CS221 Artificial Intelligence: Principles & Techniques

Challenge Problem 2

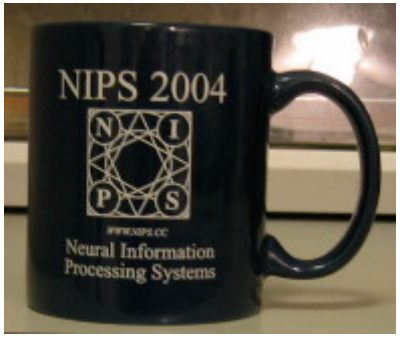
Object Recognition and Tracking

Ian Goodfellow
Olga Russakovsky
Stephen Gould
cs221qa@lists.stanford.edu
October, 2009

Overview

- Challenge problem
  - Problem statement
  - Starter code overview
  - Milestone requirements
  - Milestone techniques
  - Object recognition tips and tricks
- OpenCV tutorial
  - Introduction and installation
  - Code samples

Challenge Problem



Starter Code Overview

`classifier.cpp`	Defines the Classifier class. You are free to modify this file, but do not modify the interface to the loadState() and run() methods.
`CXMLParser.cpp`, `replay.cpp`	Contains code for replaying object labels (such as ground truth labels). **Do not modify these files.**
`evaluate.cpp`	Scores detections generated via test.cpp. **Do not modify this file.**
`objects.cpp`	Contains the data structures for annotated objects. **Do not modify this file.**
`test.cpp`	Contains main() code for testing classifiers on videos. **Do not modify this file.**
`template.cpp`	Contains code that performs template matching for you.
`train.cpp`	Contains training starter code. You are free to modify everything in this file, or even replace it with multi-stage training.
`utils.cpp`	Contains useful utility functions.

Command-line Options

- **`train [<options>] <directory>`**
  - <directory> is the root directory containing (subdirectories of) all the training images
  - -c <filename> writes learned parameters to a file after training using **`Classifier::saveState()`**
  - -h provides help
  - -v gives verbose output

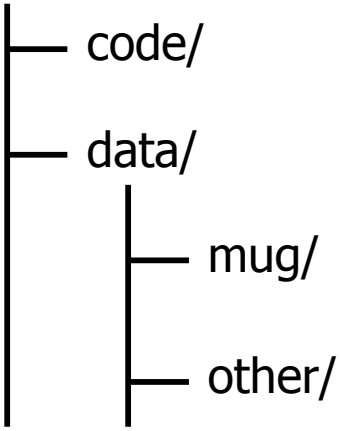
- **`test [<options>] <video directory>`**
  - <video directory> is the name of the video you want to test on (e.g. "vision/data/easy")
  - -c <filename> configures the classifier with parameters from a file using **`Classifier::loadState()`**
  - -g <filename> displays ground truth labels from an XML file
  - -h provides help
  - -o <filename> saves classifications to an XML file (same format as –g)
  - -v gives verbose output
  - -x disables display of the video (if you don't have X-windows)

Classifier Class

```
class Classifier {
protected:
    CvRNG rng;
    CvMat *parameters;
    // TO DO: ADD YOUR MEMBER VARIABLES HERE

public:
    // constructors and destructors
    Classifier();
    virtual ~Classifier();

    // load and save classifier configuration
    virtual bool loadState(const char *);
    virtual bool saveState(const char *);

    // run the classifier over a single frame
    virtual bool run(const IplImage *, CObjectList *);

    // train the classifier using given set of files
    virtual bool train(TTrainingFileList&);

protected:
    // TO DO: ADD YOUR MEMBER FUNCTIONS HERE
};
```

Training File Lists

```
typedef struct _TTrainingFile {
    std::string filename;                // full path to image file
    std::string label;                   // subdirectory name
} TTrainingFile;

typedef struct _TTrainingFileList {
    std::vector<TTrainingFile> files;    // list of files
    std::vector<std::string> classes;    // list of classes (subdirectories)
} TTrainingFileList;
```



