Klyde Breitton, Aayush Gupta
CIS 581 Final Project, Option 1
12/21/14

Face Detection:

HOG (HOG.m, spread.m or spread_mex.mexa64) -- This function was completely implemented on our own, vectorized and everything. It takes in a 36x36 grayscale window and outputs a 1x900 feature vector. A 2x2 block size, 6x6 cell size, and 36 x 36 window size was assumed. Also, billinear interpolation was used to split a pixel's gradient magnitude into the histogram bins, meaning a pixel filled only 2 bins at a time closest to its own gradient direction.

This bilinear interpolation was done with spread.m. A MEX function was also made and used that's only compatible with Linux -- if you'd like to use that instead if you have Linux (it'll be faster), change line 56 in HOG.m from

cellSpread = @(oneCdir, oneCmag) arrayfun(@spread …

to

cellSpread = @(oneCdir, oneCmag) arrayfun(@spread_mex ...

SVM (train_HOGs.m, various liblinear functions) -- Training data was obtained from a compiled course kit at Brown (http://cs.brown.edu/courses/cs143/proj4/proj4.zip). Ultimately 6700-ish training faces were used and 172,000-ish training not-faces were used, where the not-faces were obtained by running a sliding window over 274 provided not-face images.

LibLinear was used to train the data and to make predictions. This was perfectly adequate, since for our purposes we are only concerned with L2 norms (linear kernel). It also ran much faster than LibSVM.

As a script, train_HOGs gets the images you want to use a training data, trains them in a linear SVM model, and outputs the svm model.

Actually, logistic regression was used. This was because logistic regression can also output an interpretable probability, so that you can set a threshold where e.g. say a face is really a face is at $p > 0.95$.

Face detection (facedetect.m, face_detector.m, svmpredict_prob.m) -- facedetect is a wrapper for face_detector, to try face detection at different scales. It takes in the image you want to detect and the SVM model used for predictions, and outputs a bunch of things, but

most important are the bounding boxes of each detected face and the cropped images from the bounding boxes. Since we only train on 36x36 windows, the input image has to be scaled down, detect faces at multiple scales, then blow it back up to normal size and remove any overlapped detections.

face_detector.m takes an image, the scale every iteration (currently .7), the start and end scales you want to try, the face threshold probability, and the SVM model, and outputs the bounding boxes, their probabilities, and HOGs of any faces detected. The function works by sliding a window centered only over edge pixels of a given image, since a face must contain an edge. This reduces time further than a normal sliding window. As long as you catch a nose edge, you should reliably get window that should find a face. We set the threshold set for edge detection depending on the scale of the image to always get that nose edge. svmpredict_prob.m gives each observation's (window's HOG) probability of being a face.

Results

Results are in the folder 'test_results', and for face detection are labeled "detect_x.jpg" where x is the image number. Most of the time detection worked reliably, with few false positives. Some problems were cut-off faces not being detected, some faces just not being detected at all (the weird bark-fellow, Jennifer Lawrence's side-face, Mystique, Samuel L. Jackson, etc.). Most probably, failed detections occurred due to a high probability threshold set (.98), though if it were set lower, more false positives were observed. There was also a freak output for detect_15.jpg, where multiple bounding boxes were found for the same face. This is probably because our detector really likes eyes, for some reason, and our getting-rid-of-overlap procedure overlooks that (which works by comparing bounding boxes of similar centers, but the eye is always off the face center).

<u>Face Replacement:</u>

Convex Hull (convexHull.m) – To get the convex hull of the face we used a third party library by Masayuki Tanaka that detects all the features of a face i.e. left eye, right eye, mouth and nose. The library returns bounding boxes for these features, the corners of which are used for 2 purposes:

1. To get the convex hull of the face; which is the enclosure of all these points. This hull is the mask image with 0's and 1's.

2. As control points for TPS morph.

TPS morph (morph_replace.m, est_tps.m, morph_tsp.m) – We use the TPS morph from the second assignment to morph the source image to the target image. After the morph is

complete, we copy the face from the morphed source image and paste it on the target image's face/faces.

Blending (within morph_replace.m) – We use Poisson Image Editing code from Masayuki Tanaka to perform a Poisson Gaussian Seidel blend using the mask that we obtained from convexHull.m and the gradients of the two images.

Everything (face_replace_wrapper.m) -- This takes every relevant function used. The input is a source image and a target image, the source image being that of an image containing a single face which you'd like to replace in all faces detected in the target image. The function outputs the final replacement.

Results -- Results for face replacement are stored in the folder 'test_results', as 'face_replace_x.jpg' where x is the image number. Where images are missing are when face replacement could not occur due to failure to detect a face.