Tony You - Z23431759, Jacob Highbridge - Z23498763

04/15/2021

CAP 4401

Object Detection

Using Faster R-CNN Deep Learning

Motivation:

For HW2, our project was to use the image labeler understand the use for it and the situations when it can be applicable. During that project, we learned not only how to label image, but also why images should be labeled. We trained a fairly rudimentary ACF object detector to recognize certain objects in an image, given the training data from the image labeler (ground truth, training and testing images). On top of this, we are also both taking a class on deep learning, where we are learning a lot about convolutional neural networks, and ReLU activation.

Given this, we were both very interested in applying what we learned from HW2 and our class on Deep Learning to this assignment, and seeing how it could work in practice.

Background:

The Faster R-CNN was first published to NeurIPS on Jun 4th, 2015 by Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. The faster R-CNN introduces the integration of a Region Proposal Network (RPN) with the Fast R-CNN convolutional network. An RPN is already a CNN that can simultaniously predict object bounds and object scores at each position. The faster R-CNN merges this into a single network with Fast R-CNN, and uses the RPN as an "attention" mechanism, where the RPN component can "point" where the overall network should look.

A current issue with deep learning for object detection is generating an *n*-length list of bounding boxes. Usually when deep neural networks are modeled, the last block is a fixed-size tensor output. This problem can be solved using an RPN. RPNs use anchors; fixed size bounding boxes which are placed evenly throughout the original image.

Input images, represented as 3 dimensional arrays, are first passed through a pre-trained CNN, and we are given a convolutional layer map as a result. At this point, the RPN is used with the features given by the CNN to find up to *n* bounding boxes/anchors that may contain the objects we're looking for.

Key Paper: https://papers.nips.cc/paper/2015/hash/14bfa6bb14875e45bba028a21ed38046-Abstract.html

Other References: https://arxiv.org/abs/1506.01497, https://nips.cc/, https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/

Code Example (With Modifications)

Downloading Pretrained Detector (Original Testing Only)

This part of the code only applies to the original example, where the example data is downloaded and then used to run the example (vehicle detector).

```
%doTraining = false;
%if ~doTraining && ~exist('fasterRCNNResNet50EndToEndVehicleExample.mat','file')
%    disp('Downloading pretrained detector (118 MB)...');
%    pretrainedURL = 'https://www.mathworks.com/supportfiles/vision/data/fasterRCNNResNet50End
%    websave('fasterRCNNResNet50EndToEndVehicleExample.mat',pretrainedURL);
%end
```

Loading Data Needed For Object Detection

Instead of using vehicle data, we instead use the cat image labels from HW2, which have already been processed using the image labeler. On top of using the cat detection ground truth with the faster R-CNN, we also are going to test it with the YOLOv2 network and compare the results.

```
% Loading gTruth from my Image Labeling session
load('cat_image_labels.mat');
```

```
Warning: The data source points to one or more image files that cannot be found. Update the DataSource
filenames using changeFilePaths method.
```

```
% Adapting gTruth DataSource paths to a new computer so the program can run
currentPath = "C:\Users\Jacob\Documents\MATLAB\catsforlabels";
newPath = "cat_training";
alternativePaths = {[currentPath newPath]};
unresolvedPaths = changeFilePaths(gTruth, alternativePaths);
gTruth.DataSource;

% this value should be toggled based on which network is being tested
yolo = false;
% false if testing faster R-CNN
% true if testing YOLOv2

if yolo
    load('YOLOv2CatDetector.mat')
else
    load('fasterRCNNCatDetector.mat');
end
```

Loading Data Set

This part was a little bit difficult to set up. Originally, all that had to be done for example was as shown in the first three [commented out] lines below. The data was already available just from the vehicleDataset matrix.

To solve this problem, I created two n:1 matrixes of the label data and image source data from the gTruth, and then horizontally concatenated them to be processed just like the vehicleDataset matrix. The only issue here is that the name assigned to the image source column is "Var2," which isn't very descriptive. While not a script breaking problem, it is still a minor annoyance.