CObject Class

```
class CObject {
public:
    CvRect rect;              // object's bounding box (x,y,width,height)
    std::string label;        // object's class

public:
    // constructors
    CObject();
    CObject(const CObject&);
    CObject(const CvRect&, const std::string&);

    // destructor
    virtual ~CObject();

    // helper functions
    void writeAsXML(std::ostream&);
    void draw(IplImage *, CvScalar, CvFont *);
    CvRect intersect(const CObject&);
    int overlap(const CObject&);

    // operators
    CObject& operator=(const CObject&);
};
```

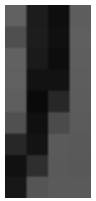
Milestone Requirements

- Build a logistic regression classifier for the "mug" class using fragment-based features
  - Load positive and negative training images
  - Convert to grayscale
  - Resize to 32-by-32
  - Extract fragment-based features (we give you the feature definitions and template matching code)
  - Train logistic regression model
  - Implement test-time code to run classifier over all scales (you can assume height = width for the milestone) and shifts (in increments of 8 pixels) within each video frame

- **Remember:** after the milestone you are free to use whatever features and classifiers you like

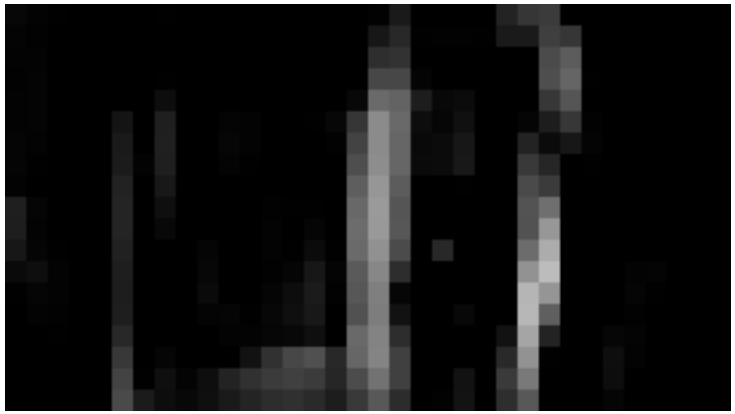
Template matching

- Given a small template (image fragment), we can plot how well each location in the image matches this template. The code to do this is in template.h



Example template

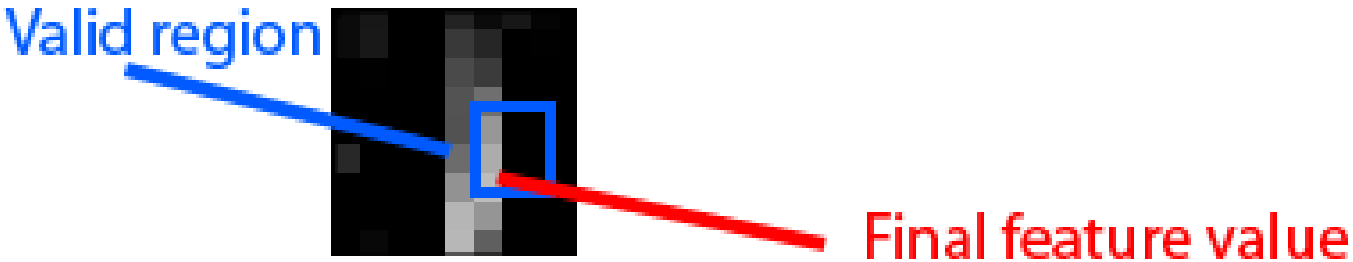
Example input image

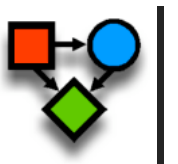


Example template response (whiter pixels indicate better match of template to input image)

