caffe (/github/BVLC/caffe/tree/master) / examples (/github/BVLC/caffe/tree/master/examples)

R-CNN (https://github.com/rbgirshick/rcnn) is a state-of-the-art detector that classifies region proposals by a finetuned Caffe model. For the full details of the R-CNN system and model, refer to its project site and the paper:

> *Rich feature hierarchies for accurate object detection and semantic segmentation*. Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik. CVPR 2014. Arxiv 2013 (http://arxiv.org/abs/1311.2524).

In this example, we do detection by a pure Caffe edition of the R-CNN model for ImageNet. The R-CNN detector outputs class scores for the 200 detection classes of ILSVRC13. Keep in mind that these are raw one vs. all SVM scores, so they are not probabilistically calibrated or exactly comparable across classes. Note that this off-the-shelf model is simply for convenience, and is not the full R-CNN model.

Let's run detection on an image of a bicyclist riding a fish bike in the desert (from the ImageNet challenge—no joke).

First, we'll need region proposals and the Caffe R-CNN ImageNet model:

- Selective Search (http://koen.me/research/selectivesearch/) is the region proposer used by R-CNN. The selective_search_ijcv_with_python (https://github.com/sergeyk/selective_search_ijcv_with_python) Python module takes care of extracting proposals through the selective search MATLAB implementation. To install it, download the module and name its directory `selective_search_ijcv_with_python`, run the demo in MATLAB to compile the necessary functions, then add it to your `PYTHONPATH` for importing. (If you have your own region proposals prepared, or would rather not bother with this step, detect.py (https://github.com/BVLC/caffe/blob/master/python/detect.py) accepts a list of images and bounding boxes as CSV.)

-Run `./scripts/download_model_binary.py models/bvlc_reference_rcnn_ilsvrc13` to get the Caffe R-CNN ImageNet model.

With that done, we'll call the bundled `detect.py` to generate the region proposals and run the network. For an explanation of the arguments, do `./detect.py --help`.