Note: The commented out lines of code are the original lines of code from the example, for comparison

```
% unzip vehicleDatasetImages.zip
% data = load('vehicleDatasetGroundTruth.mat');
% vehicleDataset = data.vehicleDataset;
```

```
source = gTruth.DataSource.Source;
labels = gTruth.LabelData;
catDataset = horzcat(labels, source);


% Splitting dataset into training, validation, and testing sets
% 60% for training, 10% for validation, and 30% for testing
rng(0)
% shuffledIndices = randperm(height(vehicleDataset));
% idx = floor(0.6 * height(vehicleDataset));
shuffledIndices = randperm(height(catDataset));
idx = floor(0.6 * height(catDataset));

trainingIdx = 1:idx;
% trainingDataTbl = vehicleDataset(shuffledIndices(trainingIdx),:);
trainingDataTbl = catDataset(shuffledIndices(trainingIdx),:);

validationIdx = idx+1 : idx + 1 + floor(0.1 * length(shuffledIndices) );
% validationDataTbl = vehicleDataset(shuffledIndices(validationIdx),:);
validationDataTbl = catDataset(shuffledIndices(validationIdx),:);

testIdx = validationIdx(end)+1 : length(shuffledIndices);
% testDataTbl = vehicleDataset(shuffledIndices(testIdx),:);
testDataTbl = catDataset(shuffledIndices(testIdx),:);
```

Creating datastores to load image and label data later during training & eval

Here, datastores are created to hold the image sources and image labels that were just shuffled in the last section.

```
imdsTrain = imageDatastore(trainingDataTbl{:,'Var2'});
bldsTrain = boxLabelDatastore(trainingDataTbl(:,'Cat'));

imdsValidation = imageDatastore(validationDataTbl{:,'Var2'});
bldsValidation = boxLabelDatastore(validationDataTbl(:,'Cat'));

imdsTest = imageDatastore(testDataTbl{:,'Var2'});
bldsTest = boxLabelDatastore(testDataTbl(:,'Cat'));

% Combining image and label datastores
trainingData = combine(imdsTrain,bldsTrain);
validationData = combine(imdsValidation,bldsValidation);
testData = combine(imdsTest,bldsTest);
```

We test the data here to check that an image is labeled correctly:

```
data = read(trainingData);
I = data{1};
bbox = data{2};
annotatedImage = insertShape(I,'Rectangle',bbox);
annotatedImage = imresize(annotatedImage,2);
figure
imshow(annotatedImage)
```

Creating Faster R-CNN Detection Network

The image input to the R-CNN is normalized, the number of anchors is declared, and the number of anchor boxes is estimated from the size of the training objects.

```
inputSize = [224 224 3];
preprocessedTrainingData = transform(trainingData, @(data)preprocessData(data,inputSize));
numAnchors = 7;
anchorBoxes = estimateAnchorBoxes(preprocessedTrainingData,numAnchors)
```

```
anchorBoxes = 7×2
    40    35
   153   190
    79    58
    91   129
   148   125
   203   200
   112    90
```

ResNet-50 is used as the CNN to extract features before the RPN, the feature activation layer is defined as ReLU, and the number of classes are detected.

```
featureExtractionNetwork = resnet50;
featureLayer = 'activation_40_relu';
numClasses = width(catDataset)-1;
```

