Machine Vision for Safer Self Driving Cars in Construction Zones

Rick Alayza, Trincy Thomas Kozhikkadan, Krystofer Newman

***Abstract*— Construction Zones are often very different from one another, making automatic road sign detection and navigation in these areas a challenge for self-driving cars. In this project our goal is to develop a Convolution Neural Network (CNN) that can correctly classify road signs based on the color and geometric information in the input image or video and use the results for more accurate and safer navigation in self-driving cars**

Keywords—Convolution Neural Network, Transfer Learning, Road sign detection, R-CNN, Faster R-CNN

I. Introduction

Automatic road sign detection is an important feature of self-driving cars. Safety of the self-driving cars depends on the car's ability to accurately recognize road signs and make decisions based on the information obtained from the road sign. An example would be being able to read a vertical channelizing road sign that is commonly used in marking paths during road construction [3]. By taking advantages of the special characteristics of traffic signs, typically the color and geometric information in the images or video to classify the sign and analyze it to help the car make decisions related to navigation [1]. The vertical channeling sign is easily recognizable by human drivers and relays vital information in the sign's geometry. This type of sign is generally used in road construction zones. The intended purpose of this sign is to mark the path for traffic to follow. The angle of the stripped lines also has a significance. These lines slope down toward the path. In Figure 1, the channeling sign is indicating that the path is to the left of the sign. The system should be able to maintain a good level of accuracy to detect and recognize road signs under varying lighting conditions, reduced clarity due to factors like bad weather or the view angles from the car-mounted cameras to the traffic signs may lead to artificially rotated and distorted images [2]. The important problem statement is how can machine vision correctly classify and use this information for a self-driving car application. The CNN implemented in our project will also be able to classify road signs in a video input providing a more practical solution for self-driving cars.

.

Figure 1: Example of vertical channelizing road signs used in construction zones [3]

II. Approach - Description

The success of traditional methods for solving computer vision problems heavily depends on the feature extraction process. But Convolutional Neural Networks (CNN) have provided an alternative for automatically learning the domain specific features through the use of region of interest (ROI). Now every problem in the broader domain of computer vision is re-examined from the perspective of this new methodology [4]. Various object detection methods for traffic-sign classification based on SVMs [11] and sparse representations are used but recently, CNN has outperformed the existing methods since the launch of the German traffic-sign detection [12]. Auto Encoder, sparse coding, Restricted Boltzmann Machine, Deep Belief Networks and Convolutional neural networks is commonly used models in deep learning [5]. Image classification being one of the main tasks of our project, we chose CNN as our approach to solve the problem because among different type of models, Convolutional neural networks have demonstrated high performance on image classification [5]. Convolutional neural networks with fixed and learnable layers can be used for detection and recognition. The fixed number of layer can reduce the amount of interest areas to detect and crop the boundaries very close to the borders of traffic signs [13]. The learnable layers can increase the accuracy of detection significantly [13].

A. *Transfer Learning*

The Transfer Learning approach was used to implement the CNN. Transfer learning is a deep learning approach in which a model that has been trained for one task is used as a starting point to train a model for similar task. Fine-tuning a network with transfer learning is usually much faster and easier than training a network from scratch. Transfer learning is a popular technique because it allows to train models using relatively little labeled data by leveraging popular models that have already been trained on large datasets [6]. Traditionally, a CNN is trained with dataset that consists of thousands of images with a specified number of classes, or classifications. Transfer learning allows for the user to use an established CNN and repurpose it as a specialized CNN for detection as depicted in Figure 3. This ability to use an existing, well defined, established CNN reduces the number images in the dataset from the thousands to only dozens. This solution was heavily researched due to the lack of defined datasets that included these channeling road signs. Transfer learning can reduce training time and compute resources. With transfer learning, the model does not need to be trained for as many epochs (a full training cycle on the entire dataset) as a new model would require. The graph below shows the network performance for models with transfer learning and models trained from scratch. With transfer learning, it is possible to achieve a higher model accuracy in a shorter time [6]. With the ability greatly reduce the number of images required to train the CNN, a dataset was generated.

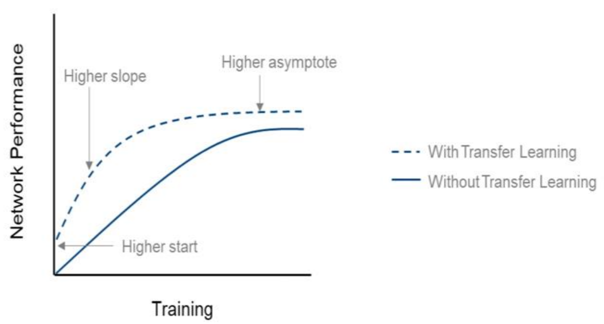

Figure 2: Network performance of training from scratch and transfer learning [6]

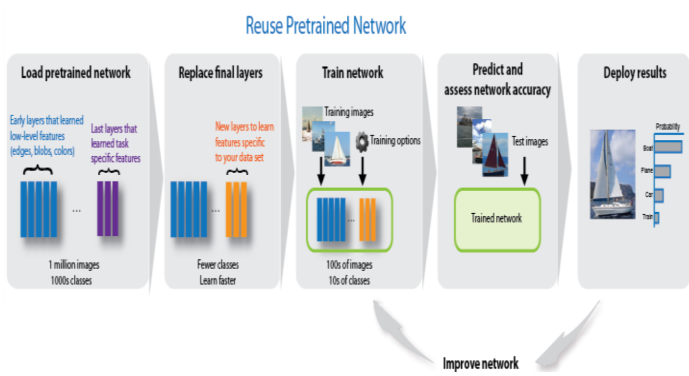

Figure 3: Transfer Learning Workflow [6]

III. Approach Algorithm/Methods

Matlab 2018b was main tool used for implementing the CNN model. The project required installation of Image Processing, Computer Vision System, Statistics and Machine Learning, Parallel Computing toolboxes in Matlab. The pretrained CNN is loaded in Matlab. This CNN has been pretrained with the CIFAR-10 dataset. The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class [7]. The network has learned rich feature representations for a wide range of images. The network takes an image as input and outputs a label for the object in the image together with the probabilities for each of the object categories [8]. To retrain the pretrained network to classify new images, the last few layers of the network are replaced. The final layers are set to match the number of classes in the new data set. The images in the training dataset are labelled using the image labeler application in Matlab. The Image Labeler app labels rectangular regions of interest (ROIs) for object detection, pixels for semantic segmentation, and scenes for image classification [9]. Using the Image Labeler app, we interactively specified ‘Pass left’ and ‘Pass Right’ labels to the images in the training dataset. The CNN was then trained using this labeled collection of training images. The CNN was then tested with the images in the test dataset.

A. *Established Data – D1*

Initial step to implement transfer learning is to select and load a pretrained network. The classification layers for the new task are replaced based on the relevant application followed by fine-tuning the weights depending on the new task and data available. The model is then tested on the test dataset to check accuracy [6]. This section will provide an abbreviated setup process for the neural network. More specific instructions are listed in the associated website [8]. The first step was to install the appropriate toolboxes MATLAB would need to create and train the convolution neural network. Neural Network, Statistics and Machine Learning, Computer Vision System, and Image Processing Toolboxes were required for the experiment. The Parallel Computing Toolbox was option since this toolbox only lowers the processing time during training. Along with the toolboxes, a dataset of 50,000 images as loaded into the workspace. In older version of MATLAB, this dataset was download but comes part of the Image Processing Toolbox in the current version. The sample dataset consisted of 10 categories. Figure 4 is a sample of the dataset generated using the supplied MATLAB commands in the tutorial. It is important to note that the images in Figure 4 are relatively small and have a low resolution. This is intentional because larger images at a higher resolution would significantly increase training times and be beyond the scope of the tutorial.

The next step was to create the convolutional neural network (CNN). The Neural Network Toolbox simplifies this process since the CNN is a layered network. This experiment used seven layers. Image input layer, 2D convolution layer, and classification output layer are some examples of the layers. The commands listed in the tutorial can be repeated with downsampling to create a deeper network. This downsampling may impact training if important or useful information is discarded. The experiment did not repeat the layering commands. A single execution was used to create the CNN.