In [1]:

```
!mkdir -p _temp
!echo `pwd`/images/fish-bike.jpg > _temp/det_input.txt
!../python/detect.py --crop_mode=selective_search --pretrained_model=../models/bvlc_refe
rence_rcnn_ilsvrc13/bvlc_reference_rcnn_ilsvrc13.caffemodel --model_def=../models/bvlc_r
eference_rcnn_ilsvrc13/deploy.prototxt --gpu --raw_scale=255 _temp/det_input.txt _temp/d
et_output.h5
```

```
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0218 20:43:25.383932 2099749632 net.cpp:42] Initializing net from paramete
rs:
name: "R-CNN-ilsvrc13"
input: "data"
input_dim: 10
input_dim: 3
input_dim: 227
input_dim: 227
state {
  phase: TEST
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 96
    kernel_size: 11
    stride: 4
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "conv1"
  top: "conv1"
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "norm1"
  type: "LRN"
  bottom: "pool1"
  top: "norm1"
  lrn_param {
    local_size: 5
    alpha: 0.0001
```

```
    beta: 0.75
  }
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "norm1"
  top: "conv2"
  convolution_param {
    num_output: 256
    pad: 2
    kernel_size: 5
    group: 2
  }
}
layer {
  name: "relu2"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "norm2"
  type: "LRN"
  bottom: "pool2"
  top: "norm2"
  lrn_param {
    local_size: 5
    alpha: 0.0001
    beta: 0.75
  }
}
layer {
  name: "conv3"
  type: "Convolution"
  bottom: "norm2"
  top: "conv3"
  convolution_param {
    num_output: 384
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu3"
  type: "ReLU"
```

```
    bottom: "conv3"
    top: "conv3"
  }
  layer {
    name: "conv4"
    type: "Convolution"
    bottom: "conv3"
    top: "conv4"
    convolution_param {
      num_output: 384
      pad: 1
      kernel_size: 3
      group: 2
    }
  }
  layer {
    name: "relu4"
    type: "ReLU"
    bottom: "conv4"
    top: "conv4"
  }
  layer {
    name: "conv5"
    type: "Convolution"
    bottom: "conv4"
    top: "conv5"
    convolution_param {
      num_output: 256
      pad: 1
      kernel_size: 3
      group: 2
    }
  }
  layer {
    name: "relu5"
    type: "ReLU"
    bottom: "conv5"
    top: "conv5"
  }
  layer {
    name: "pool5"
    type: "Pooling"
    bottom: "conv5"
    top: "pool5"
    pooling_param {
      pool: MAX
      kernel_size: 3
      stride: 2
    }
  }
  layer {
    name: "fc6"
    type: "InnerProduct"
    bottom: "pool5"
    top: "fc6"
    inner_product_param {
      num_output: 4096
```

```
    }
  }
  layer {
    name: "relu6"
    type: "ReLU"
    bottom: "fc6"
    top: "fc6"
  }
  layer {
    name: "drop6"
    type: "Dropout"
    bottom: "fc6"
    top: "fc6"
    dropout_param {
      dropout_ratio: 0.5
    }
  }
  layer {
    name: "fc7"
    type: "InnerProduct"
    bottom: "fc6"
    top: "fc7"
    inner_product_param {
      num_output: 4096
    }
  }
  layer {
    name: "relu7"
    type: "ReLU"
    bottom: "fc7"
    top: "fc7"
  }
  layer {
    name: "drop7"
    type: "Dropout"
    bottom: "fc7"
    top: "fc7"
    dropout_param {
      dropout_ratio: 0.5
    }
  }
  layer {
    name: "fc-rcnn"
    type: "InnerProduct"
    bottom: "fc7"
    top: "fc-rcnn"
    inner_product_param {
      num_output: 200
    }
  }
  I0218 20:43:25.385720 2099749632 net.cpp:336] Input 0 -> data

  I0218 20:43:25.385769 2099749632 layer_factory.hpp:74] Creating layer conv1
  I0218 20:43:25.385783 2099749632 net.cpp:76] Creating Layer conv1
  I0218 20:43:25.385790 2099749632 net.cpp:372] conv1 <- data
  I0218 20:43:25.385802 2099749632 net.cpp:334] conv1 -> conv1
  I0218 20:43:25.385815 2099749632 net.cpp:105] Setting up conv1
  I0218 20:43:25.386574 2099749632 net.cpp:112] Top shape: 10 96 55 55 (29040
```

```
00)
I0218 20:43:25.386610 2099749632 layer_factory.hpp:74] Creating layer relu1
I0218 20:43:25.386625 2099749632 net.cpp:76] Creating Layer relu1
I0218 20:43:25.386631 2099749632 net.cpp:372] relu1 <- conv1