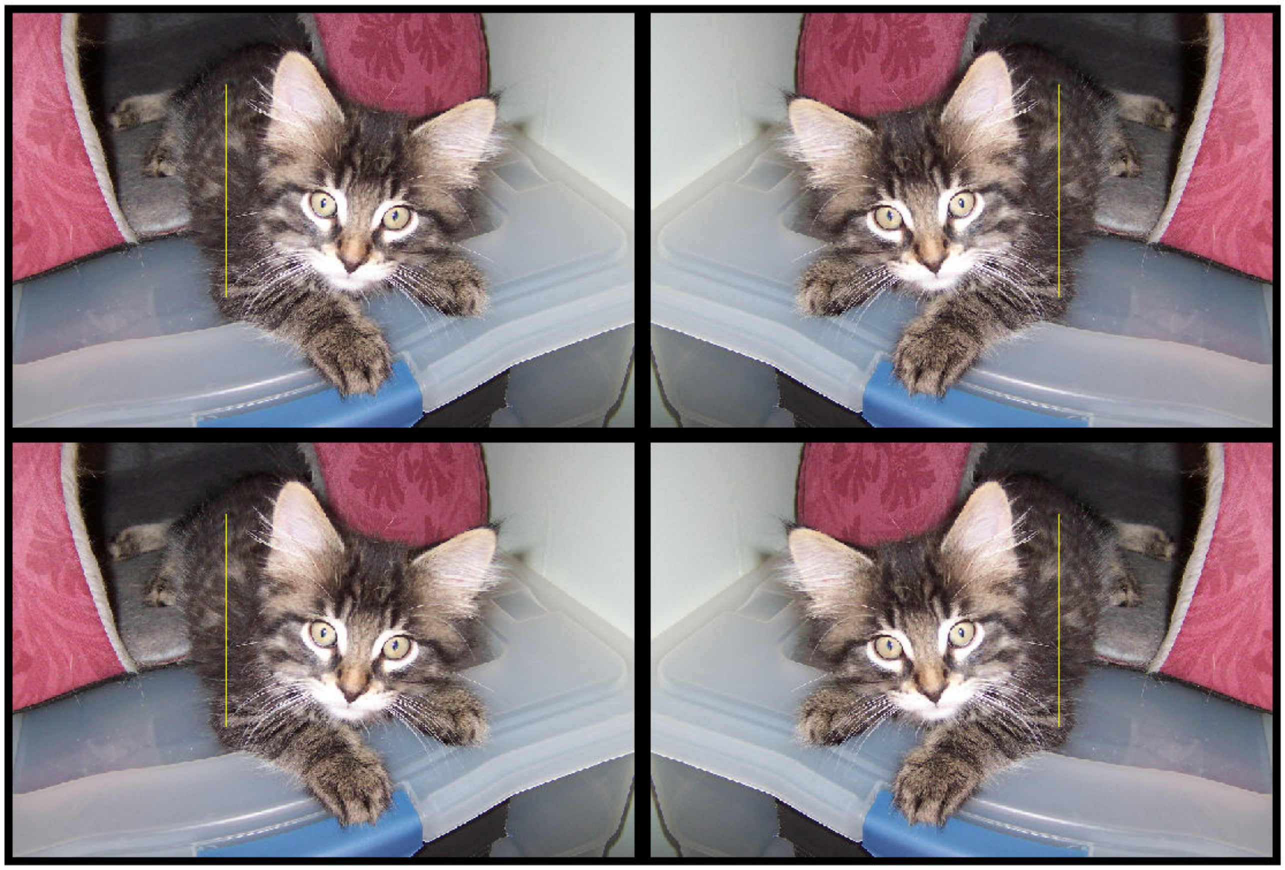
```
% The object detection network is created based on wether YOLOv2 or faster
% R-CNN is selected
if yolo
    lgraph = yolov2Layers(inputSize,numClasses,anchorBoxes,featureExtractionNetwork,featureLaye
else
    lgraph = fasterRCNNLayers(inputSize,numClasses,anchorBoxes,featureExtractionNetwork,feature
end
```

Data Augmentation

Training data is augmented to provide an increased variety of training data without having to increase the number of labeled samples.

```
augmentedTrainingData = transform(trainingData,@augmentData);

augmentedData = cell(4,1);
for k = 1:4
    data = read(augmentedTrainingData);
    augmentedData{k} = insertShape(data{1},'Rectangle',data{2});
    reset(augmentedTrainingData);
end
figure
montage(augmentedData,'BorderSize',10)
```



