Homework #3

Task: Use Python + a caffe-trained CNN to ***localize*** class instances.
Description: In the last homework, you were introduced to CNN's. In this homework, you will use CNN's again, but the main purpose will be to show you how some pre- and post-processing can perform more meaningful tasks.
Due Date: 5 p.m. EST, Friday, Dec 1, 2017
Programming Language: Python + Caffe
Dataset: ImageNet subset (4 classes)

Guide to dataset

ImageNet has ~14M images and 1000 classes. For this assignment, we will be using ~1700 images from four classes and creating a fifth "background" class:

Class Name	Class Label	Identifier	Number of Images
Fox	0	n02119789	324
Bike	1	n03792782	415
Elephant	2	n02504458	518
Car	3	n04037443	427
Background	4	NA	Details later

A few notes:

1. Class labels are 0-indexed in caffe
2. Images come in all resolutions. (Lucky for you, caffe warps them to the input size for free.)
3. Images often contain the class in their "natural habitat" but some are not.
4. To get a feel for the dataset, navigate to the folders and take a peek.

Instructions:

A) Complete Phase 1: Create a dataset for localization training
B) Complete Phase 2: Balance dataset and prepare train/test splits
C) Complete Phase 3: Train a CNN in caffe
D) Complete Phase 4: Test your CNN
E) Submit files to Blackboard for grading.

Details on all of each of these steps are provided hereafter.

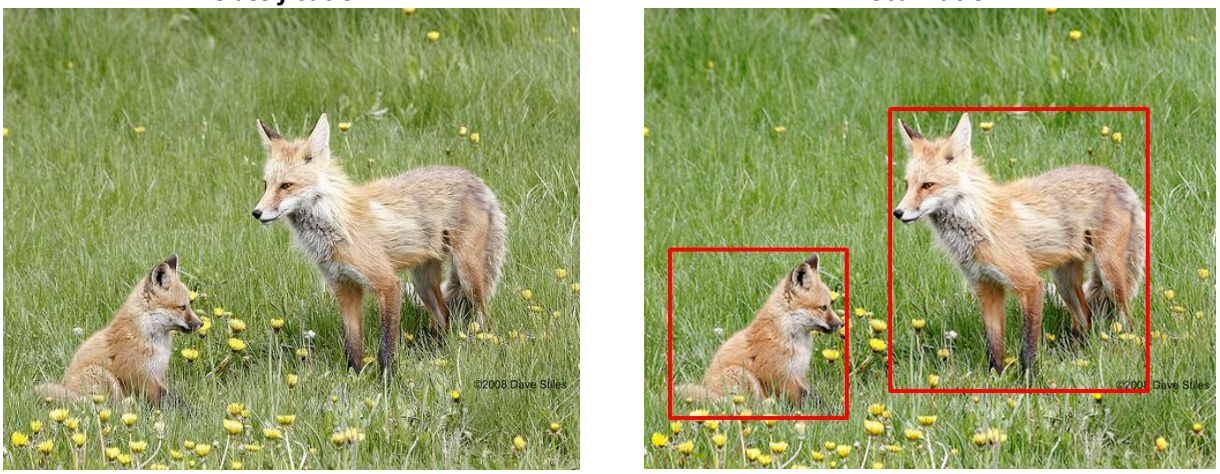
Grading Criteria:

30 points Your hw3-deploy.prototxt and hw3-weights.caffemodel load without error
30 points Your model performs a forward pass and provides class predictions
40 points Your model predicts the correct proposal 20% of the time.

Classification vs. Localization

"Classification" refers to classifying the entire image as a single class. "Localization" refers to finding instances of the class in the image. Figure 1 illustrates the difference between the two.

"Classification" ***"Localization"***



"fox" "fox"

Figure 1 - Classification vs. Localization

For Classification, the approach is simple: you feed the entire image into your CNN. This is what we did in hw2.

For Localization, the approach is a bit more involved. The CNN training portion is still the same, but we do some additional pre- and post-processing. All of this will be explained in the sections that follow.

IoU Scores

IoU stands for "Intersection Over Union". This blog post does a great job of explaining what it is (and I highly recommend that you spend a few minutes looking at it as it does a better job of explaining than I will do here): http://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

In short, given two bounding boxes (e.g. a proposal and ground truth), we can compute an IoU score. An IoU score of 1 means the bounding boxes have perfect overlap. An IoU score of 0 means the bounding boxes have no overlap. An IoU score between 0 and 1 means the bounding boxes have some level of overlap.

Guide to getting started.

