CS294-6 (Fall 2004) Recognizing People, Objects and Actions Lecture: February 17, 2004

Digit Recognition and Distance Transforms

Lecturer: Jitendra Malik *Scribe: Roger Bock*

Digit Recognition

- Shape Context Deformable Template Nearest Neighbor
  - Find correspondences
  - Align using TPS (Thin Plate Splines)
  - Use nearest neighbor classifier
    - Need to have distance function
      - Shape context distance between corresponding points
      - Appearance distance between corresponding points
        - Use blocks of pixels – i.e. 3x3 or 5x5
      - Bending energy
    - Sum distances over all points
      - Use weighted combination of three different metrics (α, β, and γ)
      - How do we tune the parameters?
        - Cross validation
          - Split your data into training data and testing data
          - Try different choices of parameters on training data
            - Quantize the parameter values and do a search
          - Performance is judged using testing data
          - Using same data for training and testing sets is cheating
  - Nearest neighbor classifiers
    - K nearest neighbor classifiers
      - Pick the k nearest (1-NN, 3-NN, 5-NN) and then have a majority vote from among them.
  - 10,000 digit test set
  - .63% error rate
- Convolutional Neural Nets
  - .82% error rate
- Tangent Distance Nearest Neighbor
  - Different data set, but similar error rate
- Support Vector Machines
  - DeCoste and Scholkopf
    - Paper on course website
    - Virtual support vectors
  - .56% error rate
- Decision Trees
  - Decision trees are classifiers
    - Work by having a test that is a function of the features

        - Test returns either true or false
        - Take appropriate branch
        - Another test on features
        - At some point you have a leaf where there is no more testing to be done
        - At this point you make the decision, i.e. this is the digit ‘3’.
            - Alternately you could state probabilities: Pr(‘5’) = .7, Pr(‘2’) = .1, Pr(‘4’) = .2, others zero
        - Using tree is easy, implement tests, keep traversing tree until you hit a leaf
            - Very fast
            - Cost is expected cost of test times number of levels in the tree
    - How do we train a decision tree (in general)?
        - For training a decision tree we have a whole bunch of training examples
            - Each example looks like <f_0, f_1, f_2, …, f_{10}, y> where y is a class label.
        - Ideal test is one where all chairs would go down one branch and all non-chairs go down opposite branch.
        - By the time you hit a leaf you should have as close to having “pure” subsets as possible
            - Pure meaning all at leaf are of one type
        - How do we measure purity/impurity?
            - Entropy
            - $\sum(Pi * \log(Pi))$ for i=1 to k where k is the number of classes
        - Let’s say for entire data set we have entropy H_0
        - After asking question the data is split into D^+ and D^-
        - Figure out entropy for D^+ and D^-
        - Entropy after split is $\alpha * H(D^+) + (1-\alpha) * H(D^-)$
        - Entropy reduction = $H_{before} – H_{after}$
            - Also referred to as information gain.
        - Pick tests that maximize entropy reduction at each stage
        - Greedy technique, at each stage look for best test
        - Infinite set of tests!
            - Use axis parallel cuts to reduce number of possible tests
            - i.e. $f_0 > 0$?
        - Other possible tests include comparison to stored prototypes – tests can be anything you want
    - What kinds of tests make sense for vision?
        - Look for specific arrangement of pixel values in certain geometric configuration with each other
            - Tags
                - Think of as a local edge
                - Description of local image window
                - i.e. horizontal edge, vertical edge, endpoint, etc.
            - Arrangements
                - Between local image patches
                - Quantize orientations into eight possibilities
                - Each zone is 360 / 8 = 45 degrees in size – no overlap

        - Can say things like Patch2 is northeast of Patch1 and Patch3 is east of Patch1
        - Also can look at distances, or ratios of distances
    - How pick tags?
      - A decision tree is used to pick the tags
      - 4x4 pixel window
      - Classify all possible 4x4 pixel windows into tags
      - Dealing with binary black and white images
      - Want tags to be discriminative
      - Split using entropy without taking into account label of category – i.e. which tag splits data most equally?
      - A given pixel can belong to more than one tag – all the ones from its leaf where all 16 pixel values are specified to the root node where only 1 pixel value is specified
      - Tags end up being edges or terminators
      - Typically, only go four levels down – no more than four pixels constrained, rest are don't care
    - An aside - consider all possible binary images with 3x3 windows
      - 2^9 possible windows
      - What windows do you see most often?
        - All white and all black
        - Most of the world is boring
        - Next most common will be edges
        - Next most common will be corners
        - Least common will be checkerboard pattern
      - For handwriting recognition, this distribution was tuned to data
    - Why not use simple edge detectors?
      - "Learning chauvinists" – don't want to build in any knowledge
    - Relations are quantized between pair of tags, just one of eight relative orientations
    - Questions will be of the form, "Is there any instance of this specific tag arrangement in the image?" i.e. Tag type 4 and 3 north of tag type 1 and tag type 2 southwest of tag type 1
  - Now we have a computational complexity problem
    - 62 possible tags
    - Top level has 62*62*8
    - Concept of minimal extension
      - Each child test is an extension of the test you already have
      - Add an edge, or another tag to the previous test
    - Even with this computational complexity is quite high
    - Randomized decision trees
      - Don't try to find single best decision tree
      - Use only part of the data to come up with the decision tree
      - Build up many different decision trees
      - Each tree is noisy

