CS B553 Assignment 3: Object recognition

Spring 2013

Due: Friday Mar 8, 11:59PM. (You may submit up to 48 hours late for a 10% penalty.)

In this assignment you'll build a program that can detect objects in images. You'll get experience working with Markov networks, implementing the max-product belief propagation algorithm, and estimating Markov potentials from labeled training data.

Please read this document carefully, as it is filled with details that are meant to help you, and you might waste time and effort on issues that are already addressed here. Please start this assignment early and feel free to ask questions on the OnCourse Forum. You may work on this assignment alone or in a partnership with another student in the class. If you work with a partner, please submit only one copy of your source code. To help get you started on the assignment, we've created some data and source code files available via OnCourse.

Background

One of the core problems of computer vision is *object recognition* (also variously known as object detection, localization, classification, categorization, or identification), in which the goal is to detect and identify the objects appearing in an image (including, for example, faces, people, bicycles, cars, trains, etc.). This problem is extremely difficult, as evidenced by the 50+ years of research that have (so far) failed to produce an algorithm capable of replicating even a child's object recognition abilities. The challenge stems from many sources, but perhaps the biggest problem is that an object's appearance can vary dramatically from image to image, due to changes in illumination and viewpoint, as well as flexibility in the object itself (e.g., a person's facial appearance changes with their emotions and facial expressions).

That said, the last few years have brought about a fairly dramatic improvement in object recognition performance, driven in large part by graphical model-based frameworks. Many objects can be naturally modeled as a collection of *parts* in some geometric arrangement. For instance, a face can be thought of as two eyes, a nose, two corners of a mouth, and a chin arranged in a particular way. An advantage of this representation is that the parts themselves tend to be relatively rigid and stable in appearance; e.g. a person's eye looks about the same whether they're smiling or frowning. The main source of variability then is in the geometric arrangement of the body parts — chins and mouth corners move a lot, for instance — and not in the appearance of parts themselves. (Of course, this is an assumption that is not really true in practice; real appearance of eyes changes dramatically when someone blinks, for instance).

A model for object recognition. Say we're given an image I that contains a face somewhere in it, but we don't know where, and we'd like to find it. We'll model the human face as a set of N parts. For each part, we'd like to estimate the position of that part in the image. This unobservable position L_i for part i will be a two-dimensional coordinate in the image — the row and column of the center of the part. We'll say that $L = (L_1, ...L_N)$ is a particular placement of the parts, and this defines a *configuration* of the model in an image. For each part i, we can define a potential function $\phi_i(L_i, I)$ that measures how well the image data around a given image coordinate L_i matches the expected appearance of the part.

Then we can use a Markov network to define a joint probability $P(L, I)$ that incorporates both this appearance model and a model of the spatial arrangement of parts in an object. It's not completely clear what graphical structure we should use here; Figure 1 shows a few possibilities. For any of these structures we can use the concepts learned in class to write $P(L|I)$ as a product over cliques of the graph. For example, for the models in Figures 1(b)-(e) we could write,

$$P(L, I) = \frac{1}{Z} \prod_{i=1}^{N} \phi_i(L_i, I) \prod_{(i,j) \in E} \phi_{ij}(L_i, L_j)$$

where E is the set of edges in the graph. Then our goal in object detection is to find the configuration of parts in a given image that maximizes this probability; i.e. to find,

$$\begin{aligned} L^* &= \arg\max_L P(L, I) \\ &= \arg\max_L \prod_{i=1}^{N} \phi_i(L_i, I) \prod_{(i,j)\in E} \phi_{ij}(L_i, L_j) \\ &= \arg\max_L \sum_{i=1}^{N} \log \phi_i(L_i, I) \sum_{(i,j)\in E} \log \phi_{ij}(L_i, L_j) \\ &= \arg\max_L \sum_{i=1}^{N} \psi_i(L_i, I) \sum_{(i,j)\in E} \psi_{ij}(L_i, L_j) \end{aligned}$$

We can find this optimal configuration using one of the inference algorithms discussed in class, like Max-Product (or Max-Sum) Belief Propagation.