1. Go to a computer lab of your choice (e.g. MSEE189, EE206, EE207).
2. Sit down at a Linux computer and login.
3. Open a terminal.
4. Type "ee570" and press <enter>.

What just happened when I did that?

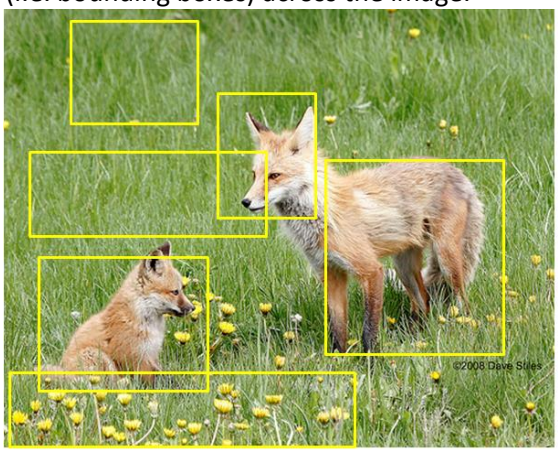
- We loaded an environment for you…and ran a script.
    - The environment has a few python and Matlab libraries that you will need to complete this project as well, some environmental variables, and a compiled program (caffe).
    - The script created a directory for you to work in, copied over some files, and printed some instructions. Details about the files will be provided in the subsequent sections.

Phase 1: Create a dataset for localization training

In order to do localization training, the network needs to learn what each class looks like and how to distinguish that class from the "background".

Here's the general idea:

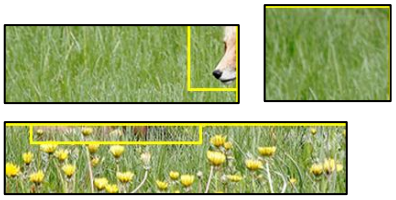
(A) We take an image from our dataset. We use an "object proposal generation algorithm" to produce "proposals" (i.e. bounding boxes) across the image:



(B) We compare each proposal to the ground truth boxes and compute IoU scores.
   a. If the proposal overlaps a ground truth box with an IoU score of 0.7 or higher, it is a "true" proposal.
   b. If the proposal overlaps all ground truth boxes with an IoU score of 0.4 or lower, it is a "false" proposal.
   c. If the IoU score is between 0.4 and 0.7, we discard the sample.

(C) We save off the true proposals and false proposals as new images.
   a. These new images will be what we train the CNN with.

"True Proposal"
IoU > 0.7



"False Proposals" (background)
IoU < 0.4

Discarded propsal
0.4 < IoU < 0.7

A few notes:

- The "object proposal generation algorithm" that you will be using is called "edge_boxes".
  - An example of how to use this library along with some useful notes is provided in the hw3-guide-phase1.py file.
- Bounding boxes are often stored via four numbers: (topLeftX, topLeftY, bottomRightX, bottomRightY). The ground truth bounding box coordinates for each image is stored in an XML file in the "Annotations" folder. The XML file shares the same name as the image.
  - An example of how to load the XML file and extract the coordinates is provided in the hw3-guide-phase1.py file.
- A function that computes IoU scores is provided in the hw3-guide-phase1.py file, along with notes on how to use it.
- A few other useful functions are also explained in the hw3-guide-phase1.py file.
- It's possible that of the N edge_boxes proposals, none of them have an IoU score over 0.7!

    - In that case, we don't have any positive samples from that image. Oh well.
- For most images, there will be hundreds of "false" proposals and only a handful of "true" proposals. You should set a limit on how many proposals you collect from each image.
    - Collect a maximum of 5 positive samples and 5 negative samples from each image. After that, move on to the next image.
- Collect background images from every class. Don't only collect background images from a single class (like "elephant").
    - Why? If you only have backgrounds from a certain class (e.g. "elephant"), the CNN will only learn to recognize backgrounds that contain trees, leaves, dirt, grass. It won't learn how to recognize the backgrounds found in images of "car".
- You should keep the "true" (positive) and "false" (negative) proposals separated. You should keep each class separated as well.
    - Why? Because we will be balancing the number of samples from each class. This will be explained in greater detail in the next section. You should create a directory structure that looks like this:

```
- /home/min/a/jjohanse/ee570/hw3/images/
        + fox
                + pos
                + neg
        - bike
                - pos
                        image_0001.JPEG
                        image_0002.JPEG
                        image_0003.JPEG
                        …
                        image_####.JPEG
                - neg
                        image_0001.JPEG
                        image_0002.JPEG
                        image_0003.JPEG
                        …
                        image_####.JPEG
        + elephant
        + car
```