I0218 20:43:25.386641 2099749632 net.cpp:323] relu1 -> conv1 (in-place)
I0218 20:43:25.386649 2099749632 net.cpp:105] Setting up relu1
I0218 20:43:25.386656 2099749632 net.cpp:112] Top shape: 10 96 55 55 (29040
00)
I0218 20:43:25.386663 2099749632 layer_factory.hpp:74] Creating layer pool1
I0218 20:43:25.386675 2099749632 net.cpp:76] Creating Layer pool1
I0218 20:43:25.386682 2099749632 net.cpp:372] pool1 <- conv1
I0218 20:43:25.386690 2099749632 net.cpp:334] pool1 -> pool1
I0218 20:43:25.386699 2099749632 net.cpp:105] Setting up pool1
I0218 20:43:25.386716 2099749632 net.cpp:112] Top shape: 10 96 27 27 (69984
0)
I0218 20:43:25.386725 2099749632 layer_factory.hpp:74] Creating layer norm1
I0218 20:43:25.386736 2099749632 net.cpp:76] Creating Layer norm1
I0218 20:43:25.386744 2099749632 net.cpp:372] norm1 <- pool1
I0218 20:43:25.386803 2099749632 net.cpp:334] norm1 -> norm1
I0218 20:43:25.386819 2099749632 net.cpp:105] Setting up norm1
I0218 20:43:25.386832 2099749632 net.cpp:112] Top shape: 10 96 27 27 (69984
0)
I0218 20:43:25.386842 2099749632 layer_factory.hpp:74] Creating layer conv2
I0218 20:43:25.386852 2099749632 net.cpp:76] Creating Layer conv2
I0218 20:43:25.386865 2099749632 net.cpp:372] conv2 <- norm1
I0218 20:43:25.386878 2099749632 net.cpp:334] conv2 -> conv2
I0218 20:43:25.386899 2099749632 net.cpp:105] Setting up conv2
I0218 20:43:25.387024 2099749632 net.cpp:112] Top shape: 10 256 27 27 (1866
240)
I0218 20:43:25.387042 2099749632 layer_factory.hpp:74] Creating layer relu2
I0218 20:43:25.387050 2099749632 net.cpp:76] Creating Layer relu2
I0218 20:43:25.387058 2099749632 net.cpp:372] relu2 <- conv2
I0218 20:43:25.387066 2099749632 net.cpp:323] relu2 -> conv2 (in-place)
I0218 20:43:25.387075 2099749632 net.cpp:105] Setting up relu2
I0218 20:43:25.387081 2099749632 net.cpp:112] Top shape: 10 256 27 27 (1866
240)
I0218 20:43:25.387089 2099749632 layer_factory.hpp:74] Creating layer pool2
I0218 20:43:25.387097 2099749632 net.cpp:76] Creating Layer pool2
I0218 20:43:25.387104 2099749632 net.cpp:372] pool2 <- conv2
I0218 20:43:25.387112 2099749632 net.cpp:334] pool2 -> pool2
I0218 20:43:25.387121 2099749632 net.cpp:105] Setting up pool2
I0218 20:43:25.387130 2099749632 net.cpp:112] Top shape: 10 256 13 13 (4326
40)
I0218 20:43:25.387137 2099749632 layer_factory.hpp:74] Creating layer norm2
I0218 20:43:25.387145 2099749632 net.cpp:76] Creating Layer norm2
I0218 20:43:25.387152 2099749632 net.cpp:372] norm2 <- pool2
I0218 20:43:25.387161 2099749632 net.cpp:334] norm2 -> norm2
I0218 20:43:25.387168 2099749632 net.cpp:105] Setting up norm2
I0218 20:43:25.387176 2099749632 net.cpp:112] Top shape: 10 256 13 13 (4326
40)
I0218 20:43:25.387228 2099749632 layer_factory.hpp:74] Creating layer conv3
I0218 20:43:25.387249 2099749632 net.cpp:76] Creating Layer conv3
I0218 20:43:25.387258 2099749632 net.cpp:372] conv3 <- norm2
I0218 20:43:25.387266 2099749632 net.cpp:334] conv3 -> conv3
I0218 20:43:25.387276 2099749632 net.cpp:105] Setting up conv3
I0218 20:43:25.389375 2099749632 net.cpp:112] Top shape: 10 384 13 13 (6489
60)
```

```
I0218 20:43:25.389408 2099749632 layer_factory.hpp:74] Creating layer relu3
I0218 20:43:25.389421 2099749632 net.cpp:76] Creating Layer relu3
I0218 20:43:25.389430 2099749632 net.cpp:372] relu3 <- conv3
I0218 20:43:25.389438 2099749632 net.cpp:323] relu3 -> conv3 (in-place)
I0218 20:43:25.389447 2099749632 net.cpp:105] Setting up relu3
I0218 20:43:25.389456 2099749632 net.cpp:112] Top shape: 10 384 13 13 (6489
60)
I0218 20:43:25.389462 2099749632 layer_factory.hpp:74] Creating layer conv4
I0218 20:43:25.389472 2099749632 net.cpp:76] Creating Layer conv4
I0218 20:43:25.389478 2099749632 net.cpp:372] conv4 <- conv3
I0218 20:43:25.389487 2099749632 net.cpp:334] conv4 -> conv4
I0218 20:43:25.389497 2099749632 net.cpp:105] Setting up conv4
