a relative position between two parts.

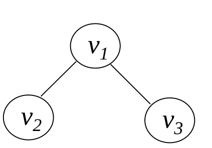
Algorithm based on Cond. Indep.

42

$$L^* = \arg\min_L \left(-\sum_{i=1}^n \ln p(I \mid l_i, \theta) - \sum_{(v_i, v_j) \in E} \ln p(l_j \mid l_i, \theta) \right)$$

How to implement this *efficiently* ?

Let's take an example with 3 nodes:



$$C^* = \min_{l_1, l_2, l_3} \left(-\ln p(I \mid l_1) - \ln p(I \mid l_2) - \ln p(I \mid l_3) - \ln p(l_2 \mid l_1) - \ln p(l_3 \mid l_1) \right)$$

computing here the value of the minimum, not the location of the minimum, however computing the location is identical, just replace min by argmin

dependence on the model parameters θ is omited

43

Alg. based on Cond. Indep.

$$C^* = \min_{l_1,l_2,l_3}\left(-\ln p(I|l_1) - \ln p(I|l_2) - \ln p(I|l_3) - \ln p(l_2|l_1) - \ln p(l_3|l_1)\right)$$

$$C^* = \min_{l_1}\left(-\ln p(I|l_1) + \min_{l_2}\left(-\ln p(I|l_2) - \ln p(l_2|l_1)\right) + \min_{l_3}\left(-\ln p(I|l_3) - \ln p(l_3|l_1)\right)\right)$$

44

Alg. based on Cond. Indep.

$$C^* = \min_{l_1}\left(-\ln p(I|l_1) + \min_{l_2}\left(-\ln p(I|l_2) - \ln p(l_2|l_1)\right) + \min_{l_3}\left(-\ln p(I|l_3) - \ln p(l_3|l_1)\right)\right)$$

```
best_C = Infinity
for l1 = 1 to h
    best_C_12[l1] = Infinity
    for l2 = 1 to h
        best_C_12[l1] = min( -log of image likelihood of part 2 in l2
                             -log of probability of l2 given l1,
                             best_C_12[l1] )
    endfor
    best_C_13[l1] = Infinity
    for l3 = 1 to h
        best_C_13[l1] = min( -log of image likelihood of part 3 in l3
                             -log of probability of l3 given l1,
                             best_C_13[l1] )
    endfor
    best_C = min( -log of image likelihood of part 1 in l1 + best_C_12[l1] + best_C_13[l1],
                  best_C )
endfor
```

only **2** nested loops

Alg. based on Cond. Indep.

Now, only $2h^2$ evaluations are needed.
With conditional independence, we go from an *exponential* time $O(h^n)$ algorithm to a *polynomial* time $O(nh^2)$ algorithm.

Using some other tricks from Dynamic Programming and Distance transforms, it can even be computed in linear time $O(nh)$.

see:
"Pictorial Structures for Object Recognition"
Felsenszwalb et al. in *Intl. Journal of Computer Vision, Jan. 2005.*

Learning Model Parameters

Θ are the model parameters. It regroups three kinds of parameters:

- Appearance parameters, denoted by u,
- Graph structure (edges), denoted by E, and
- shape parameters, denoted by $c = \{c_{ij} \,|\, (v_i, v_j) \in E\}$

Comment:
For star models i = 1

We already saw how the appearance model is learnt.

Let's now see how the graph model is learnt.

Earlier, we saw that the shape parameters can be learnt independently from the appearance parameters:

$$E^*, c^* = \arg\max_{E,c} \prod_{k=1}^{m} p(L^k \,|\, E, c)$$

Estimating the shape parameters

$$E^*, c^* = \arg\max_{E,c} \prod_{k=1}^{m} p(L^k \mid E, c)$$

We have seen that using conditional independence assumptions:

$$p(L \mid E, c) \propto \prod_{(v_i, v_j) \in E} p(l_j \mid l_i, E, c_{i|j})$$

$$= \prod_{(v_i, v_j) \in E} \frac{p(l_j, l_i \mid E, c_{i,j})}{p(l_i \mid c_i)}$$

$$\propto \prod_{(v_i, v_j) \in E} p(l_j, l_i \mid E, c_{i,j})$$

$p(l_i \mid c_i)$ encodes absolute position information, that we assume to be constant.