Pseudocode for this phase might look something like this:

```
// loop over the images in a particular folder (e.g. "n02119789")
        // get edge_boxes proposals for this image
        // loop over the proposals
                // loop over the ground truth bounding boxes for this image (some have 2+)
                        // compute the IoU score
                        // remember the maxIoU score
                // if the maxIoU > 0.7, and I've saved less than 5 samples, then save this image
                // if the maxIoU < 0.4, and I've saved less than 5 samples, then save this image
```

Your console output might look something like this:

```
...
Image 380: /home/min/a/ee570/hw3-files/hw3-dataset/n04037443/n04037443_7551.JPEG
numProposals:  2978
```

```
Proposal 0.  topLeft = (52, 99).  BottomRight = (500, 375).
       IoU Score: 0.837311956775
       You found a TRUE proposal!  (We can use this as a proposal of the true class.)
       Saving this image.
Proposal 2.  topLeft = (49, 187).  BottomRight = (274, 361).
       IoU Score: 0.37056820458
       You found a FALSE proposal! (This is a valid 'background' proposal.)
       Saving this image.
Proposal 3.  topLeft = (55, 97).  BottomRight = (447, 303).
       IoU Score: 0.735335391524
       You found a TRUE proposal!  (We can use this as a proposal of the true class.)
       Saving this image.
Proposal 8.  topLeft = (219, 75).  BottomRight = (466, 256).
       IoU Score: 0.313739813071
       You found a FALSE proposal! (This is a valid 'background' proposal.)
       Saving this image.
Proposal 14.  topLeft = (156, 122).  BottomRight = (352, 264).
       IoU Score: 0.270513448372
       You found a FALSE proposal! (This is a valid 'background' proposal.)
       Saving this image.
Proposal 15.  topLeft = (259, 99).  BottomRight = (449, 256).
       IoU Score: 0.278065351019
       You found a FALSE proposal! (This is a valid 'background' proposal.)
       Saving this image.
Proposal 16.  topLeft = (216, 258).  BottomRight = (274, 338).
       IoU Score: 0.0458905885403
       You found a FALSE proposal! (This is a valid 'background' proposal.)
       Saving this image.
Proposal 20.  topLeft = (51, 140).  BottomRight = (447, 347).
       IoU Score: 0.785806513318
       You found a TRUE proposal!  (We can use this as a proposal of the true class.)
       Saving this image.
Image 381: /home/min/a/ee570/hw3-files/hw3-dataset/n04037443/n04037443_7569.JPEG
numProposals:  480
Proposal 4.  topLeft = (162, 106).  BottomRight = (253, 154).
       IoU Score: 0.171263581795
       You found a FALSE proposal! (This is a valid 'background' proposal.)
       Saving this image.
Proposal 5.  topLeft = (24, 76).  BottomRight = (253, 168).
       IoU Score: 0.81262821974
       You found a TRUE proposal!  (We can use this as a proposal of the true class.)
       Saving this image.
...
```

<u>At the end of this phase:</u>

You should have directory structure that looks the one described above. Each directory should be full of positive and negative samples of each class.

***When done, it would be wise to look at a sampling of your saved images. You should make sure that the "pos" images are indeed representations of that particular class. You should also make sure that the "neg" images are representations of the background of that particular class. ***

Phase 2: Balance dataset and prepare train/test splits

As mentioned in phase 1, the "object proposal generation algorithm" computes many more false samples than true samples. Let's consider what would happen if we kept all of them.

Suppose that from one image we create 100 proposals: 5 of them are "true proposals" and 95 of them are "false proposals" (i.e. "background"). If we began training a network with this ratio of true/false proposals, the network would learn to predict "background" 100% of the time…and it would have an accuracy of 95%!

In order to prevent this from happening, we need to "balance" the ratio of samples from each class. This will help the network find a better solution (than just predicting "background")! That is what we will do next.

The number of samples in each folder may look something like this:

- Fox:
  - Pos: 608
  - Neg: 1620
- Bike:
  - Pos: 667
  - Neg: 2075
- Elephant:
  - Pos: 1017
  - Neg: 2590
- Car:
  - Pos: 692
  - Neg: 2135

We need to have roughly equal numbers from each class (fox, bike, elephant, car, background). Keep in mind that the "background" class is the union of the "neg" folders.

From the numbers above, we might choose to extract 600 samples from each "pos" folder and 150 samples from each "neg" folder so that each class has 600 samples.