Max-pooling

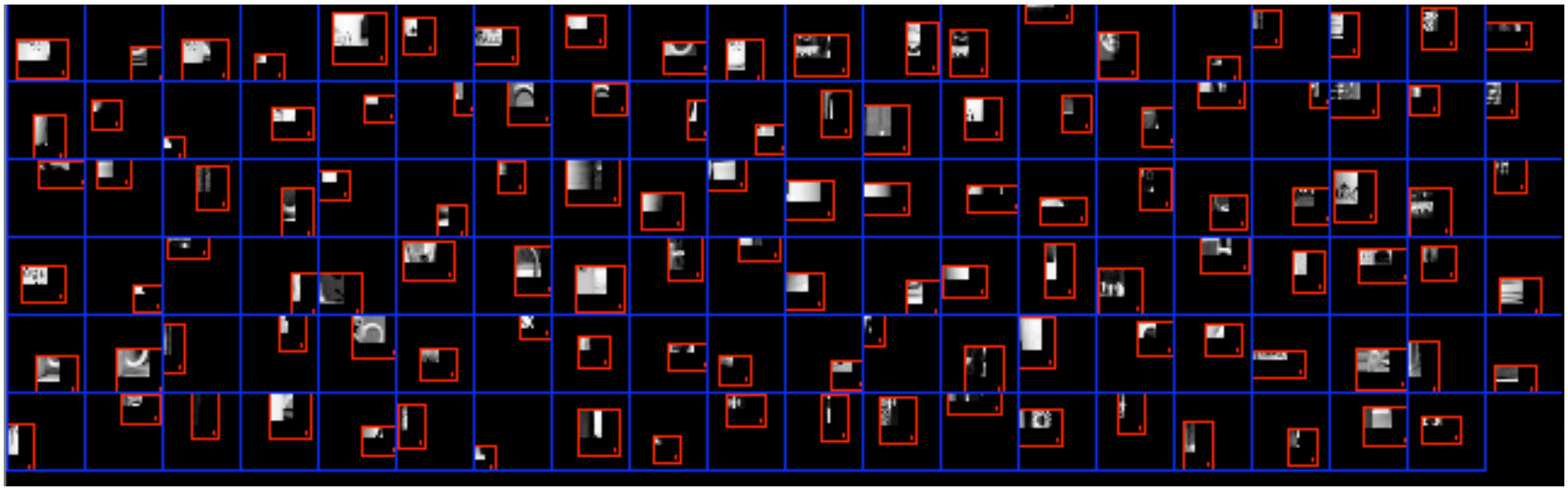
- Each feature is just a value read directly from the template response image
- Each feature is read from within a specific valid region (a sub-rectangle within the 32x32 image to be classified). The value of the feature is the brightest pixel in that region. This allows the feature value to remain the same if the shape of the object changes slightly.
- This is the only aspect of feature extraction that you need to implement in order to train your model-- we give you everything else
- To generate a full feature vector, repeat this for each template. Don't forget that each template has a separate valid region.





PatchDictionary

- We give you a collection of templates and valid regions to use for your feature extractor



Sliding window object detection

- For each size of re-scaled image, slide a 32x32 pixel "window" across the image
- At each window position, extract a feature vector and classify it using logistic regression
- You will need to implement this yourself, in Classifier::run