Once the neural network was created, it needed to be trained. The training consisted of the CNN scanning the layers and formulating classifications. Those formulations were then compared to the classifications in the dataset to calculate the CNN accuracy. The tutorial offered two options: to load training data or to execute the training. Loading the training data was not as accurate in object detection but required less processing time. Executed training averaged around 45-50 minutes. The training was separated into two different runs. The first training run was performed on the loaded dataset. This training performed 15,600 iterations and lasted 55 minutes. The average accuracy achieved was 85%. This percentage can be increased with a deeper network through more trainings. To correctly ensure that the CNN was created/trained correctly, a validation was executed.

Figure 4: Sample of D1 Dataset Images

B. *Specialized Dataset – D2*

The second dataset was designed to convert the established CNN to a Region Convolutional Neural Network (R-CNN). This type of CNN would analyze image regions to detect and identify objects. This dataset was originally a collection of images of construction zone road signs downloaded using Google ™. This dataset was comprised of 34 still images with two classes. PL for Pass Left and PR for Pass Right. The training from this dataset netted good results when tested against other still images. When tested against a video, the R-CNN was not able to detect the signs and also failed to behave correctly. Figure 6 and Figure7 are examples actual output from the R-CNN when tested on still images. The inability to analyze video required that the team reevaluate what type of neural network to implement. Research led to the concept of a Faster R-CNN (FR-CNN). The FR-CNN operates in a similar fashion as a R-CNN with the exception of how regions are treated. In a R-CNN, a different algorithm is used to extract a region proposal, area that may have the desired object. This algorithm operates outside the CNN and will lead to a bottleneck effect when images are in high resolution or speed of execution is vital. For this instance, execution speed was the limiting factor and contributor to the implementation failure. A FR-CNN trains the ROI algorithm into the neural network. This training greatly reduces execution, image analysis, but exponentially increases training time. For example: The R-CNN was trained in 20 minutes while the FR-CNN was trained in 2 hours for the same dataset. The next process was to expand the dataset from 34 images. This was done using Matlab's Ground Truth Labeler App (Figure 5) to mark ROI's on a video sample. Using the application, the dataset grew from 34 images to well over 140 images. With that increase in the dataset, training also increased to 3.5 hours.

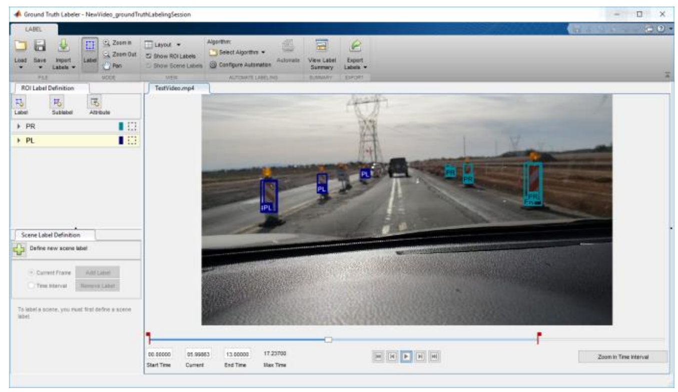

Figure 5: Matlab Ground Truth Labeler Application

Listed below was the training output for the FR-CNN. The training was divided into 4 steps. Step 1 trained a neural network to act as a Regional Proposal Network (RPN). The function of the RPN was to analyze the image and generated possible areas that may contain an object, PR or PL, and input those regions into the R-CNN for classification. Step 2 re-trained the R-CNN using the regions extracted by the newly developed RPN. Step 3 and Step 4 are a repeat of the first two steps to enhance both network accuracies. The dataset was divided into training and test sub -dataset. The mini-batch accuracy is the accuracy of the network, either RPN or R-CNN, on the test sub-dataset.

```
*************************************************************************
Training a Faster R-CNN Object Detector for the following object classes:

* PL
* PR

Step 1 of 4: Training a Region Proposal Network (RPN).
Training on single GPU.
```

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Mini-batch RMSE	Base Learning Rate
1	1	00:00:00	50.00%	0.89	0.0010
3	200	00:01:22	100.00%	1.54	0.0010
6	400	00:02:45	100.00%	1.13	0.0010
9	600	00:04:07	100.00%	1.01	0.0010
10	740	00:05:05	75.00%	1.10	0.0010

```
Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
*******************************************************************
Training a Fast R-CNN Object Detector for the following object classes:

* PL
* PR

--> Extracting region proposals from 74 training images...done.

Training on single GPU.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     RMSE     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:00 |        0.00% |         0.80 |          0.0010 |
|       3 |         200 |       00:00:58 |       75.00% |         0.62 |          0.0010 |
|       6 |         400 |       00:01:57 |       75.00% |         0.58 |          0.0010 |
|       9 |         600 |       00:02:55 |      100.00% |         0.71 |          0.0010 |
|      10 |         720 |       00:03:30 |       75.00% |         0.43 |          0.0010 |
|=========================================================================================|

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Starting parallel pool (parpool) using the 'local' profile ...
connected to 4 workers.
Training on single GPU.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     RMSE     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:00 |       75.00% |         0.91 |          0.0010 |
|       3 |         200 |       00:01:00 |      100.00% |         0.85 |          0.0010 |
|       6 |         400 |       00:02:00 |       50.00% |         1.37 |          0.0010 |
|       9 |         600 |       00:03:01 |       75.00% |         0.79 |          0.0010 |
|      10 |         740 |       00:03:43 |       75.00% |  |      1.51 |          0.0010 |
|=========================================================================================|


Step 4 of 4: Re-training Fast R-CNN using updated RPN.
*******************************************************************
Training a Fast R-CNN Object Detector for the following object classes:

* PL
* PR

--> Extracting region proposals from 74 training images...done.
|
Training on single GPU.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     RMSE     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:00 |       75.00% |         0.85 |          0.0010 |
|       4 |         200 |       00:00:38 |      100.00% |         0.47 |          0.0010 |
|       7 |         400 |       00:01:17 |       75.00% |         0.69 |          0.0010 |
|      10 |         600 |       00:01:55 |      100.00% |         0.61 |          0.0010 |
|      10 |         630 |       00:02:01 |      100.00% |         1.00 |          0.0010 |
|=========================================================================================|

Finished training Faster R-CNN object detector.
```

IV. RESULTS

The final step for this laborious report was to test eh final FR-CNN. The first run of tests was against images and video used to train the network. The second test was on video that was not part of the training.

A. Duplication of Reference

The final FR-CNN was successfully able to identify and label 'Pass left' and 'Pass Right' road signs for still images R-CNN (Figure 6 and Figure 7). The FR-CNN developed could successfully classify and label 'pass left' and 'pass right' signs in the images belonging to our test data set with a good amount of accuracy similar to the examples in the Matlab documentation related to transfer learning which we used as one our references [10]. The FR-CNN could also correctly identify and label multiple objects in an image which was a limitation not addressed in the Matlab documentation. This was remedied by changing options in the FR-CNN output. The CNN could also differentiate between pass left and pass right signs in the same image, identify different color and stripe patterns. Figure 8 is an example of the output when a video is used as an input. This section of video was also used in the training (Figure 5). Here, the FR-CNN was able to classify the objects in the image but had overlapping boundaries and low accuracy. The sign on the left was identified with 3 separate boundary boxes with accuracies ranging from 64% to 87%. This was unacceptable and caused for the second training of the FR-CNN with the same dataset as the original training. Figure 9 is the output of the FR-CNN on the same section of video as Figure 8 after a second training session. It is evident to the simplest of minds that the second training made a more robust FR-CNN. I was marked as a success with the increase in accuracies and the decrease of overlapping boundary boxes.

Figure 6: Labeled output image of different stripe pattern and color

Figure 7: Labeled output image showing correct identification of pass left and pass right signs

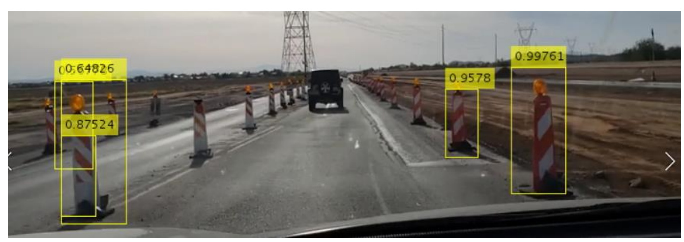

Figure 8: FR-CNN Video Output after 1 training

Figure 9: FR-CNN Video Output after 2 trainings

B. Tests on New Data

The FR-CNN was tested on video that was not part of the training dataset. The FR-CNN had an output that better than expected. The team originally expected that the FR-CNN would not be able to identify any of the objects. In general, the FR-CNN was able to classify all the target objects in the video. Unfortunately, the majority of the objects were incorrectly classified by the FR-CNN. Figure 10 is an example of a correct classification. Here, the sign is a pass right object marker. Figure 11 is an example of an incorrect classification and an incorrect boundary box. Interestingly, Figure 12 is what really made this test and all the hassle of this project worth the work. When Rick Alayza was recording this video while driving, he was not aware of the improperly placed sign by some unknow construction worker. This was only observed when analyzing the FR-CNN video output. The FR-CNN correctly classified the road sign when the human driver failed to do the same after 20 plus years for training.