I0218 20:43:25.391810 2099749632 net.cpp:112] Top shape: 10 384 13 13 (6489
60)
I0218 20:43:25.391856 2099749632 layer_factory.hpp:74] Creating layer relu4
I0218 20:43:25.391871 2099749632 net.cpp:76] Creating Layer relu4
I0218 20:43:25.391880 2099749632 net.cpp:372] relu4 <- conv4
I0218 20:43:25.391888 2099749632 net.cpp:323] relu4 -> conv4 (in-place)
I0218 20:43:25.391898 2099749632 net.cpp:105] Setting up relu4
I0218 20:43:25.391906 2099749632 net.cpp:112] Top shape: 10 384 13 13 (6489
60)
I0218 20:43:25.391913 2099749632 layer_factory.hpp:74] Creating layer conv5
I0218 20:43:25.391923 2099749632 net.cpp:76] Creating Layer conv5
I0218 20:43:25.391929 2099749632 net.cpp:372] conv5 <- conv4
I0218 20:43:25.391937 2099749632 net.cpp:334] conv5 -> conv5
I0218 20:43:25.391947 2099749632 net.cpp:105] Setting up conv5
I0218 20:43:25.393072 2099749632 net.cpp:112] Top shape: 10 256 13 13 (4326
40)
I0218 20:43:25.393108 2099749632 layer_factory.hpp:74] Creating layer relu5
I0218 20:43:25.393122 2099749632 net.cpp:76] Creating Layer relu5
I0218 20:43:25.393129 2099749632 net.cpp:372] relu5 <- conv5
I0218 20:43:25.393138 2099749632 net.cpp:323] relu5 -> conv5 (in-place)
I0218 20:43:25.393148 2099749632 net.cpp:105] Setting up relu5
I0218 20:43:25.393157 2099749632 net.cpp:112] Top shape: 10 256 13 13 (4326
40)
I0218 20:43:25.393167 2099749632 layer_factory.hpp:74] Creating layer pool5
I0218 20:43:25.393175 2099749632 net.cpp:76] Creating Layer pool5
I0218 20:43:25.393182 2099749632 net.cpp:372] pool5 <- conv5
I0218 20:43:25.393190 2099749632 net.cpp:334] pool5 -> pool5
I0218 20:43:25.393199 2099749632 net.cpp:105] Setting up pool5
I0218 20:43:25.393209 2099749632 net.cpp:112] Top shape: 10 256 6 6 (92160)
I0218 20:43:25.393218 2099749632 layer_factory.hpp:74] Creating layer fc6
I0218 20:43:25.393226 2099749632 net.cpp:76] Creating Layer fc6
I0218 20:43:25.393232 2099749632 net.cpp:372] fc6 <- pool5
I0218 20:43:25.393240 2099749632 net.cpp:334] fc6 -> fc6
I0218 20:43:25.393249 2099749632 net.cpp:105] Setting up fc6
I0218 20:43:25.516396 2099749632 net.cpp:112] Top shape: 10 4096 1 1 (40960
)
I0218 20:43:25.516445 2099749632 layer_factory.hpp:74] Creating layer relu6
I0218 20:43:25.516463 2099749632 net.cpp:76] Creating Layer relu6
I0218 20:43:25.516470 2099749632 net.cpp:372] relu6 <- fc6

I0218 20:43:25.516480 2099749632 net.cpp:323] relu6 -> fc6 (in-place)
I0218 20:43:25.516490 2099749632 net.cpp:105] Setting up relu6
I0218 20:43:25.516497 2099749632 net.cpp:112] Top shape: 10 4096 1 1 (40960
)
I0218 20:43:25.516505 2099749632 layer_factory.hpp:74] Creating layer drop6
I0218 20:43:25.516515 2099749632 net.cpp:76] Creating Layer drop6
```

```
I0218 20:43:25.516521 2099749632 net.cpp:372] drop6 <- fc6
I0218 20:43:25.516530 2099749632 net.cpp:323] drop6 -> fc6 (in-place)
I0218 20:43:25.516538 2099749632 net.cpp:105] Setting up drop6
I0218 20:43:25.516557 2099749632 net.cpp:112] Top shape: 10 4096 1 1 (40960
)
I0218 20:43:25.516566 2099749632 layer_factory.hpp:74] Creating layer fc7
I0218 20:43:25.516576 2099749632 net.cpp:76] Creating Layer fc7
I0218 20:43:25.516582 2099749632 net.cpp:372] fc7 <- fc6
I0218 20:43:25.516589 2099749632 net.cpp:334] fc7 -> fc7
I0218 20:43:25.516599 2099749632 net.cpp:105] Setting up fc7
I0218 20:43:25.604786 2099749632 net.cpp:112] Top shape: 10 4096 1 1 (40960
)
I0218 20:43:25.604838 2099749632 layer_factory.hpp:74] Creating layer relu7
I0218 20:43:25.604852 2099749632 net.cpp:76] Creating Layer relu7
I0218 20:43:25.604859 2099749632 net.cpp:372] relu7 <- fc7
I0218 20:43:25.604868 2099749632 net.cpp:323] relu7 -> fc7 (in-place)
I0218 20:43:25.604878 2099749632 net.cpp:105] Setting up relu7
I0218 20:43:25.604885 2099749632 net.cpp:112] Top shape: 10 4096 1 1 (40960
)
I0218 20:43:25.604893 2099749632 layer_factory.hpp:74] Creating layer drop7
I0218 20:43:25.604902 2099749632 net.cpp:76] Creating Layer drop7