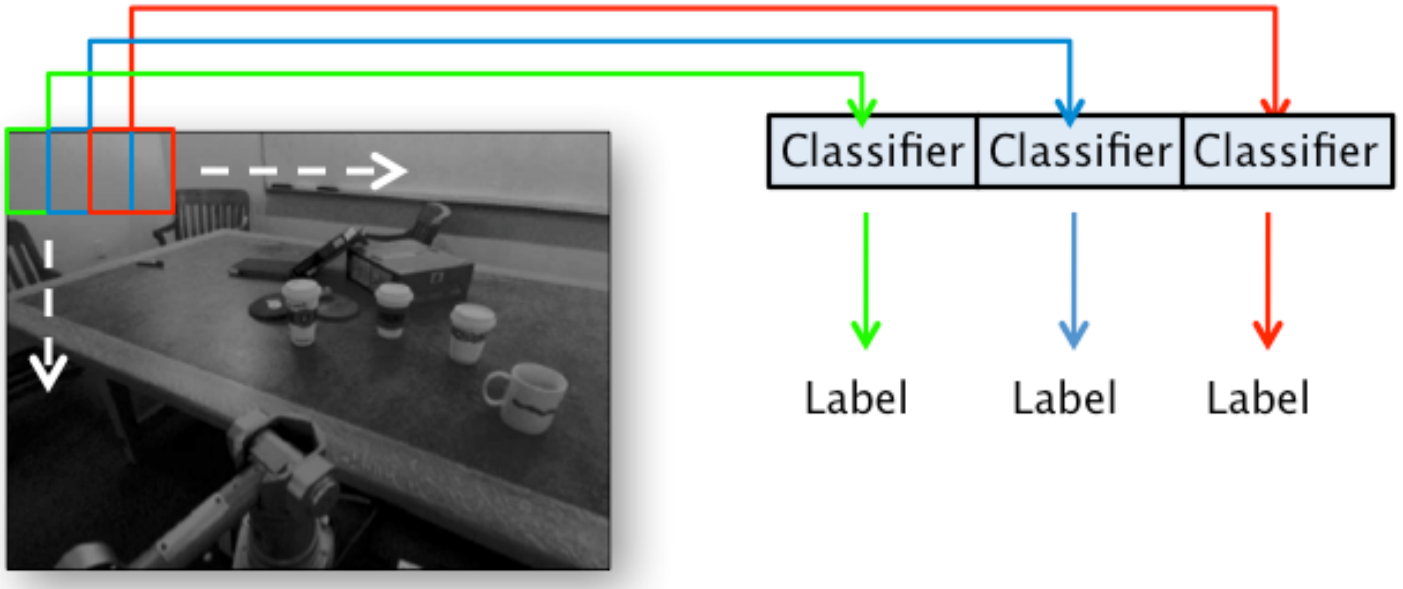
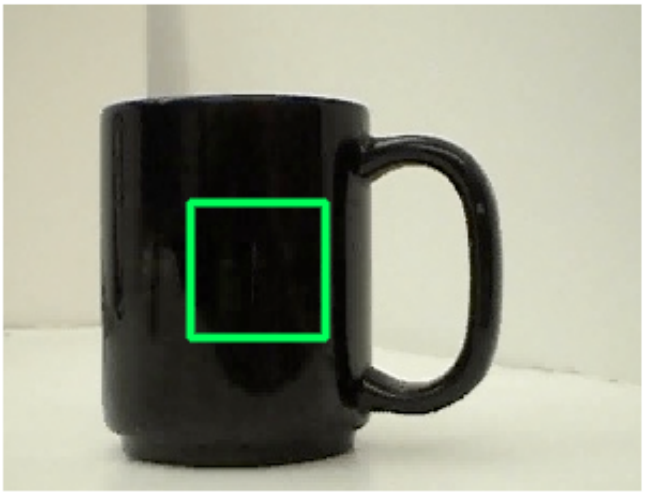