We'll next need to define the potential functions. For the $\phi_i(L_i, I)$ potentials (aka the $\psi_i(L_i, I)$ log potentials), we'll use some simple computer vision techniques that are outside the scope of this course, so we'll provide source code that implements this for you (see details below). For the $\phi(L_i, L_j)$ potentials, one possible approach is to use tables as we did in Assignment 2; however in this assignment our tables would be enormous, with millions of entries. It's thus convenient to model these potentials in a parametric form, using, for example, a Gaussian distribution,

$$\begin{aligned} \phi_{i,j}(L_i, L_j) &= \mathcal{N}(L_i - L_j; \mu_{ij}, \Sigma_{ij}) \\ &= \frac{1}{Z} \exp\{-\frac{1}{2}(L_i - L_j - \mu_{ij})^T \Sigma_{ij} (L_i - L_j - \mu_{ij})\} \end{aligned}$$

where Σ_{ij} is a 2×2 covariance matrix and μ_{ij} is a 2-dimensional vector. To simplify this potential function further, we can assume that Σ_{ij} is a diagonal matrix,

$$\Sigma_{ij} = \begin{bmatrix} \sigma^2_{x,ij} & 0 \\ 0 & \sigma^2_{y,ij} \end{bmatrix},$$

so that the potential function can be simplified to,

$$\phi_{i,j}(L_i, L_j) = \frac{1}{Z} \exp\{-\frac{(L_{x,i} - L_{x,j} - \mu_{x,ij})^2}{2\sigma^2_{x,ij}} - \frac{(L_{y,i} - L_{y,j} - \mu_{y,ij})^2}{2\sigma^2_{y,ij}}\}. \quad (1)$$

In other words, we will define the potential function $\phi_{i,j}$ in terms of four scalars: a horizontal mean $\mu_{x,ij}$, a vertical mean $\mu_{y,ij}$, a horizontal variance $\sigma^2_{x,ij}$, and a vertical variance $\sigma^2_{y,ij}$.

Writing your own object recognition engine

Your goal in this assignment is to implement a simple program for object recognition, using the above framework based on Markov networks.

Language. Unless you have a strong preference otherwise, we suggest using C or C++ for this assignment, because implementations in other languages will likely be quite slow. To help get you started, we've provided some code (available via OnCourse) that implements some basic image manipulation functionality. You can also use libraries for common data structures and algorithms (e.g. trees, vectors, sorting algorithms, etc.), as long as you implement the Markov net learning and inference (described below) yourself. (So, for example,

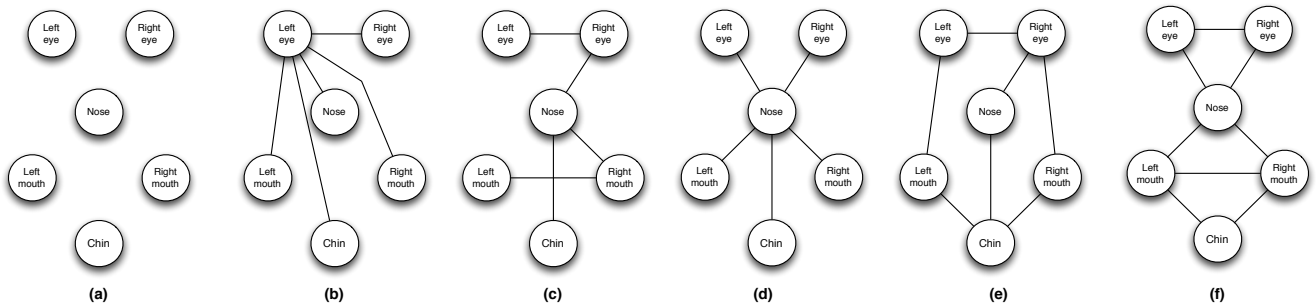

Figure 1: Some possible graphical structures for a Markov model of a face.

using the C++ Standard Template Library for convenient data structures is fine, but using a graphical model inference library or a computer vision library is not.)

Data. We've also prepared a dataset of training and testing data for your use, also available on OnCourse. These sets include images with known (ground truth) position coordinates for each part in each image. The files `gt.train` and `gt.test` contain these ground truth coordinates, wherein each line has the form:

```
image_number l_eye_col l_eye_row r_eye_col r_eye_row nose_col nose_row l_mouth_col l_mouth_row r_mouth_col r_mouth_row chin_col chin_row
```

and where `image_number` refers to an image file inside the `imgs/` directory.