Figure 10: Second Test of FR-CNN Output Sample 1

Figure 11: Second Test of FR-CNN Output Sample 2

Figure 12: Second Test of FR-CNN Output Sample 3

V. Conclusions and Discussions

Transfer learning allowed for targeted image detection. F-RCNN was trained using 140 sample images rather than thousands. All sample images were taken from one video input and some random internet search for still images. The FR-CNN was able to correctly, with a high level of accuracy, classify all objects when the training video was used as an input. It is important to note that 140 object instances were extracted from the video for training, but the FR-CNN was able to identify more than 350 instances in the entire training video. That is a relatively good return when faced with possible requirement of manually classifying thousands of images just to for an inaccurate, rudimentary CNN. Training with images and ground truth tables from a broader set will only increase the FR-CNN's versatility and accuracy.

References

[1] A. de la Escalera, L. Moreno, E.A. Puente, M.A. Salichs, "Neural traffic sign recognition for autonomous vehicles", 20th Annual Conference of IEEE Industrial Electronics.

[2] Hsiu-Ming Yang, Chao-Lin Liu, Kun-Hao Liu, and Shang-Ming Huang, "Traffic Sign Recognition in Disturbing Environments"

[3] ThreeD Plastics, "Traffic Work," Reynold Group Web, 2018. [Online]. Available: http://www.trafficwks.com/products/vertical-panels. [Accessed 13 Sept. 2018].

[4] Neena Aloysius ; M. Geetha, "A review on deep convolutional neural networks", 2017 International Conference on Communication and Signal Processing (ICCSP)

[5] Tianmei Guo ; Jiwen Dong ; Henjian Li ; Yunxing Gao, "Simple convolutional neural network on image classification", 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)

[6] https://www.mathworks.com/discovery/transfer-learning.html

[7] https://www.cs.toronto.edu/~kriz/cifar.html

[8] https://www.mathworks.com/help/deeplearning/examples/get-started-with-transfer-learning.html

[9] https://www.mathworks.com/help/vision/ug/get-started-with-the-image-labeler.html

[10] https://blogs.mathworks.com/pick/2017/02/24/deep-learning-transfer-learning-in-10-lines-of-matlab-code/

[11] S. Maldonado-Bascon, S. Lafuente-Arroyo, P. Gil-Jimenez, H. Gomez-Moreno, and F. Lopez-Ferreras. Road-sign detection and recognition based on support vector machines. Intelligent Transportation Systems, IEEE Transactions on, 8(2):264–278, June 2007

[12] Zhe Zhu, Dun Liang, Songhai Zhang, "Traffic-Sign Detection and Classification in the Wild"

[13] Yihui Wu ; Yulong Liu ; Jianmin Li ; Huaping Liu ; Xiaolin Hu, "Traffic sign detection based on convolutional neural networks", 2013 International Joint Conference on Neural Networks (IJCNN)

VI. APPENDIX

A. Matlab Code

```
%%
% Download Image Data - CIFAR-10
cifar10Data = tempdir;
url = 'https://www.cs.toronto.edu/~kriz/cifar-10-matlab.tar.gz';
helperCIFAR10Data.download(url,cifar10Data);

%%
% Load CIFAR-10 Training
[trainingImages, trainingLabels, testImages, testLabels]=...
   helperCIFAR10Data.load(cifar10Data);
%%
% Data Sample
size(trainingImages)
numImageCategories = 10;
categories(trainingLabels)
figure
thumbnails = trainingImages(:,:,:, 1:100);
montage(thumbnails)
%%
% Create Image Layers
[height, width,numChannels,~] = size(trainingImages);
imageSize = [height width numChannels];
inputLayer = imageInputLayer(imageSize)
% Convolution Layer Parameters
filterSize = [5 5];
numFilters = 32;
middleLayers = [
   convolution2dLayer(filterSize, numFilters, 'Padding', 2)
   reluLayer()
   maxPooling2dLayer(3, 'Stride', 2)
   convolution2dLayer(filterSize, numFilters, 'Padding', 2)
   reluLayer()
   maxPooling2dLayer(3, 'Stride',2)
   convolution2dLayer(filterSize, 2 * numFilters, 'Padding', 2)
   reluLayer()
   maxPooling2dLayer(3, 'Stride',2)
]
% CNN Final Layer
finalLayers = [
   fullyConnectedLayer(64)
   reluLayer
   fullyConnectedLayer(numImageCategories)
   softmaxLayer
   classificationLayer
]
%%
% Combine Input, Middle, and Final Layers
layers = [
   inputLayer
   middleLayers
   finalLayers
   ]
% Initialize Convolution Layer Weights
layers(2).Weights = 0.0001 * randn([filterSize numChannels
numFilters]);
%%
% Train CNN - Set Network Training Options
opts = trainingOptions('sgdm', ...
   'Momentum', 0.9, ...
   'InitialLearnRate', 0.001, ...
   'LearnRateSchedule', 'piecewise', ...
   'LearnRateDropFactor', 0.1, ...
   'LearnRateDropPeriod', 8, ...
   'L2Regularization', 0.004, ...
   'MaxEpochs', 40, ...
   'MiniBatchSize', 128, ...
   'Verbose', true,...
   'Plots','training-progress');
%%
% Train CNN - Execute Training (20-30 minutes)
%true to train network (20-30 minutes)
%false to load pre-trained network
doTraining = true;

if doTraining
   % Train a network.
   cifar10Net = trainNetwork(trainingImages, trainingLabels, layers,
opts);
else
   % Load pre-trained detector for the example.
   load('rcnnStopSigns.mat','cifar10Net')
end
%% *****
% Validate Training - Learned Edges
w = cifar10Net.Layers(2).Weights;
w = rescale(w);
figure
montage(w)
%% *****
% Validate Training - Test Set
YTest = classify(cifar10Net, testImages);
accuracy = sum(YTest == testLabels)/numel(testLabels)
%%
% Load Truth Table from Labeler App in Matlab
TruthTable = objectDetectorTrainingData(gTruth);
summary(TruthTable)
%%
%Train Fast R-CNN for New Sign Detection
%Changes categories from 10 to 3 -> PL,PR,Background
doTraining = true;
if doTraining
   % training options
   options = trainingOptions('sgdm', ...
      'MiniBatchSize', 30, ...
      'InitialLearnRate', 1e-3, ...
      'LearnRateSchedule', 'piecewise', ...
      'LearnRateDropFactor', 0.1, ...
      'LearnRateDropPeriod', 100, ...
      'MaxEpochs', 10, ...
      'VerboseFrequency', 200);
   % Train an R-CNN object detector. This will take several minutes.
   rcnn = trainFasterRCNNObjectDetector(TruthTable, cifar10Net,
options, ...
   'NegativeOverlapRange', [0 .3], 'PositiveOverlapRange',[.6 1])
else
   % pre-trained network for the example.
   load('rcnnStopSigns.mat','rcnn')
end
%%
% Test R-CNN Detector - Random Image
boxColor = [];
ann = [];
testImage = imread('OML_7.png');
%imageSize = size(testImage);
```

```
[bboxes,score,label] = detect(rcnn,testImage,'SelectStrongest',true);
for i = 1:length(score)
    ann{i} = sprintf('%s: %f', label(i), score(i));
    %split annotations for red or yellow boxes
    if bboxes(i) > imageSize(1)*.4 && label(i) == "PL"
        boxColor = [boxColor;([255 0 0])];

    elseif bboxes(i) < imageSize(1)*.6 && label(i) == "PR"
        boxColor = [boxColor;([255 0 0])];
    else
        boxColor = [boxColor;([255 255 0])];
    end
end


outputImage = insertObjectAnnotation(testImage, 'rectangle', ...
    bboxes, ann,...
    'LineWidth',5,...
    'Color',boxColor);
%Annotations format: [x,y,width,height]
%   x=0 & y=0 is the upper left corner of the image
figure
imshow(outputImage)
%%
% Setup Video Reader and Display
videoFReader = vision.VideoFileReader('TestVideo480.mp4',...
    'ImageColorSpace','RGB');
videoFrame = videoFReader(); %Get first frame of video file
%Use H.265 10-bit(x265) codec in Handbrake
%Test on video play back frame.
%If codec is good, image will have color and not distorted.
imshow(videoFrame)
videoPlayer = vision.DeployableVideoPlayer;
%%
% Setup Video Writer
videoFWriter =
vision.VideoFileWriter('FinalProject6.avi','FrameRate',...
    videoFReader.info.VideoFrameRate);
videoFWriter.VideoCompressor='DV Video Encoder';
%%
% Test Fast RCNN on video file
while ~isDone(videoFReader)
    image = step(videoFReader);
    im = im2uint8(image);
    [bboxes,score,label] = detect(rcnn,im,...
        'SelectStrongest',true,...
        'NumStrongestRegions',1000);
    %[bboxes,score] = detect(rcnn,im);
    if isempty(score) == 1
        label = 'NULL';
        score = 0;
        bboxes = [35 35 100 100];
    end
    ann = [];
    boxColor = [];
    for i = 1:length(score)
        ann{i} = sprintf('%s: %f', label(i), score(i));
        if label(i) == "PL"
            boxColor = [boxColor;([255 0 0])];
        else
            boxColor = [boxColor;([255 255 0])];
        end
    end
    outputImage = insertObjectAnnotation(image, 'rectangle',...
    bboxes, ann);
    step(videoPlayer, outputImage);
    step(videoFWriter,outputImage);
end
release(videoFReader)
release(videoPlayer)
release(videoFWriter)
```

