Day 2 Summary.

Lecture 1. Segmentation is hard. But consider a different problem – classify pixels as sky, vegetation, road, building, or other. Use groundtruth labels specified by a human. Learn the conditional probabilities of filter responses conditioned on the labels (represented non-parametrically). Use color filters and texture filters separately, or in combination. Use Bayes to do classification – estimate most probable label for each pixel. Results show that sky, vegetation, and road can be detected (classified) fairly well even using simple filters. This is partly because they are typically spatially uniform – one part of the sky looks like most other parts. Buildings are not well detected- because different buildings look different, and different parts of the same building look different (e.g., doors, windows, façade). In short, we can learn features which are invariant (approximately) to different positions and types of vegetation (and which differ from the feature responses on sky, road, and so on). But we cannot find features which are invariant to different buildings and positions on buildings. Note that assume that labels are independent between pixels – but this ignores important information because neighboring pixels typically have the same label, and also there are spatial relations between labels (e.g., sky must be above the road and not below the road). More recent, state of the art models use this information. But image labeling, for some classes is surprisingly easy – much easier than general segmentation (true also for medical images). See `LabelingRegions.pdf’.

Lecture 2. Markov Random Fields.

The models we have used so far (except TV-norm) have ignored spatial context – neighboring pixels in an image typically have the same labels (e.g., a sky pixel is almost always next to another sky pixel). How to model this spatial context? One way is to define a Markov Random Field (MRF). For example, we define the probability of the labels for every pixel of the image by a Gibbs distribution with an energy function which contains unary terms, which depend only on the label of each pixel (and the image), and binary terms which depend on the labels of neighboring pixels (and maybe the intensity). The unary terms are like the models we used before (for image labeling, lecture 1 on day 2, and for edge detection, lecture 3 on day 1). They can be learnt in a similar manner – by learning the conditional probabilities of features responses conditioned on the label (or by regression). The binary terms contain local context. They can pay a penalty (give high energy, or low probability) if neighboring pixels have different labels (i.e. they encourage the labels to be spatially smooth). This penalty can be reduced if the two pixels have very different intensity (i.e. if there is evidence that there is an intensity edge separating the two pixels). Note that we can convert the TV-norm energy function (lecture 4 day 1) into an MRF by discretizing space (replacing the continuous positions x,y by discrete lattice positions i,j – replacing derivatives by differences) and by discretizing the values of w(x). This shows that TV-norm corresponds to an MRF with very local interactions – this is one reason why TV-norm is no longer the state of the art for image denoising because current methods use longer range interactions. More generally, an MRF is defined on a graph G with nodes/vertices V and edges E (linking the vertices). There are state variables defined on the nodes. The edges show which state variables directly influence each other. The Markov property is that a node is only connected by edges to a subset of all the other nodes. This means that the conditional probability of the state variable at the node conditioned on the state of all other nodes of the graph only depends on this subset of nodes (connected by edges).Note that MRF’s are a special case of graphical models – used for probabilistic inference on graphs. See `Tutorial\_Bayesian\_Inference’.

Lecture 3. Inference on MRF’s: Max-Flow/Min-Cut and Belief Propagation. When we have an MRF P(x) we need to compute its most probable state, and sometimes compute marginal probabilities of the state variables at each graph node. Max-Flow/Min-Cut is one class of algorithm. It can be proved to converge to the correct results for some MRF’s. It is illustrated for a binary MRF (where state variables at nodes take only two possible values). The energy of the MRF is first written so that it only has non-zero terms when neighboring graph nodes (connected by edges) have different states. This formalizes a min-cut problem. This can be converted to a max-flow problem which tries to maximize flow between a source and a sink with capacities on pipes that limits the flow. The max-flow solution is equivalent to finding the cut where the limited pipe capacity has a bottleneck restricting the flow. This max-flow problem can be solved by a range of algorithms – often called Ford-Fulkerson. Belief Propagation is another class of algorithms. There are two forms – sum product (Pearl) which estimates the marginal probabilities and max product (Gallager) which estimates the most probable states. Both methods propagate messages between nodes and read off expression for the marginal probabilities after the messages have converged. The fixed points of sum-product correspond to extrema of the Bethe Free energy (which relates to the literature on variational methods). Belief propagation is guaranteed to converge to the correct solution for MRF’s which are defined on graphs with no closed loops (for these graphs, dynamic programming also converges correctly). Belief propagation often converges to a good approximation on MRF’s whose graphs do have closed loops. See `MRFs&Inference.pdf’. See `Review\_Vision\_Models.pdf’. See `GrabCut\_Examples.pdf’.