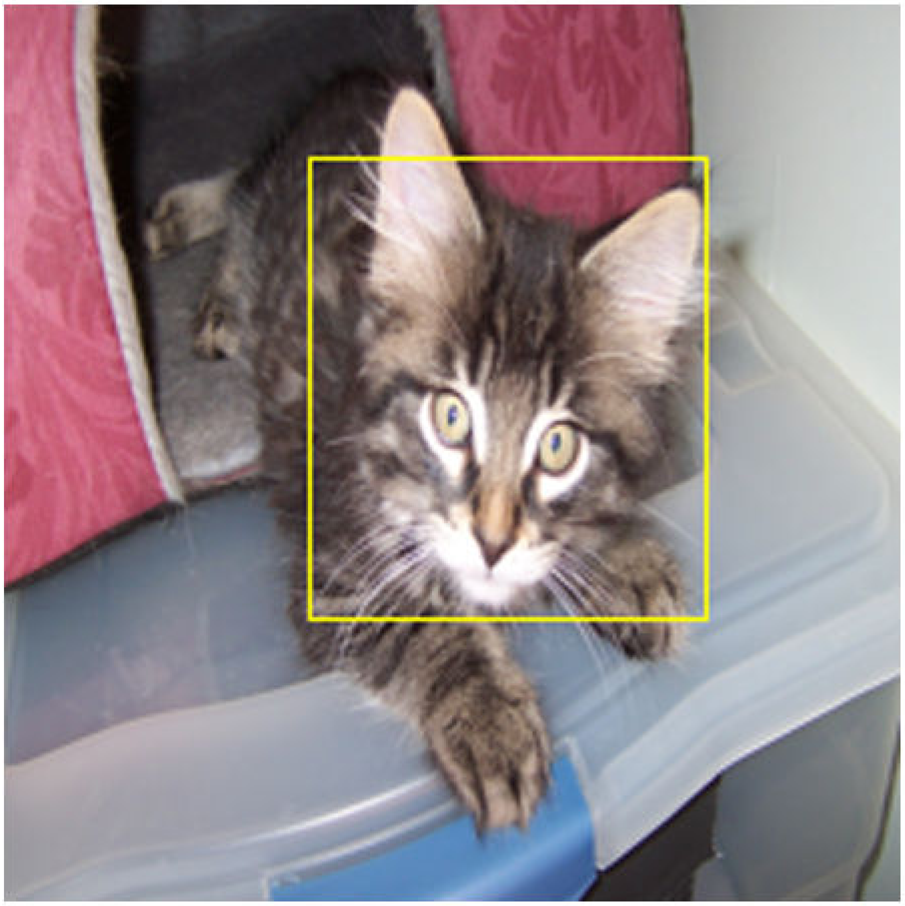
Preprocessing Training Data

Processing training and validation data for training.

```
trainingData = transform(augmentedTrainingData,@(data)preprocessData(data,inputSize));
validationData = transform(validationData,@(data)preprocessData(data,inputSize));

data = read(trainingData);

% display a preprocessed image and bounding box
I = data{1};
bbox = data{2};
annotatedImage = insertShape(I,'Rectangle',bbox);
annotatedImage = imresize(annotatedImage,2);
figure
imshow(annotatedImage)
```



Training Faster R-CNN and YOLOv2

Now here's the real meat of the project. My first run through training the network was extremely slow, which I discovered was because MATLAB was only utilizing a single core from my CPU. To fix this, I downloaded the Parallel Computing Toolbox to train the network using my GPU, which is a pascal archtecture NVIDIA card (GTX 1080).

Another thing to note was my accuracy while training with a GPU instead of a CPU shot way up. I'm not sure if there is a correlation here, but I do appreciate it.

The results after initially training the network were okay. It was able to identify cats with a precision of about 0.66, and the confidence of the labeled images were around .50 to .60. This is okay, but I think we can do better after adjusting the options.

Training YOLOv2

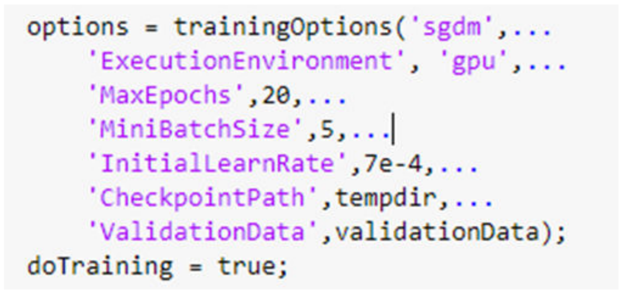
When training YOLOv2, I change the max epochs to 20 and the mini batch size to 5, then leave the learning rate the same. However, there is a problem getting it to properly label anything. When evaluating the detector, I got a percision of 0.00. This was obviously a problem, so I did some research until I came across this message on the MATLAB forums:

https://www.mathworks.com/matlabcentral/answers/460276-issues-with-yolo-v2-network#answer_538520

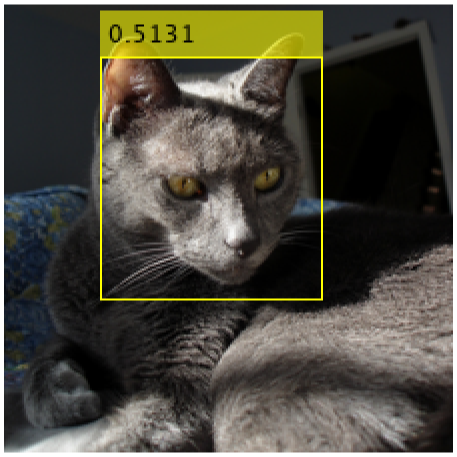
Apparently, the documentation on the YOLOv2 detector is innacurate, and a lower learning rate is needed to aleve these problems. After lowering the learning rate to 1e-4, I finally get some convergence in the network. After this, I test different learning rates until I get the best precision:

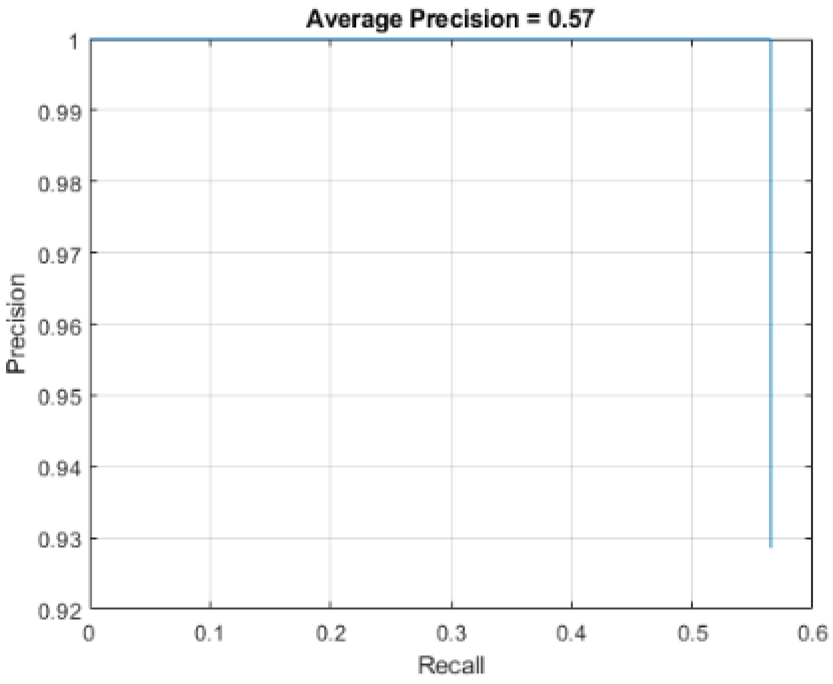
- First test: 1e-4, 0.09 precision
- Second test: 3e-4, 0.13 precision
- Third test: 5e-4, 0.26 precision
- Fourth test: 8e-4, 0.55 precision
- Fifth test: 9e-4, 0.30 precision
- Last test: 7e-4, **0.57 precision**

Thus, the best learning rate I was able to find was 7e-4, with a precision of 0.57.



```
options = trainingOptions('sgdm',...
    'ExecutionEnvironment', 'gpu',...
    'MaxEpochs',20,...
    'MiniBatchSize',5,...
    'InitialLearnRate',7e-4,...
    'CheckpointPath',tempdir,...
    'ValidationData',validationData);
doTraining = true;
```

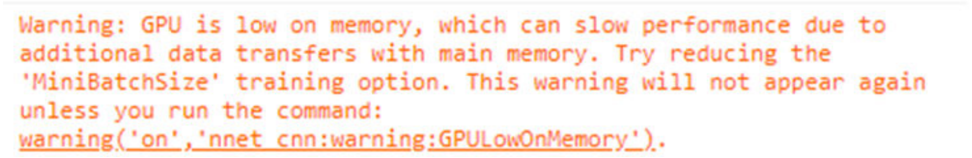




Training faster R-CNN

The same testing done to the YOLOv2 network is now also done to the faster R-CNN.

When training a faster R-CNN network with the same options as the YOLOv2 network (except the learning rate was changed back to 1e-3). However, I ran into an interesting error after an exceptionally long run:



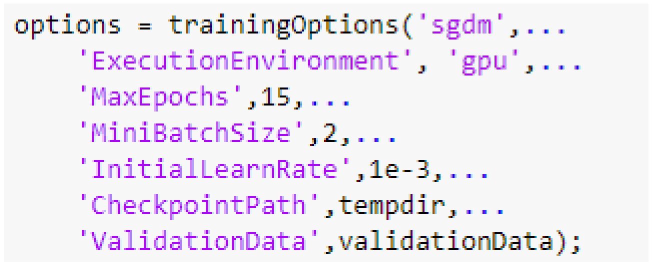
```
Warning: GPU is low on memory, which can slow performance due to
additional data transfers with main memory. Try reducing the
'MiniBatchSize' training option. This warning will not appear again
unless you run the command:
warning('on','nnet_cnn:warning:GPULowOnMemory').
```