B. Training Diary

```
% Download Image Data - CIFAR-10
cifar10Data = tempdir;
url = 'https://www.cs.toronto.edu/~kriz/cifar-10-matlab.tar.gz';
helperCIFAR10Data.download(url,cifar10Data);
%%
% Load CIFAR-10 Training
[trainingImages, trainingLabels, testImages, testLabels]=...
   helperCIFAR10Data.load(cifar10Data);
%%
% Data Sample
size(trainingImages)

ans =

        32          32           3       50000

numImageCategories = 10;
categories(trainingLabels)

ans =

  10×1 <a href="matlab:helpPopup cell" style="font-weight:bold">cell</a>
array

    {'airplane'  }
    {'automobile'}
    {'bird'      }
    {'cat'       }
    {'deer'      }
    {'dog'       }
    {'frog'      }
    {'horse'     }
    {'ship'      }
    {'truck'     }

figure
thumbnails = trainingImages(:,:,:, 1:100);
montage(thumbnails)
% Create Image Layers
[height, width,numChannels,~] = size(trainingImages);
imageSize = [height width numChannels];
inputLayer = imageInputLayer(imageSize)

inputLayer =

  <a href="matlab:helpPopup nnet.cnn.layer.ImageInputLayer" style="font-
weight:bold">ImageInputLayer</a> with properties:

            Name: ''
       InputSize: [32 32 3]

   Hyperparameters
    DataAugmentation: 'none'
       Normalization: 'zerocenter'

% Convolution Layer Parameters
filterSize = [5 5];
numFilters = 32;
middleLayers = [
    convolution2dLayer(filterSize, numFilters, 'Padding', 2)
    reluLayer()
```

```
    maxPooling2dLayer(3, 'Stride', 2)
    convolution2dLayer(filterSize, numFilters, 'Padding', 2)
    reluLayer()
    maxPooling2dLayer(3, 'Stride',2)
    convolution2dLayer(filterSize, 2 * numFilters, 'Padding', 2)
    reluLayer()
    maxPooling2dLayer(3, 'Stride',2)
]

middleLayers =

  9x1 <a href="matlab:helpPopup nnet.cnn.layer.Layer" style="font-
weight:bold">Layer</a> array with layers:

     1   ''   Convolution   32 5x5 convolutions with stride [1  1] and padding [2
2  2  2]
     2   ''   ReLU          ReLU
     3   ''   Max Pooling   3x3 max pooling with stride [2  2] and padding [0  0
0  0]
     4   ''   Convolution   32 5x5 convolutions with stride [1  1] and padding [2
2  2  2]
     5   ''   ReLU          ReLU
     6   ''   Max Pooling   3x3 max pooling with stride [2  2] and padding [0  0
0  0]
     7   ''   Convolution   64 5x5 convolutions with stride [1  1] and padding [2
2  2  2]
     8   ''   ReLU          ReLU
     9   ''   Max Pooling   3x3 max pooling with stride [2  2] and padding [0  0
0  0]
% CNN Final Layer
finalLayers = [
    fullyConnectedLayer(64)
    reluLayer
    fullyConnectedLayer(numImageCategories)
    softmaxLayer
    classificationLayer
]

finalLayers =

  5x1 <a href="matlab:helpPopup nnet.cnn.layer.Layer" style="font-
weight:bold">Layer</a> array with layers:

     1   ''   Fully Connected         64 fully connected layer
     2   ''   ReLU                    ReLU
     3   ''   Fully Connected         10 fully connected layer
     4   ''   Softmax                 softmax
     5   ''   Classification Output   crossentropyex
% Combine Input, Middle, and Final Layers
layers = [
    inputLayer
    middleLayers
    finalLayers
    ]

layers =

  15x1 <a href="matlab:helpPopup nnet.cnn.layer.Layer" style="font-
weight:bold">Layer</a> array with layers:

     1   ''   Image Input             32x32x3 images with 'zerocenter' normalization
     2   ''   Convolution             32 5x5 convolutions with stride [1  1] and
padding [2  2  2  2]
     3   ''   ReLU                    ReLU
     4   ''   Max Pooling             3x3 max pooling with stride [2  2] and padding
[0  0  0  0]
     5   ''   Convolution             32 5x5 convolutions with stride [1  1] and
padding [2  2  2  2]
     6   ''   ReLU                    ReLU
     7   ''   Max Pooling             3x3 max pooling with stride [2  2] and padding
[0  0  0  0]
     8   ''   Convolution             64 5x5 convolutions with stride [1  1] and
padding [2  2  2  2]
     9   ''   ReLU                    ReLU
    10   ''   Max Pooling             3x3 max pooling with stride [2  2] and padding
[0  0  0  0]
    11   ''   Fully Connected         64 fully connected layer
    12   ''   ReLU                    ReLU
    13   ''   Fully Connected         10 fully connected layer
    14   ''   Softmax                 softmax
    15   ''   Classification Output   crossentropyex
% Initialize Convolution Layer Weights
layers(2).Weights = 0.0001 * randn([filterSize numChannels numFilters]);
% Train CNN - Set Network Training Options
opts = trainingOptions('sgdm', ...
    'Momentum', 0.9, ...
    'InitialLearnRate', 0.001, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropFactor', 0.1, ...
    'LearnRateDropPeriod', 8, ...
    'L2Regularization', 0.004, ...
    'MaxEpochs', 40, ...
    'MiniBatchSize', 128, ...
    'Verbose', true,...
    'Plots','training-progress');
% Train CNN - Execute Training (20-30 minutes)
%true to train network (20-30 minutes)
%false to load pre-trained network
doTraining = true;

if doTraining
    % Train a network.
    cifar10Net = trainNetwork(trainingImages, trainingLabels, layers, opts);
else
    % Load pre-trained detector for the example.
    load('rcnnStopSigns.mat','cifar10Net')
end
Training on single GPU.
Initializing image normalization.
|======================================================================
==================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base
Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate      |
|======================================================================
==================================|
|       1 |           1 |       00:00:02 |        8.59% |       2.3026 |         0.0010 |
|       1 |          50 |       00:00:04 |       13.28% |       2.3022 |         0.0010 |
|       1 |         100 |       00:00:05 |        9.38% |       2.3027 |         0.0010 |
|       1 |         150 |       00:00:07 |       11.72% |       2.3016 |         0.0010 |
|       1 |         200 |       00:00:08 |       14.84% |       2.2984 |         0.0010 |
|       1 |         250 |       00:00:10 |       12.50% |       2.2823 |         0.0010 |
|       1 |         300 |       00:00:11 |       15.63% |       2.2488 |         0.0010 |
|       1 |         350 |       00:00:13 |       20.31% |       2.2037 |         0.0010 |
|       2 |         400 |       00:00:14 |       14.84% |       2.1734 |         0.0010 |
|       2 |         450 |       00:00:16 |       28.91% |       1.9965 |         0.0010 |
|       2 |         500 |       00:00:17 |       32.81% |       1.8913 |         0.0010 |
|       2 |         550 |       00:00:19 |       28.91% |       1.8289 |         0.0010 |
|       2 |         600 |       00:00:20 |       34.38% |       1.8000 |         0.0010 |
|       2 |         650 |       00:00:22 |       35.16% |       1.6815 |         0.0010 |
|       2 |         700 |       00:00:23 |       42.97% |       1.6240 |         0.0010 |
|       2 |         750 |       00:00:24 |       45.31% |       1.5761 |         0.0010 |
|       3 |         800 |       00:00:26 |       35.94% |       1.6299 |         0.0010 |
|       3 |         850 |       00:00:27 |       46.09% |       1.3647 |         0.0010 |
|       3 |         900 |       00:00:29 |       42.19% |       1.6671 |         0.0010 |
|       3 |         950 |       00:00:30 |       45.31% |       1.5126 |         0.0010 |
|       3 |        1000 |       00:00:32 |       42.97% |       1.5908 |         0.0010 |
|       3 |        1050 |       00:00:33 |       58.59% |       1.3326 |         0.0010 |
|       3 |        1100 |       00:00:35 |       57.03% |       1.1875 |         0.0010 |
|       3 |        1150 |       00:00:36 |       46.09% |       1.5562 |         0.0010 |
|       4 |        1200 |       00:00:38 |       52.34% |       1.4292 |         0.0010 |
|       4 |        1250 |       00:00:39 |       54.69% |       1.1906 |         0.0010 |
|       4 |        1300 |       00:00:41 |       48.44% |       1.3944 |         0.0010 |
```