Diagram by Adam Coates

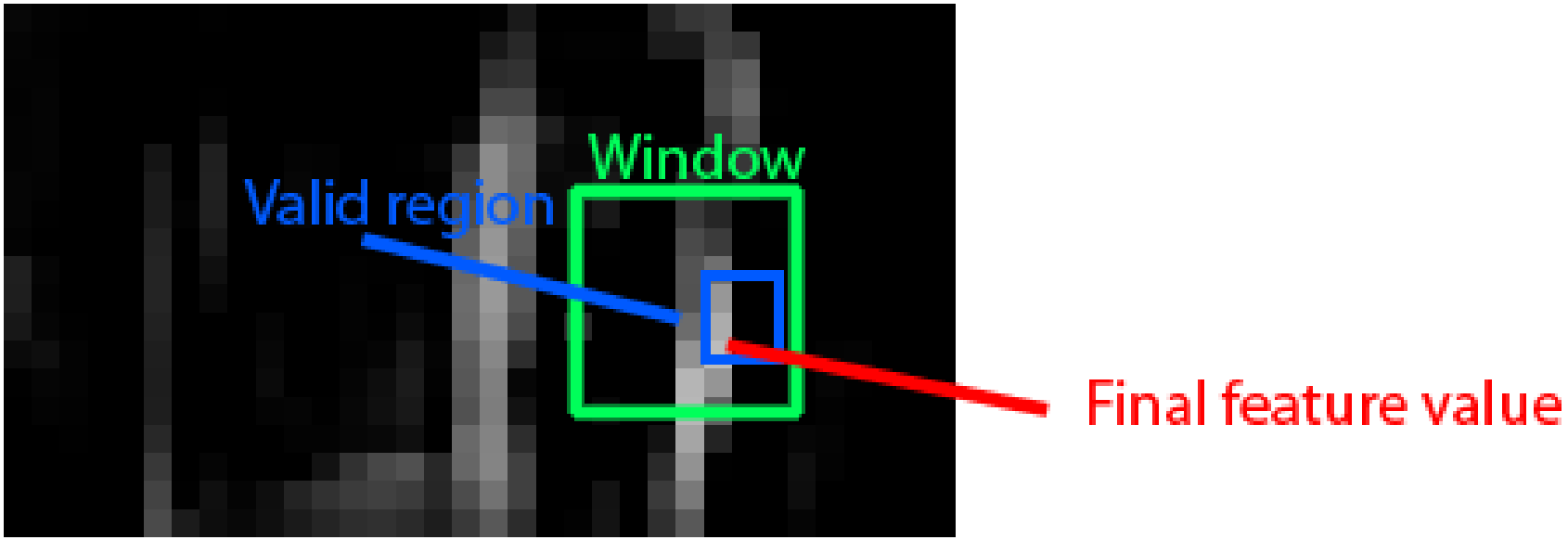
Image pyramiding

- Your classifier should be able to determine if a 32x32 image is a mug
- You need to find mugs of size 64x64 or greater
- Search for different sizes of mugs by resizing the input image
- You will need to implement this yourself, in Classifier::run



Feature extraction at test time

- Be sure to calculate each template response image only once per size of each frame
- You will need to move the region that you do max-pooling over as you move the window that is being classified

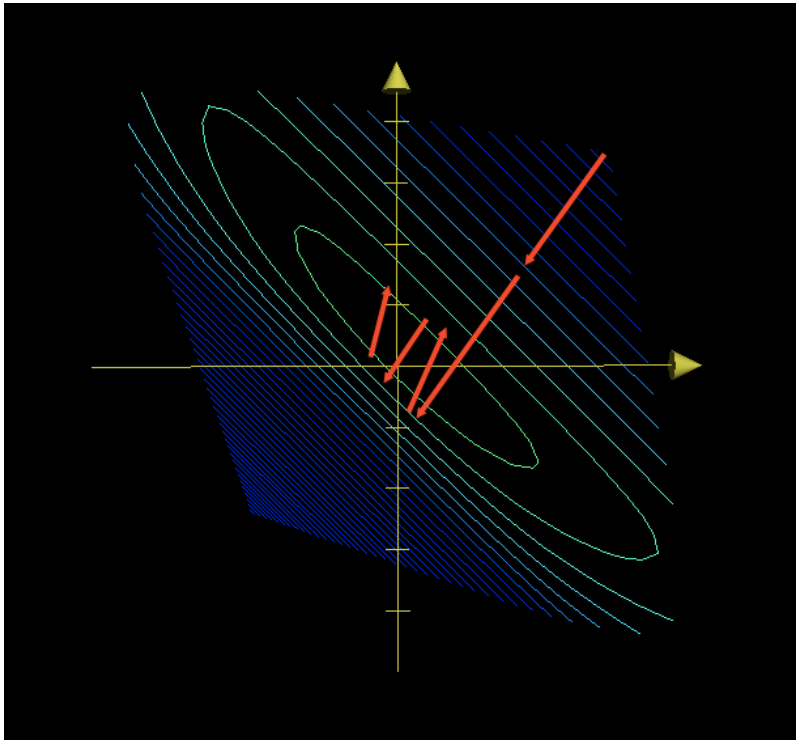


Logistic regression

- Logistic regression is a pattern recognition algorithm that will be covered in lecture on Thursday
- For the milestone, you'll need to get the average squared error of your training labels down to about 9e-8
- Squared error: (true label-predicted probability)^2
- To do this using gradient descent may require some hacks. Keep in mind this is not the most theoretically sound way of optimizing the model, but will be helpful for getting the milestone done easily.