Apparently, the batch size of 5 is too large for training the faster R-CNN on my gpu, so I lowered it to 4, then 3, then down to it's original size of 2. After this, I test different options until I get the best precision:

- First test: 20 epochs, lr 1e-3, 0.66 precision
- Second test: 15 epochs, lr 1e-3, **0.91 precision**
- Third test: 10 epochs, lr 1e-3, 0.53 precision
- Last test: 15 epochs, lr 3e-3, 0.34 precision

Training the faster R-CNN network also took significantly longer to train compared to the YOLOv2 network.

Thus, the best options I found were 15 epochs, and a learning rate of 1e-3, with a precision of 0.91.



```
options = trainingOptions('sgdm',...
    'ExecutionEnvironment', 'gpu',...
    'MaxEpochs',15,...
    'MiniBatchSize',2,...
    'InitialLearnRate',1e-3,...
    'CheckpointPath',tempdir,...
    'ValidationData',validationData);
```

```
options = trainingOptions('sgdm',...
    'ExecutionEnvironment', 'gpu',...
    'MaxEpochs',15,...
    'MiniBatchSize',2,...
    'InitialLearnRate',1e-3,...
    'CheckpointPath',tempdir,...
    'ValidationData',validationData);
doTraining = false;

if doTraining
    if yolo
        [detector,info] = trainYOLOv2ObjectDetector(trainingData,lgraph,options);
    else
        % Train the Faster R-CNN detector.
        % * Adjust NegativeOverlapRange and PositiveOverlapRange to ensure
        %   that training samples tightly overlap with ground truth.
        [detector, info] = trainFasterRCNNObjectDetector(trainingData,lgraph,options, ...
        'NegativeOverlapRange',[0 0.3], ...
        'PositiveOverlapRange',[0.6 1]);
    end
    % Train the Faster R-CNN detector.
    % * Adjust NegativeOverlapRange and PositiveOverlapRange to ensure
    %   that training samples tightly overlap with ground truth.

%else
    % Load pretrained detector for the example.
 %   pretrained = load('fasterRCNNResNet50EndToEndVehicleExample.mat');
 %   detector = pretrained.detector;
end
```

Comparing faster R-CNN and YOLOv2

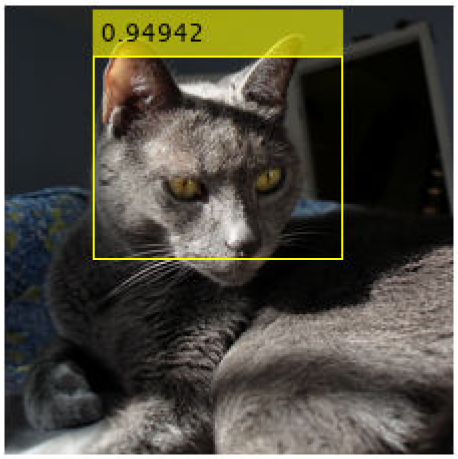
YOLOv2 was a much lighter network to train compared to the faster R-CNN. It was easier enough on memory allocation that batch sized can be significantly larger, and more epochs can be cycled in a shorter period of time. On top of that, the learning rate of the YOLOv2 network should also be significantly lower than the faster R-CNN.

However, while the faster R-CNN network is clearly more beefy and complex, I think the results speak for themselves. The precision rate of the faster R-CNN network is much much higher than the YOLOv2 network.

Testing Trained Network on Test Images

```
% Running trained detector on 1 image as a test
I = imread(testDataTbl.Var2{20});
I = imresize(I,inputSize(1:2));
[bboxes,scores] = detect(detector,I);

% Displaying results
I = insertObjectAnnotation(I,'rectangle',bboxes,scores);
figure
imshow(I)
```