4	1350	00:00:42	58.59%	1.1588	0.0010
4	1400	00:00:44	53.13%	1.2898	0.0010
4	1450	00:00:45	55.47%	1.2385	0.0010
4	1500	00:00:47	54.69%	1.3089	0.0010
4	1550	00:00:48	53.91%	1.2493	0.0010
5	1600	00:00:50	53.91%	1.2567	0.0010
5	1650	00:00:51	60.94%	1.1291	0.0010
5	1700	00:00:53	59.38%	1.1653	0.0010
5	1750	00:00:54	60.16%	1.1021	0.0010
5	1800	00:00:56	59.38%	1.0651	0.0010
5	1850	00:00:57	61.72%	1.1191	0.0010
5	1900	00:00:58	61.72%	1.1744	0.0010
5	1950	00:01:00	50.78%	1.1229	0.0010
6	2000	00:01:01	57.03%	1.2667	0.0010
6	2050	00:01:03	70.31%	0.9323	0.0010
6	2100	00:01:04	67.97%	0.9897	0.0010
6	2150	00:01:06	63.28%	1.0445	0.0010
6	2200	00:01:07	62.50%	1.0425	0.0010
6	2250	00:01:09	67.97%	0.8547	0.0010
6	2300	00:01:10	68.75%	0.9220	0.0010
7	2350	00:01:12	60.16%	1.1943	0.0010
7	2400	00:01:13	71.09%	0.8562	0.0010
7	2450	00:01:15	70.31%	1.0289	0.0010
7	2500	00:01:16	62.50%	1.0202	0.0010
7	2550	00:01:18	71.88%	0.8999	0.0010
7	2600	00:01:19	77.34%	0.8057	0.0010
7	2650	00:01:21	66.41%	0.9234	0.0010
7	2700	00:01:22	75.78%	0.7688	0.0010
8	2750	00:01:24	64.84%	0.9759	0.0010
8	2800	00:01:25	83.59%	0.5923	0.0010
8	2850	00:01:27	67.97%	0.9287	0.0010
8	2900	00:01:28	71.88%	0.8105	0.0010
8	2950	00:01:30	71.88%	0.8452	0.0010
8	3000	00:01:31	75.00%	0.7664	0.0010
8	3050	00:01:33	77.34%	0.6321	0.0010
8	3100	00:01:34	67.19%	0.9249	0.0010
9	3150	00:01:36	74.22%	0.7488	0.0001
9	3200	00:01:37	80.47%	0.5893	0.0001
9	3250	00:01:39	68.75%	0.7115	0.0001
9	3300	00:01:40	75.00%	0.7304	0.0001
9	3350	00:01:42	76.56%	0.6372	0.0001
9	3400	00:01:43	81.25%	0.5885	0.0001
9	3450	00:01:45	73.44%	0.6585	0.0001
9	3500	00:01:46	84.38%	0.5451	0.0001
10	3550	00:01:48	81.25%	0.6507	0.0001
10	3600	00:01:49	78.13%	0.6368	0.0001
10	3650	00:01:51	75.78%	0.6628	0.0001
10	3700	00:01:53	78.91%	0.6306	0.0001
10	3750	00:01:54	75.78%	0.6214	0.0001
10	3800	00:01:56	78.91%	0.6741	0.0001
10	3850	00:01:57	75.78%	0.7132	0.0001
10	3900	00:01:59	80.47%	0.6711	0.0001
11	3950	00:02:00	73.44%	0.8565	0.0001
11	4000	00:02:02	83.59%	0.4927	0.0001
11	4050	00:02:03	73.44%	0.6199	0.0001
11	4100	00:02:04	76.56%	0.6750	0.0001
11	4150	00:02:06	78.91%	0.6084	0.0001
11	4200	00:02:07	78.91%	0.5906	0.0001
11	4250	00:02:09	82.81%	0.5610	0.0001
12	4300	00:02:10	75.78%	0.7129	0.0001
12	4350	00:02:12	77.34%	0.5900	0.0001
12	4400	00:02:13	75.78%	0.7758	0.0001
12	4450	00:02:15	75.00%	0.6870	0.0001
12	4500	00:02:16	81.25%	0.6836	0.0001
12	4550	00:02:18	86.72%	0.4857	0.0001
12	4600	00:02:19	78.13%	0.5626	0.0001
12	4650	00:02:21	82.81%	0.4913	0.0001
13	4700	00:02:22	81.25%	0.6113	0.0001
13	4750	00:02:24	89.06%	0.3677	0.0001
13	4800	00:02:25	74.22%	0.7198	0.0001
13	4850	00:02:26	82.03%	0.5783	0.0001
13	4900	00:02:28	80.47%	0.6336	0.0001
13	4950	00:02:29	83.59%	0.4866	0.0001
13	5000	00:02:31	85.16%	0.4948	0.0001
13	5050	00:02:32	71.09%	0.7271	0.0001
14	5100	00:02:34	79.69%	0.6298	0.0001
14	5150	00:02:35	84.38%	0.5108	0.0001
14	5200	00:02:37	76.56%	0.5905	0.0001
14	5250	00:02:38	76.56%	0.6581	0.0001
14	5300	00:02:40	79.69%	0.5905	0.0001
14	5350	00:02:41	82.03%	0.5361	0.0001
14	5400	00:02:43	76.56%	0.6037	0.0001
14	5450	00:02:44	84.38%	0.5137	0.0001
15	5500	00:02:45	82.03%	0.5779	0.0001
15	5550	00:02:47	81.25%	0.6002	0.0001
15	5600	00:02:48	77.34%	0.5835	0.0001
15	5650	00:02:50	81.25%	0.5847	0.0001
15	5700	00:02:51	78.91%	0.5531	0.0001
15	5750	00:02:53	78.13%	0.6294	0.0001
15	5800	00:02:54	75.00%	0.6485	0.0001
15	5850	00:02:56	79.69%	0.6300	0.0001
16	5900	00:02:57	75.78%	0.7963	0.0001
16	5950	00:02:59	87.50%	0.4221	0.0001
16	6000	00:03:00	77.34%	0.5736	0.0001
16	6050	00:03:02	78.91%	0.5789	0.0001
16	6100	00:03:03	78.13%	0.5579	0.0001
16	6150	00:03:05	80.47%	0.5509	0.0001
16	6200	00:03:06	82.03%	0.5233	0.0001
17	6250	00:03:08	78.13%	0.6621	1.0000e-05
17	6300	00:03:09	80.47%	0.5373	1.0000e-05
17	6350	00:03:10	76.56%	0.7321	1.0000e-05
17	6400	00:03:12	78.13%	0.6068	1.0000e-05
17	6450	00:03:13	83.59%	0.5761	1.0000e-05
17	6500	00:03:15	88.28%	0.3826	1.0000e-05
17	6550	00:03:16	82.03%	0.5050	1.0000e-05
17	6600	00:03:18	84.38%	0.4362	1.0000e-05
18	6650	00:03:19	80.47%	0.5558	1.0000e-05
18	6700	00:03:21	91.41%	0.3021	1.0000e-05
18	6750	00:03:22	75.00%	0.6814	1.0000e-05
18	6800	00:03:24	83.59%	0.5040	1.0000e-05
18	6850	00:03:25	83.59%	0.5854	1.0000e-05
18	6900	00:03:27	85.16%	0.4479	1.0000e-05
18	6950	00:03:28	85.94%	0.4497	1.0000e-05
18	7000	00:03:30	71.88%	0.6606	1.0000e-05
19	7050	00:03:31	78.91%	0.5740	1.0000e-05
19	7100	00:03:33	85.94%	0.4923	1.0000e-05
19	7150	00:03:34	78.91%	0.5373	1.0000e-05
19	7200	00:03:36	76.56%	0.5872	1.0000e-05
19	7250	00:03:37	80.47%	0.5468	1.0000e-05
19	7300	00:03:38	83.59%	0.5032	1.0000e-05
19	7350	00:03:40	81.25%	0.5607	1.0000e-05
19	7400	00:03:41	84.38%	0.4532	1.0000e-05
20	7450	00:03:43	84.38%	0.5192	1.0000e-05
20	7500	00:03:44	81.25%	0.5741	1.0000e-05
20	7550	00:03:46	78.13%	0.5378	1.0000e-05
20	7600	00:03:47	82.03%	0.5756	1.0000e-05
20	7650	00:03:49	81.25%	0.5192	1.0000e-05
20	7700	00:03:50	80.47%	0.5749	1.0000e-05
20	7750	00:03:52	80.47%	0.5883	1.0000e-05
20	7800	00:03:53	79.69%	0.5826	1.0000e-05
21	7850	00:03:55	75.78%	0.7591	1.0000e-05
21	7900	00:03:56	88.28%	0.4177	1.0000e-05
21	7950	00:03:58	78.91%	0.5339	1.0000e-05
21	8000	00:03:59	83.59%	0.5292	1.0000e-05
21	8050	00:04:01	80.47%	0.5144	1.0000e-05
21	8100	00:04:02	78.13%	0.5556	1.0000e-05
21	8150	00:04:04	84.38%	0.4762	1.0000e-05
22	8200	00:04:05	78.91%	0.6339	1.0000e-05
22	8250	00:04:07	80.47%	0.5144	1.0000e-05
22	8300	00:04:08	78.91%	0.7137	1.0000e-05
22	8350	00:04:10	82.03%	0.5898	1.0000e-05
22	8400	00:04:11	83.59%	0.5614	1.0000e-05
22	8450	00:04:13	89.06%	0.3778	1.0000e-05
22	8500	00:04:14	80.47%	0.5019	1.0000e-05