Lecture 4. Learning MRFs without Hidden Variables. If we have an MRF where we can observe all the variables (i.e. no variables are hidden) then we can learn the probabilities, or equivalently the energy function, by maximum likelihood (ML) learning from a set of training examples (assumed to be independent). We express the MRF as an exponential distribution which depends on statistics phi() of the data and parameters lambda which must be learnt. ML estimates the parameters lambda to maximize the probability of generating the data. This corresponds to minimizing an ML criterion which can be shown to be convex and to have a single minimum (but minimizing it is still difficult because of the difficulty of doing inference on MRFs). We show that doing ML, minimizing the ML criterion, corresponds to finding the parameter values so that the expected value of the statistics (expectation with respect to the model) matches the statistics of the data (i.e. phi() averaged over the data examples). E.g., if we have a Gaussian distribution, then the statistics are x and x^2 – the expected value of x w.r.t. the model is the mean mu which we set to be equal to the average (1/N) \sum \_i x\_i of the data examples). If we use the statistic to be the histogram of the difference between intensity at neighboring pixels (e.g., lecture 2,3 on day 1) then ML gives us a MRF where neighboring pixels are connected -- the characteristic form of the histogram (strong peak at 0) gives an MRF which is piecewise smooth (like the TV-norm). If we apply this to learning models of text (like Shannon) then if we use statistics which depend on the frequencies of pairs of letters, then we get a MRF for text which has nearest neighbor structure. Now consider the harder problem when we do not know which statistics to use – e.g., for text maybe we use the frequencies of letters, or of pairs of letters, or triples of letters, or? First, suppose we have two possible statistics we can use – for each statistics we use ML to learn the best lambda, this gives us two probability distributions. Then we compute the probability of the data using both models – and select the model which has higher probability of generating the data. (Note that the probability of generating the data, with the parameter estimated by ML, to equal to the entropy of the probability model and so a model with low entropy – i.e. less uncertainty in predicting the data – will be selected). This idea can be extended to allow for feature pursuit which starts with a dictionary of possible statistics. For each statistic we perform ML to estimate the best parameter for each model (total number of models is the size of the feature dictionary). Then we do model selection to pick the best model. Then we try to grow this model by adding another statistic (and parameter). We start with a large number of models (equal to the size of the dictionary) which contain the statistic already chosen and another statistic from the dictionary – and do ML to estimate the best parameter values. We select the best model by model selection (i.e. the model that best matches the data). Then we proceed to add a third statistic, learn models by ML, do model selection, and so on. As we add extra statistics we always get a model that fits the data better (because the models with more statistics have more parameters to adjust) so we have to add a penalty which penalizes models by their number of statistics. We stop adding models when the improvement in adding a new statistics falls below a threshold. Feature pursuit is used for modeling tecture by SC Zhu, YN Wu, and D. Mumford – the statistics are the histograms of filters, and the methods must select a few filters from a large dictionary. The texture gets better as the feature pursuit adds more filter. See `FRAME.pdf’.

Lecture 5. Learning with Hidden Variables. We represent probability distributions as exponential distributions with parameters lambda and statistics, as before, but the statistics depend both on observed data and on unobserved, or hidden, data h. We formulate this by ML of the lambda parameters, similar to Lecture 5, but we obtain the probability distribution by summing out the hidden variables (usually impractical). We now apply a trick – we introduce a new variable q(h) which is the probability distribution over the hidden variables h. We add a Kullback-Leibler term to the ML learning criterion between the distribution q(h) and the distribution of the hidden variables of the exponential model. This gives a criterion F[lambda,q] whose global minimum corresponds to the ML estimate of lambda (but F usually has many minima). We then search for the minimum of F[lambda,q] by minimizing alternatively with respect to lambda (with q fixed) and q (with lambda fixed). This is the Expectation Maximization (EM) algorithm. Estimating q, for fixed lambda \_t, give q(h) = p(h|d,lambda \_t). Estimating lambda, with fixed q, is done by choosing the value of lambda so that the expected statistics of the exponential model are equal to observed statistics of the data and the expected statistics of the hidden variables (with expectation taken with respect to q). EM is extremely general, one example is Hidden Markov Models (HMMs) used for speech recognition, modeling vision sequences, and many other applications.