Evaluating Detector Using Test Data

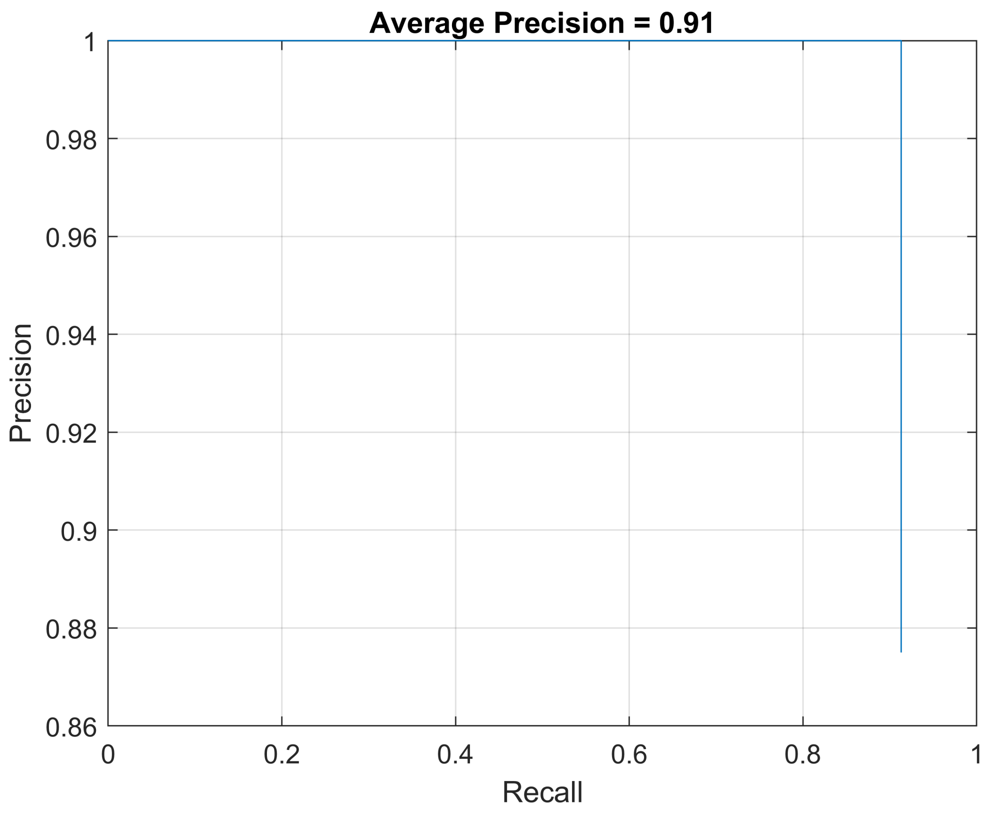
```
testData = transform(testData,@(data)preprocessData(data,inputSize));

detectionResults = detect(detector,testData,'MinibatchSize',4);

[ap, recall, precision] = evaluateDetectionPrecision(detectionResults,testData);

figure
plot(recall,precision)
xlabel('Recall')
ylabel('Precision')
```

```
grid on
title(sprintf('Average Precision = %.2f', ap))
```



```
file = 'YOLOv2CatDetector';
save(file, 'detector', 'info');
```

Lessons Learned

There is a very large difference between just training an ACF object detector in a few lines of code and trying it on a few images, and going though a the whole process of augmenting data, processing that data, and then using it to train a convolutional network.

The difference in confidence scores is almost uncomparable, even with the YOLOv2 network. It's also good to have first hand experience with the difference that augmenting data can make when training a network.

Future Ideas

Given extra time, we could use the faster R-CNN and YOLOv2 networks for semantic image segmentation. As of now, they're only labeling one box on a picture that only has ground truth for one object. On top of that, the YOLOv2 network can surely be more fine-tuned for higher precision.

Supporting Functions

```
function data = augmentData(data)
% Randomly flip images and bounding boxes horizontally.
tform = randomAffine2d('XReflection',true);
sz = size(data{1});
```

```
rout = affineOutputView(sz,tform);
data{1} = imwarp(data{1},tform,'OutputView',rout);

% Sanitize box data, if needed.
data{2} = helperSanitizeBoxes(data{2}, sz);

% Warp boxes.
data{2} = bboxwarp(data{2},tform,rout);
end

function data = preprocessData(data,targetSize)
% Resize image and bounding boxes to targetSize.
sz = size(data{1},[1 2]);
scale = targetSize(1:2)./sz;
data{1} = imresize(data{1},targetSize(1:2));

% Sanitize box data, if needed.
data{2} = helperSanitizeBoxes(data{2}, sz);

% Resize boxes.
data{2} = bboxresize(data{2},scale);
end
```