Recall that to train/test in caffe, we need a file with paths to the image and a label. Like this:

<absolutePathToImage1> <label>
<absolutePathToImage2> <label>
…
<absolutePathToImageN> <label>

We will first create a "data.txt" file that has the proper balance of samples. It should have 3000 rows (600 samples x 5 classes) and look something like this:

/home/min/a/jjohanse/ee570/hw3/images/fox/pos/image_0001.JPEG 0
/home/ min/a/jjohanse /ee570/hw3/images/fox/pos/image_0003.JPEG 0
/home/ min/a/jjohanse /ee570/hw3/images/fox/pos/image_0008.JPEG 0
…
/home/ min/a/jjohanse /ee570/hw3/images/bike/pos/image_0000.JPEG 1
/home/ min/a/jjohanse /ee570/hw3/images/bike/pos/image_0022.JPEG 1
…
/home/ min/a/jjohanse /ee570/hw3/images/fox/neg/image_0001.JPEG 4

/home/ min/a/jjohanse /ee570/hw3/images/elephant/neg/image_0073.JPEG 4
/home/ min/a/jjohanse /ee570/hw3/images/car/neg/image_0037.JPEG 4
…

Then, we will split "data.txt" into two files:

- hw3-train-split.txt
- hw3-test-split.txt

A file to do this has been provided for you: hw3-split-data.py. Look at the user-specified values, change them if necessary, and run the script. You will get the desired splits.

A few notes:

- hw3-train-split.txt should contain ~90% of the rows from data.txt
- hw3-test-split.txt should contain ~10% of the rows from data.txt
- The hw3-split-data.py folder will ensure that there is no overlap of data between the hw3-test-split.txt and hw3-train-split.txt). It will also ensure ~equal representation from each class. (You don't want only "car" samples in your test set!)

<u>At the end of this phase:</u>
You will have the following files:

- data.txt
- hw3-train-split.txt
- hw3-test-split.txt

Each should have the structure explained above and contain roughly equal numbers of samples from each class.

Phase 3: Train the CNN

Now to training the network! This part is exactly the same as in hw2. In fact, I used the exact same network architecture that I used in hw2 and it provided me with good enough results.
When you're done training, create your hw3-deploy.prototxt file.

Some notes:

- Your NN should take a 32x32x3 image as input.
- You should not perform any additional transformations on the input (e.g. scaling the pixel values from [0 255] to [0 1].
- Your NN should have 5 output nodes. (Remember we added a background class!)
- If you don't want to use your network from hw2, you can use the hw2-solution that was provided. (This phase is meant to be quick and easy. It should only take a small amount of time.)
- If you want to see what your network looks like, I suggested pasting it into this tool (and then pressing <shift> + <enter>): http://ethereon.github.io/netscope/#/editor.
    - This will also let you know if you have a "syntax error" somewhere in the file.
- My model got ~70% test accuracy in < 5 minutes of training. (Chance is 20%.)

At the end of this phase:
You will have the following files:

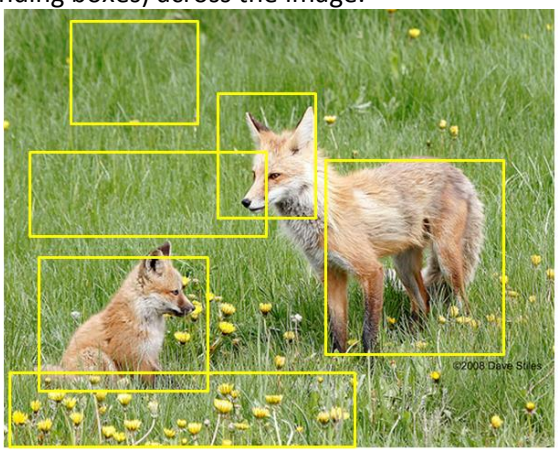
hw3-train.prototxt
hw3-solver.prototxt
hw3-weights.caffemodel
hw3-deploy.prototxt file

Phase 4: Test the CNN

This phase shouldn't require a lot of "new" code (as it combines code from hw2 and phase1 of hw3). Here's the general idea:

(A) We load our CNN.