22	8550	00:04:15	85.16%	0.4360	1.0000e-05
23	8600	00:04:17	80.47%	0.5453	1.0000e-05
23	8650	00:04:18	92.19%	0.2949	1.0000e-05
23	8700	00:04:20	75.00%	0.6696	1.0000e-05
23	8750	00:04:21	83.59%	0.4957	1.0000e-05
23	8800	00:04:23	82.81%	0.5824	1.0000e-05
23	8850	00:04:24	85.94%	0.4442	1.0000e-05
23	8900	00:04:26	85.94%	0.4475	1.0000e-05
23	8950	00:04:27	71.88%	0.6592	1.0000e-05
24	9000	00:04:29	78.91%	0.5647	1.0000e-05
24	9050	00:04:30	85.94%	0.4874	1.0000e-05
24	9100	00:04:32	78.91%	0.5283	1.0000e-05
24	9150	00:04:33	76.56%	0.5814	1.0000e-05
24	9200	00:04:35	80.47%	0.5431	1.0000e-05
24	9250	00:04:36	83.59%	0.4995	1.0000e-05
24	9300	00:04:38	81.25%	0.5547	1.0000e-05
24	9350	00:04:39	84.38%	0.4494	1.0000e-05
25	9400	00:04:41	84.38%	0.5134	1.0000e-06
25	9450	00:04:42	81.25%	0.5691	1.0000e-06
25	9500	00:04:44	78.13%	0.5271	1.0000e-06
25	9550	00:04:45	83.59%	0.5633	1.0000e-06
25	9600	00:04:47	80.47%	0.5096	1.0000e-06
25	9650	00:04:48	80.47%	0.5718	1.0000e-06
25	9700	00:04:50	80.47%	0.5791	1.0000e-06
25	9750	00:04:51	81.25%	0.5726	1.0000e-06
26	9800	00:04:53	77.34%	0.7576	1.0000e-06
26	9850	00:04:54	89.06%	0.4142	1.0000e-06
26	9900	00:04:56	79.69%	0.5295	1.0000e-06
26	9950	00:04:57	83.59%	0.5222	1.0000e-06
26	10000	00:04:59	81.25%	0.5102	1.0000e-06
26	10050	00:05:00	78.91%	0.5430	1.0000e-06
26	10100	00:05:02	85.16%	0.4682	1.0000e-06
27	10150	00:05:03	77.34%	0.6180	1.0000e-06
27	10200	00:05:05	81.25%	0.5100	1.0000e-06
27	10250	00:05:06	77.34%	0.7139	1.0000e-06
27	10300	00:05:08	82.03%	0.5853	1.0000e-06
27	10350	00:05:09	84.38%	0.5573	1.0000e-06
27	10400	00:05:11	89.06%	0.3756	1.0000e-06
27	10450	00:05:12	82.81%	0.4961	1.0000e-06
27	10500	00:05:14	85.16%	0.4346	1.0000e-06
28	10550	00:05:15	82.03%	0.5388	1.0000e-06
28	10600	00:05:17	92.97%	0.2907	1.0000e-06
28	10650	00:05:18	74.22%	0.6657	1.0000e-06
28	10700	00:05:20	83.59%	0.4941	1.0000e-06
28	10750	00:05:21	84.38%	0.5785	1.0000e-06
28	10800	00:05:22	86.72%	0.4381	1.0000e-06
28	10850	00:05:24	85.94%	0.4438	1.0000e-06
28	10900	00:05:25	72.66%	0.6476	1.0000e-06
29	10950	00:05:27	78.91%	0.5614	1.0000e-06
29	11000	00:05:28	85.16%	0.4844	1.0000e-06
29	11050	00:05:30	80.47%	0.5226	1.0000e-06
29	11100	00:05:31	77.34%	0.5768	1.0000e-06
29	11150	00:05:33	80.47%	0.5436	1.0000e-06
29	11200	00:05:34	82.03%	0.4963	1.0000e-06
29	11250	00:05:36	79.69%	0.5539	1.0000e-06
29	11300	00:05:37	84.38%	0.4494	1.0000e-06
30	11350	00:05:39	84.38%	0.5085	1.0000e-06
30	11400	00:05:40	82.03%	0.5648	1.0000e-06
30	11450	00:05:42	78.13%	0.5283	1.0000e-06
30	11500	00:05:43	82.81%	0.5641	1.0000e-06
30	11550	00:05:45	81.25%	0.5083	1.0000e-06
30	11600	00:05:46	81.25%	0.5694	1.0000e-06
30	11650	00:05:48	81.25%	0.5769	1.0000e-06
30	11700	00:05:49	81.25%	0.5725	1.0000e-06
31	11750	00:05:51	77.34%	0.7578	1.0000e-06
31	11800	00:05:52	89.84%	0.4127	1.0000e-06
31	11850	00:05:54	79.69%	0.5286	1.0000e-06
31	11900	00:05:55	83.59%	0.5210	1.0000e-06
31	11950	00:05:57	82.03%	0.5091	1.0000e-06
31	12000	00:05:58	78.91%	0.5426	1.0000e-06
31	12050	00:06:00	84.38%	0.4682	1.0000e-06
32	12100	00:06:01	77.34%	0.6172	1.0000e-06
32	12150	00:06:03	81.25%	0.5092	1.0000e-06
32	12200	00:06:04	77.34%	0.7123	1.0000e-06
32	12250	00:06:06	82.03%	0.5843	1.0000e-06
32	12300	00:06:07	84.38%	0.5559	1.0000e-06
32	12350	00:06:09	89.06%	0.3759	1.0000e-06
32	12400	00:06:10	82.81%	0.4961	1.0000e-06
32	12450	00:06:12	85.16%	0.4344	1.0000e-06
33	12500	00:06:13	82.03%	0.5381	1.0000e-07
33	12550	00:06:15	92.97%	0.2906	1.0000e-07
33	12600	00:06:16	74.22%	0.6650	1.0000e-07
33	12650	00:06:18	83.59%	0.4940	1.0000e-07
33	12700	00:06:19	84.38%	0.5774	1.0000e-07
33	12750	00:06:21	87.50%	0.4368	1.0000e-07
33	12800	00:06:22	86.72%	0.4435	1.0000e-07
33	12850	00:06:24	72.66%	0.6474	1.0000e-07
34	12900	00:06:25	78.91%	0.5609	1.0000e-07
34	12950	00:06:27	85.16%	0.4836	1.0000e-07
34	13000	00:06:28	81.25%	0.5232	1.0000e-07
34	13050	00:06:30	77.34%	0.5763	1.0000e-07
34	13100	00:06:31	80.47%	0.5428	1.0000e-07
34	13150	00:06:33	82.03%	0.4954	1.0000e-07
34	13200	00:06:34	79.69%	0.5539	1.0000e-07
34	13250	00:06:36	84.38%	0.4481	1.0000e-07
35	13300	00:06:37	84.38%	0.5081	1.0000e-07
35	13350	00:06:39	82.03%	0.5646	1.0000e-07
35	13400	00:06:40	78.91%	0.5280	1.0000e-07
35	13450	00:06:42	82.81%	0.5625	1.0000e-07
35	13500	00:06:43	81.25%	0.5078	1.0000e-07
35	13550	00:06:45	81.25%	0.5692	1.0000e-07
35	13600	00:06:46	81.25%	0.5761	1.0000e-07
35	13650	00:06:48	81.25%	0.5721	1.0000e-07
36	13700	00:06:49	77.34%	0.7580	1.0000e-07
36	13750	00:06:51	89.84%	0.4122	1.0000e-07
36	13800	00:06:52	79.69%	0.5289	1.0000e-07
36	13850	00:06:54	83.59%	0.5209	1.0000e-07
36	13900	00:06:55	81.25%	0.5086	1.0000e-07
36	13950	00:06:57	78.91%	0.5426	1.0000e-07
36	14000	00:06:58	85.16%	0.4671	1.0000e-07
37	14050	00:07:00	77.34%	0.6167	1.0000e-07
37	14100	00:07:01	81.25%	0.5093	1.0000e-07
37	14150	00:07:03	77.34%	0.7123	1.0000e-07
37	14200	00:07:04	81.25%	0.5856	1.0000e-07
37	14250	00:07:06	84.38%	0.5560	1.0000e-07
37	14300	00:07:07	89.06%	0.3759	1.0000e-07
37	14350	00:07:09	82.81%	0.4957	1.0000e-07
37	14400	00:07:10	85.16%	0.4340	1.0000e-07
38	14450	00:07:12	82.03%	0.5384	1.0000e-07
38	14500	00:07:13	92.97%	0.2904	1.0000e-07
38	14550	00:07:15	74.22%	0.6649	1.0000e-07
38	14600	00:07:16	83.59%	0.4935	1.0000e-07
38	14650	00:07:18	84.38%	0.5776	1.0000e-07
38	14700	00:07:19	87.50%	0.4371	1.0000e-07
38	14750	00:07:21	86.72%	0.4435	1.0000e-07
38	14800	00:07:22	72.66%	0.6471	1.0000e-07
39	14850	00:07:24	78.91%	0.5606	1.0000e-07
39	14900	00:07:25	85.16%	0.4836	1.0000e-07
39	14950	00:07:27	81.25%	0.5230	1.0000e-07
39	15000	00:07:28	77.34%	0.5762	1.0000e-07
39	15050	00:07:30	80.47%	0.5429	1.0000e-07
39	15100	00:07:31	82.03%	0.4955	1.0000e-07
39	15150	00:07:33	79.69%	0.5538	1.0000e-07
39	15200	00:07:34	84.38%	0.4483	1.0000e-07
40	15250	00:07:35	84.38%	0.5079	1.0000e-07
40	15300	00:07:37	82.03%	0.5644	1.0000e-07
40	15350	00:07:38	78.91%	0.5280	1.0000e-07
40	15400	00:07:40	82.81%	0.5626	1.0000e-07
40	15450	00:07:41	81.25%	0.5077	1.0000e-07
40	15500	00:07:43	81.25%	0.5691	1.0000e-07
40	15550	00:07:44	81.25%	0.5761	1.0000e-07
40	15600	00:07:46	81.25%	0.5721	1.0000e-07