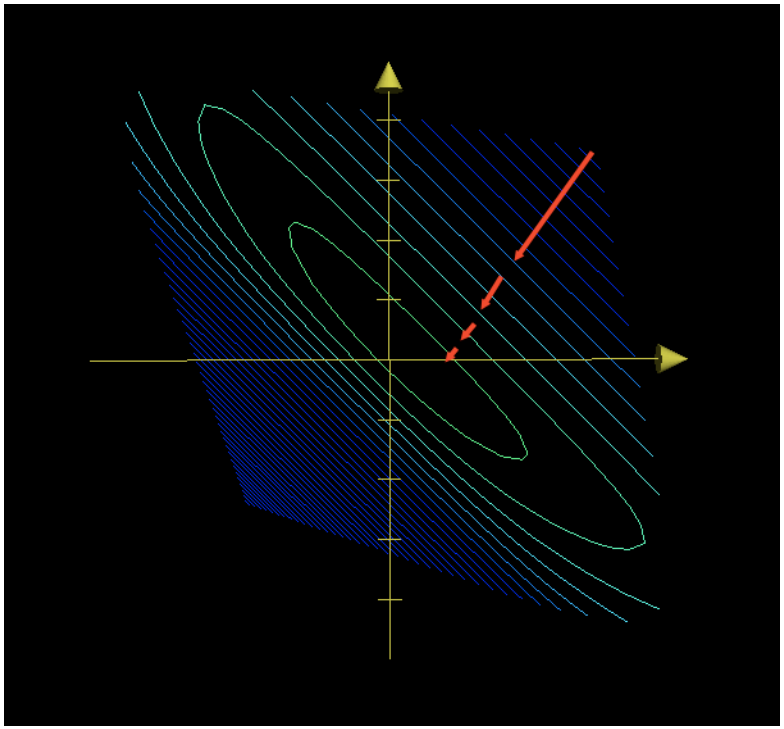
Logistic regression

- Hack #1: If your learning rate is too high, you will overshoot the maximum. If you see the training error increase, you should decrease your learning rate.



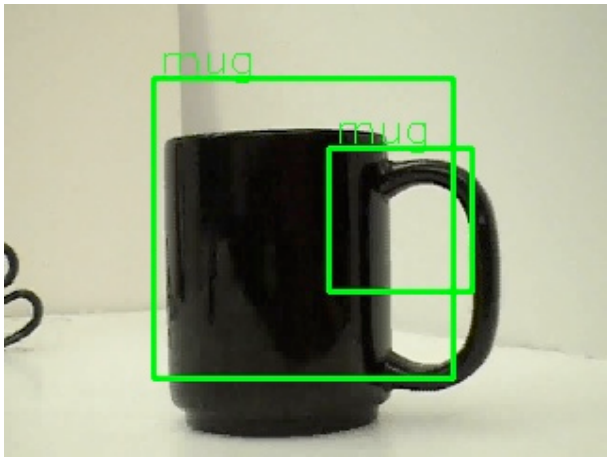
Logistic regression

- Hack #2: If your learning rate is too low, you will not reach the global maximum. You may need to increase your learning rate as you get close to the solution and the gradient flattens out.