I0218 20:43:25.604908 2099749632 net.cpp:372] drop7 <- fc7
I0218 20:43:25.604917 2099749632 net.cpp:323] drop7 -> fc7 (in-place)
I0218 20:43:25.604924 2099749632 net.cpp:105] Setting up drop7
I0218 20:43:25.604933 2099749632 net.cpp:112] Top shape: 10 4096 1 1 (40960
)
I0218 20:43:25.604939 2099749632 layer_factory.hpp:74] Creating layer fc-rc
nn
I0218 20:43:25.604948 2099749632 net.cpp:76] Creating Layer fc-rcnn
I0218 20:43:25.604954 2099749632 net.cpp:372] fc-rcnn <- fc7
I0218 20:43:25.604962 2099749632 net.cpp:334] fc-rcnn -> fc-rcnn
I0218 20:43:25.604971 2099749632 net.cpp:105] Setting up fc-rcnn
I0218 20:43:25.606878 2099749632 net.cpp:112] Top shape: 10 200 1 1 (2000)
I0218 20:43:25.606904 2099749632 net.cpp:165] fc-rcnn does not need backwar
d computation.
I0218 20:43:25.606909 2099749632 net.cpp:165] drop7 does not need backward
computation.
I0218 20:43:25.606916 2099749632 net.cpp:165] relu7 does not need backward
computation.
I0218 20:43:25.606922 2099749632 net.cpp:165] fc7 does not need backward co
mputation.
I0218 20:43:25.606928 2099749632 net.cpp:165] drop6 does not need backward
computation.
I0218 20:43:25.606935 2099749632 net.cpp:165] relu6 does not need backward
computation.
I0218 20:43:25.606940 2099749632 net.cpp:165] fc6 does not need backward co
mputation.
I0218 20:43:25.606946 2099749632 net.cpp:165] pool5 does not need backward
computation.
I0218 20:43:25.606952 2099749632 net.cpp:165] relu5 does not need backward
computation.
I0218 20:43:25.606958 2099749632 net.cpp:165] conv5 does not need backward
computation.
I0218 20:43:25.606964 2099749632 net.cpp:165] relu4 does not need backward
computation.
I0218 20:43:25.606971 2099749632 net.cpp:165] conv4 does not need backward
computation.
```

```
I0218 20:43:25.606976 2099749632 net.cpp:165] relu3 does not need backward
computation.
I0218 20:43:25.606982 2099749632 net.cpp:165] conv3 does not need backward
computation.
I0218 20:43:25.606988 2099749632 net.cpp:165] norm2 does not need backward
computation.
I0218 20:43:25.606995 2099749632 net.cpp:165] pool2 does not need backward
computation.
I0218 20:43:25.607002 2099749632 net.cpp:165] relu2 does not need backward
computation.
I0218 20:43:25.607007 2099749632 net.cpp:165] conv2 does not need backward
computation.
I0218 20:43:25.607013 2099749632 net.cpp:165] norm1 does not need backward
computation.
I0218 20:43:25.607199 2099749632 net.cpp:165] pool1 does not need backward
computation.
I0218 20:43:25.607213 2099749632 net.cpp:165] relu1 does not need backward
computation.
I0218 20:43:25.607219 2099749632 net.cpp:165] conv1 does not need backward
computation.
I0218 20:43:25.607225 2099749632 net.cpp:201] This network produces output
fc-rcnn
I0218 20:43:25.607239 2099749632 net.cpp:446] Collecting Learning Rate and
Weight Decay.
I0218 20:43:25.607255 2099749632 net.cpp:213] Network initialization done.
I0218 20:43:25.607262 2099749632 net.cpp:214] Memory required for data: 624
25920
E0218 20:43:26.388214 2099749632 upgrade_proto.cpp:618] Attempting to upgra
de input file specified using deprecated V1LayerParameter: ../models/bvlc_r
eference_rcnn_ilsvrc13/bvlc_reference_rcnn_ilsvrc13.caffemodel
I0218 20:43:27.089423 2099749632 upgrade_proto.cpp:626] Successfully upgrad
ed file specified using deprecated V1LayerParameter
GPU mode
Loading input...
selective_search_rcnn({'/Users/shelhamer/h/desk/caffe/caffe-dev/examples/im
ages/fish-bike.jpg'}, '/var/folders/bk/dtkn5qjd11bd17b2j36zplyw0000gp/T/tmp
akaRLL.mat')
Processed 1570 windows in 102.895 s.
/Users/shelhamer/anaconda/lib/python2.7/site-packages/pandas/io/pytables.py
:2453: PerformanceWarning:
your performance may suffer as PyTables will pickle object types that it ca
nnot
map directly to c-types [inferred_type->mixed,key->block1_values] [items->[
'prediction']]

  warnings.warn(ws, PerformanceWarning)
Saved to _temp/det_output.h5 in 0.298 s.
```