(B) We take an image from our dataset. We use a "proposal generation algorithm" to produce "proposals" (i.e. bounding boxes) across the image:

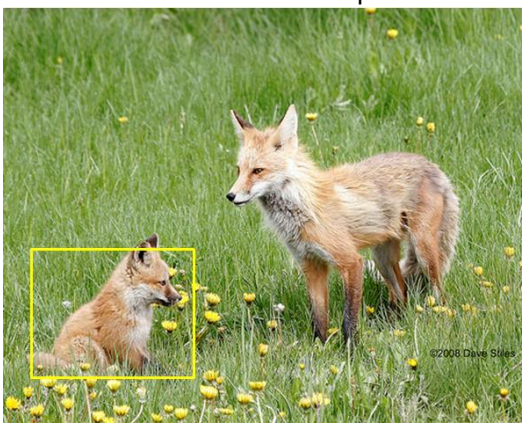


(C) We loop over each proposal and feed it into our CNN.

   a. If the output prediction is of the "background" class, we discard it.
   b. If the output prediction is of another class, we check to see if it is the highest score we've seen so far. If it is, we save it for later.

(D) Once we have gone through all the proposals, we have one proposal that scored the highest. We display the original image with the proposal's bounding box. We also display (or print) the predicted label (so we know what class the NN predicted).

   a. Hopefully the predicted label is from the class we expect!



"fox"

A few notes:

(a) Make sure your hw3-deploy.prototxt file has five output classes. (Don't forget the new background class!)

(b) When the highest prediction for a particular proposal is background, ignore that proposal. (We aren't interested in displaying the highest scoring proposal if the class is "background".)

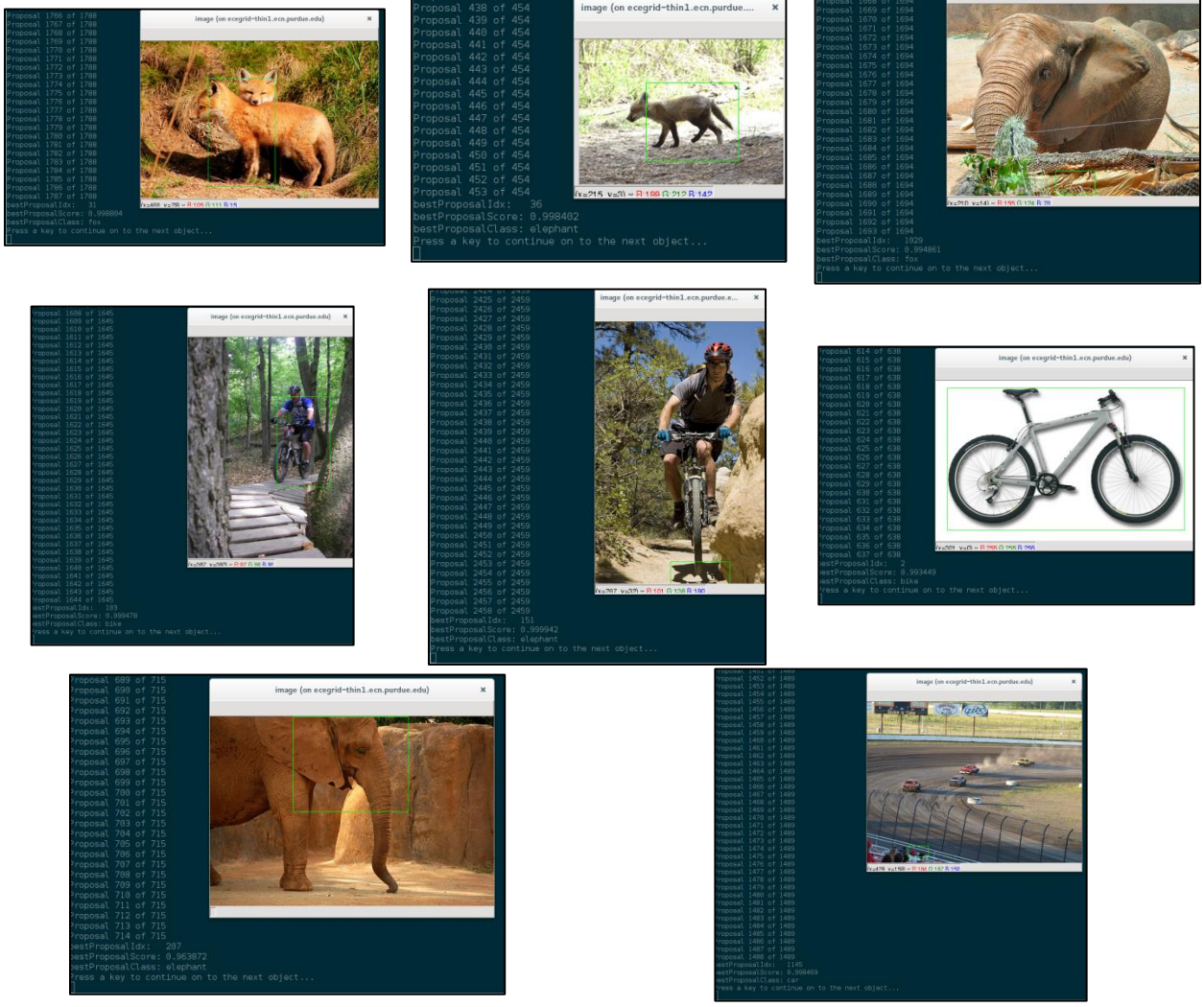
(c) When we were training the network in Phase 3, caffe warped all input images to the proper dimensions for us. Now, we don't have that luxury. We need to warp the image ourselves before inputting it into the network.

