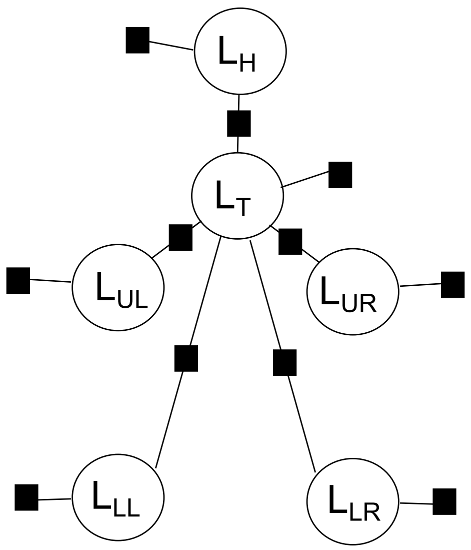
Probabilistic Graphical Models
Exercise Sheet 2

Bernt Schiele

Due Date: December 10th, 11:59pm
Submit to your solution to Anna Kukleva (akukleva at mpi-inf.mpg.de). Begin the subject of your e-mail with [PGM-Exercise2]. Only send the code that you have written (*.m files), no data files. Do not send solution that do not even run.

Pictorial Structures

The goal of this exercise is to implement a simplified version of the pictorial structures model, although using an efficient algorithm, and test it on a pedestrian dataset. You are going to use a star model over (upper and lower) legs, head, and torso as follows:



Each L_i in this distribution is actually a two-dimensional random variable representing image coordinates of the center of each body part: $L_i = (x_i, y_i)$. Scale and rotation is not considered here. Unary factors represent likelihoods

$p_i(L_i)$ and pairwise factors stand for kinematic priors $p_{ij}(L_i, L_j)$. You will learn the priors from a training dataset. Then, you will perform inference on test images, for which you are given the corresponding likelihood maps.

The first step is to download **assignment02-data.zip** from the lecture's website and import the **data.mat**. The variable `likelihoods{k,i}` is the 2D likelihood map $p_i(L_i)$ for test image number k. The images can be found in the directory `testset` for reference. The index i is defined as follows:

i	Body part
1	Lower Left leg
2	Upper Left leg
3	Upper Right leg
4	Lower Right leg
5	Head
6	Torso

The variable `train` contains the training data. More precisely, each row k of `train{i}` represents the coordinates L_i^k of body part i in training image k.

1 Learning Kinematic Priors

Points: 4

We set the prior as a 2D Gaussian $p_{ij}(L_i, L_j) \sim \mathcal{N}(L_i - T_{ij}(L_j); 0, \Sigma_{ij})$ and we define the transformation $T_{ij}(L_j) = L_j + \mu_{ij}$. This is a reasonable approximation for small joint rotations, as it is the case for pedestrians. The parameters to be determined for each prior are thus the covariance matrix Σ_{ij} and the mean μ_{ij}.

Implement the function `pairwisePots = learnPairwisePots(train)` which, for each body part i, computes the maximum-likelihood estimate of μ_{ij} as a row vector `pairwisePots{i,1}` and of Σ_{ij} as `pairwisePots{i,2}`. Note that there are only potentials between the torso and the remaining body parts, hence we assume $j = 6$ is fixed. Use the built-in Matlab functions `mean` and `cov`.

2 Maximal Marginal States

Points: 6

The goal is to compute maximal marginals of the model using the sum-product algorithm. To this end, implement the corresponding function `maxstates = sumproduct(pairwisePots, unaryPots)`. It returns a 6x2 matrix of x, y coordinates.

- Note that the graph is not a chain but a tree, so you have to think about the correct message scheduling.

- You will need computations like $f(L_i) = \sum_{L_j} \mathcal{N}(L_i - T_{ij}(L_j))g(L_j)$ in your code. However, summing over L_j is impractical due to the large state space. That is why you should implement the sum as a convolution $(\mathcal{N} * (g \circ T_{ij}^{-1}))(L_i) := \sum_s \mathcal{N}(L_i - s)g(T_{ij}^{-1}(s))$ with $s = T_{ij}(L_j)$ using built-in functions. We additionally assume a diagonal covariance so that the Gaussian is separable (simply ignore the off-diagonal entries of Σ_{ij}). The Matlab functions `fspecial`, `conv2` and the prepared file **shiftimg.m** (it shifts a given image for a given 2D offset) will be useful here. Take care when computing messages in the opposite direction.

- You can visualize your maxima using **drawmaxima.m**. You can compare your results for the first 10 images with the official solution in the directory `solution`.

- Hint: Recall that we work with (x, y) vectors but Matlab indexes its matrices by (row, column).

3 Modes

Points: 6

The goal is to compute the maximum state of the joint distribution using the min-sum algorithm (i.e. using negated log potentials). Implement the function `maxstates = minsum(pairwisePots, unaryPots)`. It returns a 6x2 matrix of x, y coordinates.

- For reasons of efficiency, we compute the minima using the generalized distance transform

$$\begin{aligned}\mathrm{DT}_{-\log[g(T_{ij}^{-1}(\cdot))]}(L_i) &= \min_s \delta(L_i, s) - \log[g(T_{ij}^{-1}(s))] \\ &= \min_{L_j} -\log[\mathcal{N}(L_i - T_{ij}(L_j))] - \log[g(L_j)].\end{aligned}$$

  You can find the code in **DT.m**, the function takes the covariance matrix of the Gaussian as the second argument.

- Unfortunately, DT does not give you the argmin, only the min. For this reason, you cannot backtrack and you need to implement the min-sum algorithm as in the lecture on max-sum algorithm earlier, i.e. computing all messages (two messages per edge) and taking a node-wise minimum. This will probably fail on potential ties (multiple modes) but that is fine in this exercise.

4 Evaluation

Points: 4

Now you are going to evaluate the model as a person detector by writing a script **evaluation.m**. To keep it simple, fix a bounding box of size 80x200px around the torso (horizontally centered at it, vertically offset in 1:2 ratio). A predicted bounding box is considered *correct* if it overlaps more than 50% with a ground-truth bounding box, otherwise the bounding box is considered a false positive detection. Ground truth can be found in the variable `GT` in the supplied mat file, each row is a rectangle $[x1, y1, w, h]$. Bounding box overlap is computed using the **boxoverlap.m** function which is provided for your convenience.

Compute bounding boxes for each test image using three ways of choosing the torso:

- Torso as the result of min-sum.
- Torso as the result of sum-product.
- Torso as the maximum of a torso's likelihood, which corresponds to using no model at all.

Your script should output the accuracy of each method, i.e. the percentage of correctly predicted bounding boxes. Note that you can visualize your bounding boxes and detections using **drawmaxima.m**.