This run was in GPU mode. For CPU mode detection, call `detect.py` without the `--gpu` argument.

Running this outputs a DataFrame with the filenames, selected windows, and their detection scores to an HDF5 file. (We only ran on one image, so the filenames will all be the same.)

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

df = pd.read_hdf('_temp/det_output.h5', 'df')
print(df.shape)
print(df.iloc[0])
```

```
(1570, 5)
prediction    [-2.62247, -2.84579, -2.85122, -3.20838, -1.94...
ymin                                                     79.846
xmin                                                       9.62
ymax                                                     246.31
xmax                                                    339.624
Name: /Users/shelhamer/h/desk/caffe/caffe-dev/examples/images/fish-bike.jpg
, dtype: object
```

1570 regions were proposed with the R-CNN configuration of selective search. The number of proposals will vary from image to image based on its contents and size -- selective search isn't scale invariant.

In general, `detect.py` is most efficient when running on a lot of images: it first extracts window proposals for all of them, batches the windows for efficient GPU processing, and then outputs the results. Simply list an image per line in the `images_file`, and it will process all of them.

Although this guide gives an example of R-CNN ImageNet detection, `detect.py` is clever enough to adapt to different Caffe models' input dimensions, batch size, and output categories. You can switch the model definition and pretrained model as desired. Refer to `python detect.py --help` for the parameters to describe your data set. There's no need for hardcoding.

Anyway, let's now load the ILSVRC13 detection class names and make a DataFrame of the predictions. Note you'll need the auxiliary ilsvrc2012 data fetched by `data/ilsvrc12/get_ilsvrc12_aux.sh`.

In [3]:

```
with open('../data/ilsvrc12/det_synset_words.txt') as f:
    labels_df = pd.DataFrame([
        {
            'synset_id': l.strip().split(' ')[0],
            'name': ' '.join(l.strip().split(' ')[1:]).split(',')[0]
        }
        for l in f.readlines()
    ])
labels_df.sort('synset_id')
predictions_df = pd.DataFrame(np.vstack(df.prediction.values), columns=labels_df['name']
)
print(predictions_df.iloc[0])
```

```
name
accordion      -2.622471
airplane       -2.845788
ant            -2.851219
antelope       -3.208377
apple          -1.949950
armadillo      -2.472935
artichoke      -2.201684
axe            -2.327404
baby bed       -2.737925
backpack       -2.176763
bagel          -2.681061
balance beam   -2.722538
banana         -2.390628
band aid       -1.598909
banjo          -2.298197
...
trombone        -2.582361
trumpet         -2.352853
turtle          -2.360859
tv or monitor   -2.761043
unicycle        -2.218467
vacuum          -1.907717
violin          -2.757079
volleyball      -2.723689
waffle iron     -2.418540
washer          -2.408994
water bottle    -2.174899
watercraft      -2.837425
whale           -3.120338
wine bottle     -2.772960
zebra           -2.742913
Name: 0, Length: 200, dtype: float32
```

Let's look at the activations.

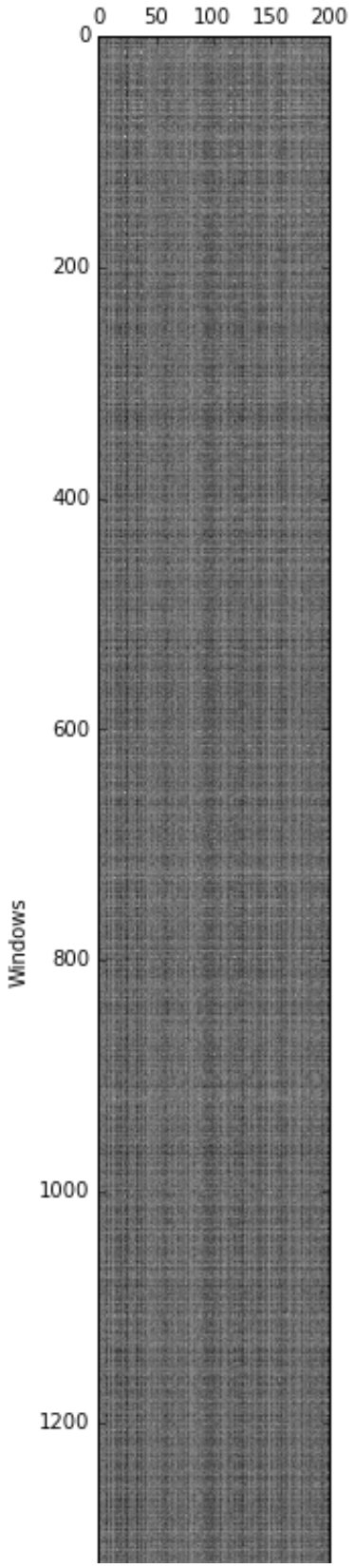
In [4]:

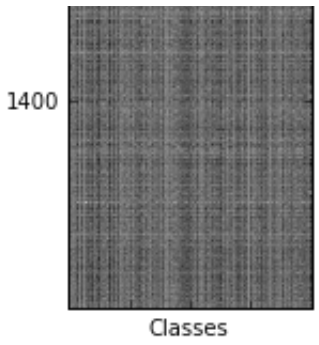
```
plt.gray()
plt.matshow(predictions_df.values)
plt.xlabel('Classes')
plt.ylabel('Windows')
```

Out[4]:

<matplotlib.text.Text at 0x114f15f90>

<matplotlib.figure.Figure at 0x114254b50>





Now let's take max across all windows and plot the top classes.

In [5]:

```
max_s = predictions_df.max(0)
max_s.sort(ascending=False)
print(max_s[:10])
```

```
name
person          1.835771
bicycle         0.866110
unicycle        0.057080
motorcycle     -0.006122
banjo          -0.028209
turtle         -0.189831
electric fan   -0.206788
cart           -0.214235
lizard         -0.393519
helmet         -0.477942
dtype: float32
```

The top detections are in fact a person and bicycle. Picking good localizations is a work in progress; we pick the top-scoring person and bicycle detections.

In [6]:

```
# Find, print, and display the top detections: person and bicycle.
i = predictions_df['person'].argmax()
j = predictions_df['bicycle'].argmax()

# Show top predictions for top detection.
f = pd.Series(df['prediction'].iloc[i], index=labels_df['name'])
print('Top detection:')
print(f.order(ascending=False)[:5])
print('')

# Show top predictions for second-best detection.
f = pd.Series(df['prediction'].iloc[j], index=labels_df['name'])
print('Second-best detection:')
print(f.order(ascending=False)[:5])

# Show top detection in red, second-best top detection in blue.
im = plt.imread('images/fish-bike.jpg')
plt.imshow(im)
currentAxis = plt.gca()

det = df.iloc[i]
coords = (det['xmin'], det['ymin']), det['xmax'] - det['xmin'], det['ymax'] - det['ymin']
currentAxis.add_patch(plt.Rectangle(*coords, fill=False, edgecolor='r', linewidth=5))

det = df.iloc[j]
coords = (det['xmin'], det['ymin']), det['xmax'] - det['xmin'], det['ymax'] - det['ymin']
currentAxis.add_patch(plt.Rectangle(*coords, fill=False, edgecolor='b', linewidth=5))
```

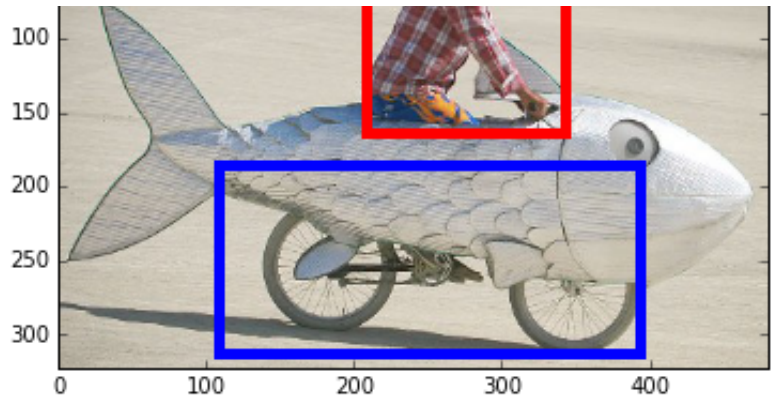
```
Top detection:
name
person             1.835771
swimming trunks   -1.150371
rubber eraser     -1.231106
turtle            -1.266037
plastic bag       -1.303265
dtype: float32

Second-best detection:

name
bicycle     0.866110
unicycle   -0.359139
scorpion   -0.811621
lobster    -0.982891
lamp       -1.096808
dtype: float32
```

Out[6]:

```
<matplotlib.patches.Rectangle at 0x118576a90>
```



That's cool. Let's take all 'bicycle' detections and NMS them to get rid of overlapping windows.

In [7]:

```
def nms_detections(dets, overlap=0.3):
    """
    Non-maximum suppression: Greedily select high-scoring detections and
    skip detections that are significantly covered by a previously
    selected detection.