$$E^*, c^* = \arg\max_{E,c} \prod_{(v_i, v_j) \in E} \prod_{k=1}^{m} p(l_i^k, l_j^k \mid E, c_{i,j})$$

Estimating the shape parameters

$$E^*, c^* = \arg\max_{E,c} \prod_{(v_i, v_j) \in E} \prod_{k=1}^{m} p(l_i^k, l_j^k \mid E, c_{i,j})$$

For now, let's assume that we have a set of graph connections E, hence the parameters for each connection can be estimated separately:

$$c_{i,j}^* = \arg\max_{c_{ij}} \prod_{k=1}^{m} p(l_i^k, l_j^k \mid c_{i,j})$$

Again, the PDF chosen to model this joint probability can be any model we have learnt previously, however, using a Gaussian model offers some advantage:

$$p(l_i^k, l_j^k \mid c_{i,j}^*) = N(\mu_{i,j}, \Sigma_{i,j}) \qquad \text{with} \qquad \mu_{i,j} = \begin{bmatrix} \mu_i \\ \mu_j \end{bmatrix} \quad \Sigma_{i,j} = \begin{bmatrix} \Sigma_i & \Sigma_{ij} \\ \Sigma_{ji} & \Sigma_j \end{bmatrix}$$

Gaussian Conditional Probability

$$p(l_i, l_j \mid c^*_{i,j}) = N(\mu_{i,j}, \Sigma_{i,j}) \quad \text{with} \quad \mu_{i,j} = \begin{bmatrix} \mu_i \\ \mu_j \end{bmatrix} \quad \Sigma_{i,j} = \begin{bmatrix} \Sigma_i & \Sigma_{ij} \\ \Sigma_{ji} & \Sigma_j \end{bmatrix}$$

However, later in the cost we need a function of the conditional instead of the joint probability:

$$L^* = \arg\min_L \left(-\sum_{i=1}^{n} \ln p(I \mid l_i, \theta) - \sum_{(v_i, v_j) \in E} \ln \boxed{p(l_j \mid l_i, \theta)} \right)$$

Recall from the first exercise that for a Gaussian distribution, conditioning on a set of variable preserves the Gaussian property:

$$p(l_j \mid l_i, c^*_{j|i}) = N(\mu_{j|i}, \Sigma_{j|i}) \quad \text{with} \quad \begin{aligned} \mu_{j|i}(l_i) &= \mu_j + \Sigma_{ji}\Sigma_i^{-1}(l_i - \mu_i) \\ \Sigma_{j|i} &= \Sigma_j - \Sigma_{ji}\Sigma_i^{-1}\Sigma_{ij} \end{aligned}$$

Learning the Graph Structure

The last thing to be learnt is the graph connections, E.
Recall that the ML estimate of the shape model parameters is:

$$E^*, c^* = \arg\max_{E,c} \prod_{(v_i, v_j) \in E} \prod_{k=1}^{m} p(l_i^k, l_j^k \mid E, c_{i,j})$$

$$c^*_{i,j} = \arg\max_{c_{ij}} \prod_{k=1}^{m} p(l_i^k, l_j^k \mid c_{i,j})$$

Hence, the quality of a connection between two parts is given by the probability of the examples under the ML estimate of their joint distribution:

$$q(v_i, v_j) = \prod_{k=1}^{m} p(l_i^k, l_j^k \mid c^*_{i,j})$$

And the optimal graph is given by: $E^* = \arg\max_E \prod_{(v_i, v_j) \in E} q(v_i, v_j)$

Learning the Graph Structure

The optimal graph is given by: $E^* = \arg\max_E \prod_{(v_i, v_j) \in E} q(v_i, v_j)$

$$E^* = \arg\min_E \sum_{(v_i, v_j) \in E} -\ln q(v_i, v_j)$$

The Algorithm for finding this acyclic graph maximizing E^*:

1. Compute $c^*_{j|i}$ for all connections.
2. Compute $q(v_i, v_j) = \prod_{k=1}^{m} p(l_i^k, l_j^k | c^*_{i,j})$ for all connections.
3. Find the set of best edges using the *Minimum Spanning Tree* algorithm.