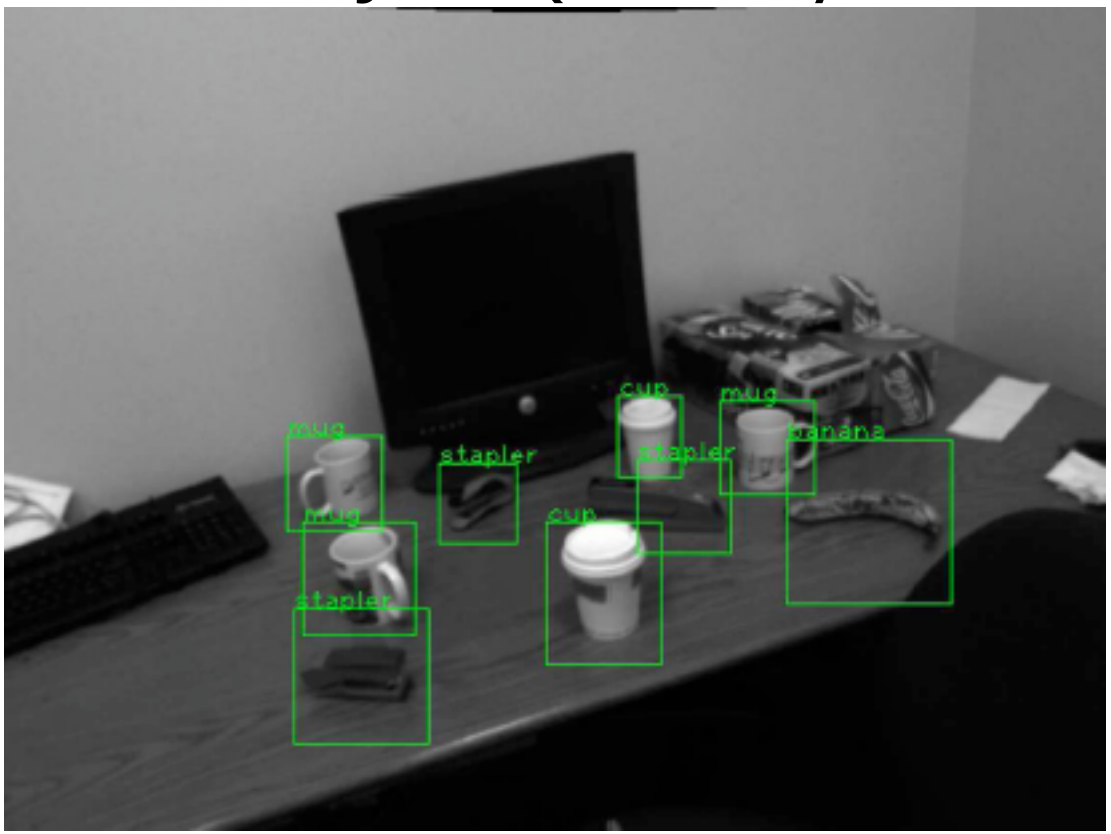
Example Results (Milestone)

- Expect an F-score of about 0.25



Example Results (Final)

- State-of-the-art example with sightly different objects (courtesy Adam Coates)



Object Recognition Tips

- Use non-maxima suppression-- if two detections overlap by more than some amount, discard the one with lower confidence.
- Read some papers for ideas on better features. See the handout for suggestions.
- Try different classifiers and training methods
  - Boosted classifiers, support vector machines
- Filter the output of your classifiers and use motion estimation to predict where an object in frame *n* will be in frame *n*+1
- Anything that works!

Coding Tips

- Modular, general design
- Use OpenCV and other libraries like Eigen
  - make sure you cite external libraries
- Use SVN or CVS
- Your code **must** compile on the myth machines
- Most of the milestone should be implementing logistic regression. My Classifier::run method is about 100 lines long.

Design Process Tips

- Automate testing early
- Consider how you will avoid overfitting
- Make unit tests to check each component individually before combining them into a whole system
- Visualization code usually pays off

- Start early!

What is OpenCV?

- The Open Computer Vision Library is a collection of algorithms and sample code for various computer vision problems:
  - **libcxcore**: core data structures and linear algebra library
  - **libcv**: computer vision library
  - **libhighgui**: media and graphics i/o handling
  - **libml**: machine learning functionality
- Developed originally at Intel, currently maintained by Willow Garage.
- Free access from *.stanford.edu machines to *Learning OpenCV* by Gary Bradski:
  - http://proquest.safaribooksonline.com/9780596516130?tocview=true
- Wiki has lots of information and API documentation
  - http://opencv.willowgarage.com/wiki/

Installing OpenCV (Linux)

- Class copy is available on all leland machines
- To set up on your own machine use,

```
svn co https://opencvlibrary.svn.sourceforge.net/svnroot/opencvlibrary/
opencv

cd opnecv/opencv

mkdir build

cd build

cmake -D CMAKE_BUILD_TYPE=DEBUG ..

make

sudo make install
```