    This version is translated from Matlab code by Tomasz Malisiewicz,
    who sped up Pedro Felzenszwalb's code.

    Parameters
    ----------
    dets: ndarray
        each row is ['xmin', 'ymin', 'xmax', 'ymax', 'score']
    overlap: float
        minimum overlap ratio (0.3 default)

    Output
    ------
    dets: ndarray
        remaining after suppression.
    """
    x1 = dets[:, 0]
    y1 = dets[:, 1]
    x2 = dets[:, 2]
    y2 = dets[:, 3]
    ind = np.argsort(dets[:, 4])

    w = x2 - x1
    h = y2 - y1
    area = (w * h).astype(float)

    pick = []
    while len(ind) > 0:
        i = ind[-1]
        pick.append(i)
        ind = ind[:-1]

        xx1 = np.maximum(x1[i], x1[ind])
        yy1 = np.maximum(y1[i], y1[ind])
        xx2 = np.minimum(x2[i], x2[ind])
        yy2 = np.minimum(y2[i], y2[ind])

        w = np.maximum(0., xx2 - xx1)
        h = np.maximum(0., yy2 - yy1)

        wh = w * h
        o = wh / (area[i] + area[ind] - wh)

        ind = ind[np.nonzero(o <= overlap)[0]]

    return dets[pick, :]
```

In [8]:

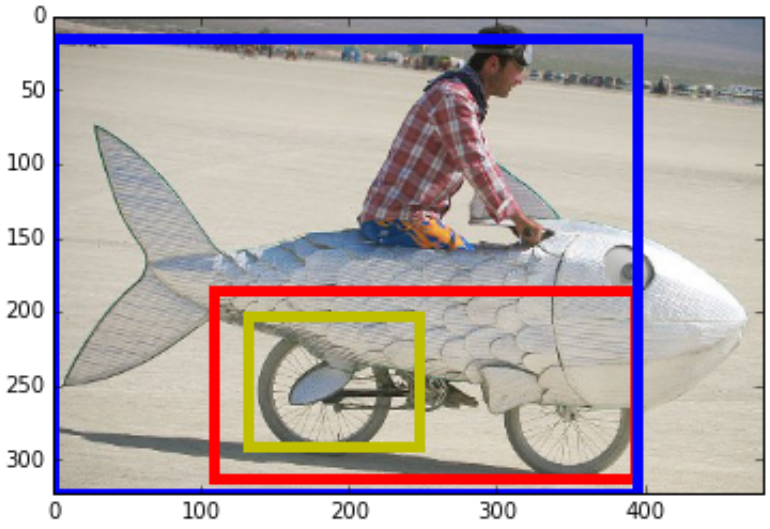
```
scores = predictions_df['bicycle']
windows = df[['xmin', 'ymin', 'xmax', 'ymax']].values
dets = np.hstack((windows, scores[:, np.newaxis]))
nms_dets = nms_detections(dets)
```

Show top 3 NMS'd detections for 'bicycle' in the image and note the gap between the top scoring box (red) and the remaining boxes.

In [9]:

```
plt.imshow(im)
currentAxis = plt.gca()
colors = ['r', 'b', 'y']
for c, det in zip(colors, nms_dets[:3]):
    currentAxis.add_patch(
        plt.Rectangle((det[0], det[1]), det[2]-det[0], det[3]-det[1],
        fill=False, edgecolor=c, linewidth=5)
    )
print 'scores:', nms_dets[:3, 4]
```

```
scores: [ 0.86610985 -0.70051557 -1.34796357]
```



This was an easy instance for bicycle as it was in the class's training set. However, the person result is a true detection since this was not in the set for that class.

You should try out detection on an image of your own next!

(Remove the temp directory to clean up, and we're done.)

In [10]:

```
!rm -rf _temp
```