Step 1: Naive recognition. We've implemented some code that will compute the $\psi_i(L_i, I)$ log potential functions for you. (This code involves using some computer vision and image processing techniques that are beyond the scope of this course, so you can treat this code as a "black box" without understanding its internal functionality.) As a first step, this code also implements inference on the naive Markov model in Figure 1(a). To try it out, download the source code and image dataset from OnCourse. On a Mac or Linux machine, run:

```
mkdir a3; cd a3
unzip ../a3-skel.zip
make
./a3 gt.train gt.test
```

The result will appear in the `naive_result.png` file; as you'll see, the naive inference finds one or two parts correctly but otherwise does not work very well.

Step 2: Learning. To perform recognition with the more sophisticated Markov models of Figure 1, we need to learn the parameters of the potential functions $\phi_{i,j}(L_i, L_j)$ given in equation (1). This can be done from training data using the standard formulae for estimating the maximum-likelihood parameters (mean and variance) of a Gaussian function. To help you out, we've provided labeled training data for a small set of a few dozen images. The training data includes the row-column coordinates of each face part in each image. This basic procedure can be used to learn the potential functions needed for any of the graphical structures in Figure 1.

Step 3: Recognition with exact MAP inference. Now implement recognition by performing MAP inference on Markov networks, using the Max-Product (or Sum-Product) Belief Propagation algorithm described in class. Implement inference for each of the three networks shown in Figures 1(b), (c), and (d). As in step 1, use the functions in the C++ code provided to visualize the detected part locations.

Step 4: Recognition with approximate MAP inference. Exact inference on the model in Figure 1(e) would be prohibitively expensive due to the cycles in the graph. Implement approximate inference on this model using loopy Max-Product (or Sum-Product) Belief Propagation, as described in class.

Step 5: Evaluation. We've provided a separate test dataset in addition to the training dataset. This set also includes images labeled with correct part locations, so that you can test and compare the performance of the recognition algorithms implemented above. Report the recognition accuracy for each of the above algorithms according to the percentage of parts that were located correctly, where "located correctly" means "within 20 pixels of the correct location." Make sure to include these figures in your project report.

Important hints and warnings

Design decisions. You'll likely encounter some decisions as you write your programs that are not specified in this assignment. Feel free to make some reasonable assumptions in these cases, documenting them in your report, or to ask on the OnCourse forum for clarification.

Academic integrity. The goal of this assignment is to help you learn about Markov networks and object recognition. **We thus require that you (and your partner) design and implement the recognition program yourselves.** You and your partner may discuss the assignment with other people at a high level, e.g. discussing general strategies to solve the problem, talking about language syntax and features, etc. You may also consult printed and/or online references, including books, tutorials, etc., but you must cite these materials in the documentation of your source code and make it explicitly clear exactly which parts of your code were influenced by which sources. **However, the code that you (and your partner, if working in a group) submit must be your own work, which you personally designed and wrote. You may not share written code with any other students except your own partner, nor may you possess code written by another person who is not your partner, either in whole or in part, regardless of format.** Please note that copying or modifying code from a student, from online sources, or from a textbook is a violation of the academic integrity policy. As described in the course syllabus and the IU Academic Handbook, University policy requires us to investigate suspected cases of academic integrity, to assign grade penalties, and to report the case to the Office of the Dean of Students for possible additional sanctions (up to and including expulsion from the University).

That said, the course staff wants you to succeed and is very happy to help if you have questions about the assignment. Please stop by office hours, write a question on the OnCourse forum, or set up an appointment to meet with us if you have concerns or questions.

What to turn in

Turn in two files, via OnCourse:

1. A .zip archive file containing your source code. Make sure your code is thoroughly debugged and tested. Make sure your code is thoroughly legible, understandable, and commented. Please use meaningful variable and function names. Cryptic or uncommented code is not acceptable; we can't grade your assignment unless we can understand how it works!

2. A separate text file or PDF with a brief report about your work. Explain, at a high level, how you implemented it, what design decisions and other assumptions you made, any problems you had to overcome, etc. Give credit to any source of assistance (students with whom you discussed your assignments, instructors, books, online sources, etc.). Also give the accuracy details requested above (on the results of your testing on our sample datasets).

Extra credit. We may give extra credit to assignments that implement more sophisticated features than the ones described here. Make sure to describe these extra features in your documentation file. For example, you could try to perform exact inference on the network in Figure 1(f).