|==|

```
%for video training
%trainingData = objectDetectorTrainingData(TruthTable);
% Validate Training - Learned Edges
w = cifar10Net.Layers(2).Weights;
w = rescale(w);
figure
montage(w)
% Validate Training - Test Set
YTest = classify(cifar10Net, testImages);
accuracy = sum(YTest == testLabels)/numel(testLabels)

accuracy =

    0.7419

{Error using
driving.internal.videoLabeler.tool.VideoLabelingTool.validateAndProcessLoa
dedSessionWithImageSequence
Undefined variable "this" or class "this.getGroupName".
}
[Warning: Error occurred while evaluating listener callback.]

gTruth =

  <a href="matlab:helpPopup groundTruth" style="font-
weight:bold">groundTruth</a> with properties:

          DataSource: [1×1 groundTruthDataSource]
    LabelDefinitions: [2×3 table]
           LabelData: [508×2 timetable]

TruthTable = objectDetectorTrainingData(gTruth);

Write images extracted for training to folder:
    C:\Users\RA_Desktop\OneDrive\Fall 2018\Final Project\V3_Final_Project

Writing 74 images extracted from TestVideo.mp4...Completed.
summary(TruthTable)

Description:  This was created using the Ground Truth Labeler app on 25-
Nov-2018.

Variables:

    <strong>imageFilename</strong>: 74×1 cell array of character vectors

    <strong>PL</strong>: 74×1 cell

    <strong>PR</strong>: 74×1 cell

%Train Fast R-CNN for New Sign Detection
%Changes categories from 10 to 3 -> PL,PR,Background
doTraining = true;
if doTraining
    % training options
    options = trainingOptions('sgdm', ...
        'MiniBatchSize', 4, ...
        'InitialLearnRate', 1e-3, ...
        'LearnRateSchedule', 'piecewise', ...
        'LearnRateDropFactor', 0.1, ...
        'LearnRateDropPeriod', 100, ...
        'MaxEpochs', 10, ...
        'VerboseFrequency', 200);
    % Train an R-CNN object detector. This will take several minutes.
    rcnn = trainFasterRCNNObjectDetector(TruthTable3, cifar10Net, options,
...
    'NegativeOverlapRange', [0 .3], 'PositiveOverlapRange',[.6 1])
else
    % pre-trained network for the example.
    load('rcnnStopSigns.mat','rcnn')
end
{Undefined function or variable 'TruthTable3'.
}
%Train Fast R-CNN for New Sign Detection
%Changes categories from 10 to 3 -> PL,PR,Background
doTraining = true;
if doTraining
    % training options
    options = trainingOptions('sgdm', ...
        'MiniBatchSize', 4, ...
        'InitialLearnRate', 1e-3, ...
        'LearnRateSchedule', 'piecewise', ...
        'LearnRateDropFactor', 0.1, ...
        'LearnRateDropPeriod', 100, ...
        'MaxEpochs', 10, ...
        'VerboseFrequency', 200);
    % Train an R-CNN object detector. This will take several minutes.
    rcnn = trainFasterRCNNObjectDetector(TruthTable, cifar10Net, options, ...
    'NegativeOverlapRange', [0 .3], 'PositiveOverlapRange',[.6 1])
else
    % pre-trained network for the example.
    load('rcnnStopSigns.mat','rcnn')
end
*************************************************************
***********
Training a Faster R-CNN Object Detector for the following object classes:

* PL
* PR

Step 1 of 4: Training a Region Proposal Network (RPN).
Training on single GPU.
|======================================================
================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base
Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     RMSE     |     Rate      |
|======================================================
================================|
|       1 |           1 |       00:00:00 |       50.00% |         0.89 |        0.0010 |
|       3 |         200 |       00:01:22 |      100.00% |         1.54 |        0.0010 |
|       6 |         400 |       00:02:45 |      100.00% |         1.13 |        0.0010 |
|       9 |         600 |       00:04:07 |      100.00% |         1.01 |        0.0010 |
|      10 |         740 |       00:05:05 |       75.00% |         1.10 |        0.0010 |
|======================================================
================================|

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
*************************************************************
*****
Training a Fast R-CNN Object Detector for the following object classes:

* PL
* PR

--> Extracting region proposals from 74 training images...done.

Training on single GPU.
|======================================================
================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base
Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     RMSE     |     Rate      |
|======================================================
================================|
|       1 |           1 |       00:00:00 |        0.00% |         0.80 |        0.0010 |
|       3 |         200 |       00:00:58 |       75.00% |         0.62 |        0.0010 |
|       6 |         400 |       00:01:57 |       75.00% |         0.58 |        0.0010 |
|       9 |         600 |       00:02:55 |      100.00% |         0.71 |        0.0010 |
|      10 |         720 |       00:03:30 |       75.00% |         0.43 |        0.0010 |
|======================================================
================================|

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
```

```
Starting parallel pool (parpool) using the 'local' profile ...
connected to 4 workers.
Training on single GPU.
|=======================================================
=================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   | Accuracy   | RMSE       | Rate |
|=======================================================
=================================|
|     1 |         1 |     00:00:00 |     75.00% |       0.91 |      0.0010 |
|     3 |       200 |     00:01:00 |    100.00% |       0.85 |      0.0010 |
|     6 |       400 |     00:02:00 |     50.00% |       1.37 |      0.0010 |
|     9 |       600 |     00:03:01 |     75.00% |       0.79 |      0.0010 |
|    10 |       740 |     00:03:43 |     75.00% |       1.51 |      0.0010 |
|=======================================================
=================================|

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
*************************************************************
*****
Training a Fast R-CNN Object Detector for the following object classes:

* PL
* PR

--> Extracting region proposals from 74 training images...done.

Training on single GPU.
|=======================================================