   a. You should be able to copy your code that you used in hw2.

(d) We need to input images into the “bottom blob” of the data layer and extract the predictions from the “top blob” of the last layer.
    a. You should be able to copy your code that you used in hw2.
(e) An example of how to load an image in python, overlay a rectangle, and display the image with the rectangle, can be found in hw3-guide-phase1.py
(f) For this homework, don’t worry about images that have multiple instances of the class (like the example of the two foxes above).
    a. If there are 2+ instances of a class in an image, follow the algorithm above (keep track of the proposal with the highest score and display it). Don’t worry about finding a bounding box for the additional instances of the class. That gets more tricky.

<u>At the end of this phase:</u>
You will have the following files:
hw3-phase4.py

You will also be able to see your network in action, localizing instances of each class. Wow! Cool! Congrats! You’re amazing. You should buy yourself some ice cream!
<u>Sample Outputs:</u>



Grading Scripts

You need to submit these files, named exactly this way:

- hw3-train.prototxt
- hw3-solver.prototxt
- hw3-deploy.prototxt
- hw3-weights.caffemodel
- hw3-phase1.py
- hw3-phase2.py
- hw3-phase4.py

Put these two files into a folder named exactly this way:

- hw3

Zip this file. (No tarballs, please.)

Upload the zipped file to Blackboard.

The grading script will do the following:

- Unzip the hw3.zip folder and extract the files.
    - If your files/folders are named wrong, you get 0 points on the homework.
- Load your network (i.e. "hw3-deploy.prototxt" with "hw3-weights.caffemodel")
    - If successful, you earn the points.
- Loop over 20 test images.
    - For each test image, create 10 proposals.
        - 9 of them have an IoU score of < 0.4
        - 1 of them has an IoU score of > 0.7
    - Feed each proposal into CNN
        - Perform a forward pass
            - If successful, you earn the points.
        - Keep track of which proposal got the highest score and its class label
- Calculate what percentage of the proposals/labels the CNN got right
    - If the CNN chose the correct proposal and assigned it the correct label 20% of the time, you earn the points (chance = 2%).

NOTE: My model (which got 70% test accuracy during training) scored 45% accuracy on the grader. You can see from the sample outputs I shared above that there are plenty of errors. The grader is a bit generous in that it only supplies one proposal with an IoU score of > 0.7.

ImageNet usage

You are using images from ImageNet as part of this homework. As such, you agree to abide by the following terms and conditions:

You (the "Researcher") has requested permission to use **the ImageNet database** (the "Database") at **Princeton University and Stanford University**. In exchange for such permission, Researcher hereby agrees to the following terms and conditions:

1. Researcher shall use the Database only for non-commercial research and educational purposes.
2. Princeton University and Stanford University make no representations or warranties regarding the Database, including but not limited to warranties of non-infringement or fitness for a particular purpose.
3. Researcher accepts full responsibility for his or her use of the Database and shall defend and indemnify the ImageNet team, Princeton University, and Stanford University, including their employees, Trustees, officers and agents, against any and all claims arising from Researcher's use of the Database, including but not limited to Researcher's use of any copies of copyrighted images that he or she may create from the Database.
4. Researcher may provide research associates and colleagues with access to the Database provided that they first agree to be bound by these terms and conditions.
5. Princeton University and Stanford University reserve the right to terminate Researcher's access to the Database at any time.
6. If Researcher is employed by a for-profit, commercial entity, Researcher's employer shall also be bound by these terms and conditions, and Researcher hereby represents that he or she is fully authorized to enter into this agreement on behalf of such employer.
7. The law of the State of New Jersey shall apply to all disputes under this agreement.