                - Best decision tree had error rate of 7%
                    - Average error rate of 10%
                - Average predictions of all the decision trees
                - Aggregate classifier had error rate of under 1%
                - Similar to “committee of experts” work
                    - Build up good estimator from lousy estimators as long as errors are uncorrelated
                    - Errors are uncorrelated here because the different trees used different training data
        - Generalization
            - 3x3 windows aren’t fully specified – they have don’t care pixels
            - Range of possible angles and distances allowed for by arrangements
        - Order Structure
            - Work by Carlsson and Sullivan
            - Given four points and four lines (one through each point)
            - Each point has some relation to each line (left side vs. right side)
            - Defining relationships between points and lines
            - A small perturbation of diagram won’t change description
            - Big perturbations will
            - Related to the idea of rank from statistics
                - Largest number gets N, smallest gets 1
                - Tests are done on rank, instead of actual number.
                - Much more robust
                - This is for a line
                    - In a plane or in 3D, becomes harder
                    - Order structure is the attempt to do rank in 2D
        - Different data set, but similar error rate

Distance Transforms

- Chamfer Distance
    - Proposed in 1977 by Barrow et al
    - Want to match to images to each other
    - Their application was matching images from an airplane with images from maps
    - Want to align to boundaries to each other
    - Different types of data, one is an image, other is lines on a map
        - Can’t just do SSD
    - Run edge detector to get contours
    - How align the two shapes?
    - One shape is projection of 3D shape.
    - Fiddle with parameters to get two curves to line up
    - Fundamentally different from correspondence based approach
        - Correspondence based approach tries to find best match for every point
    - Distance based, just find closest point in other set
    - For each point on one curve, measure distance to closest point in other set
        - Distance is zero at intersections
    - How do we aggregate distances?

        - Take average?
        - Take maximum?
    - Consider straightforward version where we take some kind of average
    - One curve is fixed, other curve is swung around until it lines up best
    - Precomputation on fixed curve
        - Auxiliary array D(x,y) over whole image where D(x,y) is distance to nearest edge point
            - Values are zero along contour
            - Perpendicular to edge, values will go …, 3, 2, 1, 0, 1, 2, 3, …
            - Back in the day they had to use integers, so we double everything to avoid real numbers, take 1.5 as approximation of $\sqrt{2}$, and get a basic pattern of:

              3 2 3
              2 0 2
              3 2 3
            - Forward pass (L->R, top->bottom) and backward pass (R->L, bottom->top)
                - Take smallest value
            - Approximating real Euclidean distance
    - Once you have precomputed array, you can put down another contour and compute its cost
    - Essentially a correspondence to blurring
    - A chamfer in woodworking is a groove
    - Local maximum between two edges – "Medial Axis Transform"
    - Another term is "Voronoi Surface" of a set of points
        - Assume we have a finite set of points
        - Picture a graph
        - Every pixel has some height that is D(x,y)
        - What will be nature of surface?
            - Around one point it will be a cone
            - Cones will intersect and cut each other off
        - Voronoi diagrams are dividing space into polygons around sites
            - Each polygon contains points closest to the site within it
- Hausdorff Distance
    - h(A, B) is the directed distance from A to B
    - h(A, B) is max over (a elements of A) min over (b Elements of B) ||a-b||
    - For each point on A, find its closest point on B
    - Now look at max distance over all A points
    - This is directional, A to B might be different from B to
        - Not a symmetric function – not a distance!
    - Make it symmetrical by taking max of h(A, B) and h(B, A)
    - kth directed distance would be to take the kth percentile (say 75%) that ignores outliers
    - Standard notion in math
    - Algorithms for computation come from Hultenlocher, Klanderman, and Rucklidge

    - How find translation that minimizes Hausdorff distance?
    - Take one shape, sweep over the other
- Philosophical difference between distances of today and distances of last lecture
    - Before, wanted nose to line up to nose
        - Requires much more “mumbo jumbo”
    - The algorithms presented today don’t care if the nose lines up to the nose
        - Probably faster but not as good
    - In both cases, work well when you don’t have to consider a large set of transformations