Example 1: Loading and Displaying Images

```
#include "cv.h"
#include "cxcore.h"
#include "highgui.h"

#define WINDOW_NAME "MyWindow"

int main(int argc, char *argv[])
{
    IplImage *image;

    cvNamedWindow(WINDOW_NAME, CV_WINDOW_AUTOSIZE);
    for (int i = 1; i <= argc; i++) {
        image = cvLoadImage(argv[i], 0);   // load from file
        cvShowImage(WINDOW_NAME, image);   // display on screen
        cvWaitKey(0);                      // wait for key press
        cvReleaseImage(&image);            // free memory
    }
    cvDestroyWindow(WINDOW_NAME);

    return 0;
}
```

Example 2: Converting from Color to Grayscale

```
IplImage *image;

// acquire RGB color image somehow
...

// allocate memory for grayscale image
IplImage *gray = cvCreateImage(
    cvGetSize(image),       // same size as original image
    IPL_DEPTH_8U,           // data type (8-bit unsigned)
    1);                     // grayscale has one channel

// color convert the image (source, destination)
cvCvtColor(image, gray, CV_BGR2GRAY);

// do something with greyscale image
...

// free memory used by images
cvReleaseImage(&gray);
cvReleaseImage(&image);
```

Example 3: Resizing an Image

```
IplImage *image;

// acquire image somehow
...

// allocate memory for resized image
IplImage *resizedImage = cvCreateImage(
    cvSize(32, 32),         // new size (width, height)
    image->depth,           // data type (e.g. 8-bit unsigned)
    image->nChannels);      // number of planes (e.g. RGB)

// resize the image (source, destination)
cvResize(image, resizedImage);

// do something with resized image
...

// free memory used by images
cvReleaseImage(&resizedImage);
cvReleaseImage(&image);
```

Example 4: Clipping a Small Region out of an Image

```
IplImage *image;

...// acquire image somehow

// clip out 64-by-64 image patch at (4,8)
CvRect region = cvRect(4, 8, 64, 64);
IplImage *clippedImage = cvCreateImage(
    cvSize(region.width, region.height),
    image->depth, image->nChannels);
cvSetImageROI(image, region);
cvCopyImage(image, clippedImage);
cvResetImageROI(image);

...// do something with clipped region

// and always free memory
cvReleaseImage(&clippedRegion);
cvReleaseImage(&image);
```

Example 5: Optical Flow

```
IplImage *bwImg1, *bwImg2;

...// acquire images somehow

// allocate memory for optical flow vectors
CvMat *dx = cvCreateMat(bwImg1.height, bwImg1.width, CV_32FC1);
CvMat *dy = cvCreateMat(bwImg1.height, bwImg1.width, CV_32FC1);

// compute dense Lucus-Kanade optical flow (previous, current)
// (also see cvCalcOpticalFlowPyrLK for sparse optical flow)
cvCalcOpticalFlowLK(bwImg1, bwImg2, cvSize(5, 5), dx, dy);

double deltaX = cvAvg(dx).val[0]; // bulk x-direction motion
double deltaY = cvAvg(dy).val[0]; // bulk y-direction motion

// free memory
cvReleaseMat(&dy);
cvReleaseMat(&dx);
cvReleaseImage(&bwImg2);
cvReleaseImage(&bwImg1);
```

OpenCV Tips

- Don't forget to free allocated memory
  - cvReleaseImage, cvReleaseMat
- Try to allocate and free memory outside of loops if possible
- Image size is width-by-height; matrix size is rows-by-columns
- Never set the region of interest (ROI) outside of the image/matrix
  - $x \geq 0$, $y \geq 0$, $x + width < image \rightarrow width$, $y + height < image \rightarrow height$
- Never operate on images of different types (image→depth)
  - Make sure you convert first (e.g. RGB to grayscale)
- Check the return values (for NULL)
  - e.g. loading images and allocating images

Contacting the TAs

- Ian Goodfellow is the TA focusing on this project.
- Office hours Wednesday 7-8 PM
- Olga Russakovsky also studies object detection and will be very helpful for this project
- Office hours Friday 2:30-3:30 PM

Finally…

- Good luck!
- (and have fun)