=================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   | Accuracy   | RMSE       | Rate |
|=======================================================
=================================|
|     1 |         1 |     00:00:00 |     75.00% |       0.85 |      0.0010 |
|     4 |       200 |     00:00:38 |    100.00% |       0.47 |      0.0010 |
|     7 |       400 |     00:01:17 |     75.00% |       0.69 |      0.0010 |
|    10 |       600 |     00:01:55 |    100.00% |       0.61 |      0.0010 |
|    10 |       630 |     00:02:01 |    100.00% |       1.00 |      0.0010 |
|=======================================================
=================================|

Finished training Faster R-CNN object detector.


rcnn =

  <a href="matlab:helpPopup fasterRCNNObjectDetector" style="font-
weight:bold">fasterRCNNObjectDetector</a> with properties:

                ModelName: 'PL'
                  Network: [1×1 vision.cnn.FastRCNN]
   RegionProposalNetwork: [1×1 vision.cnn.RegionProposalNetwork]
             MinBoxSizes: [37 17]
         BoxPyramidScale: 2
     NumBoxPyramidLevels: 6
              ClassNames: {'PL'  'PR'  'Background'}
           MinObjectSize: [5 5]

IdleTimeout has been reached.
Parallel pool using the 'local' profile is shutting down.
IdleTimeout has been reached.
Parallel pool using the 'local' profile is shutting down.
% Setup Video Reader and Display
videoFReader = vision.VideoFileReader('TestVideo480.mp4',...
   'ImageColorSpace','RGB');
videoFrame = videoFReader(); %Get first frame of video file
%Use H.265 10-bit(x265) codec in Handbrake
%Test on video play back frame.
%If codec is good, image will have color and not distorted.
imshow(videoFrame)
videoPlayer = vision.DeployableVideoPlayer;
% Setup Video Writer
videoFWriter = vision.VideoFileWriter('FinalProject4.avi','FrameRate',...
   videoFReader.info.VideoFrameRate);
videoFWriter.VideoCompressor='DV Video Encoder';
% Test Fast RCNN on video file
while ~isDone(videoFReader)
   image = step(videoFReader);
   im = im2uint8(image);
   [bboxes,score,label] = detect(rcnn,im,...
      'SelectStrongest',true,...
      %'RatioType','Min'...
       %'RatioType','Min'...

{Error: Invalid expression. When calling a function or indexing a variable,
use parentheses. Otherwise, check for mismatched delimiters.
}
while ~isDone(videoFReader)
   image = step(videoFReader);
   im = im2uint8(image);
   [bboxes,score,label] = detect(rcnn,im,...
      'SelectStrongest',true,...
      'NumStrongestRegions',1000);
   %[bboxes,score] = detect(rcnn,im);
   if isempty(score) == 1
      label = 'NULL';
      score = 0;
      bboxes = [35 35 100 100];
   end
   ann = [];
   boxColor = [];
   for i = 1:length(score)
      ann{i} = sprintf('%s: %f', label(i), score(i));
      if label(i) == "PL"
         boxColor = [boxColor;([255 0 0])];
      else
         boxColor = [boxColor;([255 255 0])];
      end
   end
   outputImage = insertObjectAnnotation(image, 'rectangle', ...
   bboxes, ann...
   'LineWidth',5,...
    'LineWidth',5,...

{Error: Invalid expression. Check for missing multiplication operator, missing
or unbalanced delimiters, or other syntax error. To construct matrices, use
brackets instead of parentheses.
}
while ~isDone(videoFReader)
   image = step(videoFReader);
   im = im2uint8(image);
   [bboxes,score,label] = detect(rcnn,im,...
      'SelectStrongest',true,...
      'NumStrongestRegions',1000);
   %[bboxes,score] = detect(rcnn,im);
   if isempty(score) == 1
      label = 'NULL';
      score = 0;
      bboxes = [35 35 100 100];
   end
   ann = [];
   boxColor = [];
   for i = 1:length(score)
      ann{i} = sprintf('%s: %f', label(i), score(i));
      if label(i) == "PL"
         boxColor = [boxColor;([255 0 0])];
      else
         boxColor = [boxColor;([255 255 0])];
      end
   end
   outputImage = insertObjectAnnotation(image, 'rectangle', ...
```

```
    bboxes, ann...
    'LineWidth',5,...
     'LineWidth',5,...

{Error: Invalid expression. Check for missing multiplication operator, missing
or unbalanced delimiters, or other syntax error. To construct matrices, use
brackets instead of parentheses.
}
release(videoFReader);
release(videoPlayer);
% Test Fast RCNN on video file
while ~isDone(videoFReader)
    image = step(videoFReader);
    im = im2uint8(image);
    [bboxes,score,label] = detect(rcnn,im,...
        'SelectStrongest',true,...
        'NumStrongestRegions',1000);
    %[bboxes,score] = detect(rcnn,im);
    if isempty(score) == 1
        label = 'NULL';
        score = 0;
        bboxes = [35 35 100 100];
    end
    ann = [];
    boxColor = [];
    for i = 1:length(score)
        ann{i} = sprintf('%s: %f', label(i), score(i));
        if label(i) == "PL"
            boxColor = [boxColor;([255 0 0])];
        else
            boxColor = [boxColor;([255 255 0])];
        end
    end
    outputImage = insertObjectAnnotation(image, 'rectangle',...
    bboxes, ann...
    'LineWidth',5,...
     'LineWidth',5,...

{Error: Invalid expression. Check for missing multiplication operator, missing
or unbalanced delimiters, or other syntax error. To construct matrices, use
brackets instead of parentheses.
}
% Test Fast RCNN on video file
while ~isDone(videoFReader)
    image = step(videoFReader);
    im = im2uint8(image);
    [bboxes,score,label] = detect(rcnn,im,...
        'SelectStrongest',true,...
        'NumStrongestRegions',1000);
    %[bboxes,score] = detect(rcnn,im);
    if isempty(score) == 1
        label = 'NULL';
        score = 0;
        bboxes = [35 35 100 100];
    end
    ann = [];
    boxColor = [];
    for i = 1:length(score)
        ann{i} = sprintf('%s: %f', label(i), score(i));
        if label(i) == "PL"
            boxColor = [boxColor;([255 0 0])];
        else
            boxColor = [boxColor;([255 255 0])];
        end
    end
    outputImage = insertObjectAnnotation(image, 'rectangle',...
    bboxes, ann,'LineWidth',5,...
    'Color',boxColor);
    step(videoPlayer, outputImage);
    step(videoFWriter,outputImage);
end
release(videoFReader)
release(videoPlayer)
release(videoFWriter)
% Test Fast RCNN on video file
while ~isDone(videoFReader)
    image = step(videoFReader);
    im = im2uint8(image);
    [bboxes,score,label] = detect(rcnn,im,...
        'SelectStrongest',true,...
        'NumStrongestRegions',1000);
    %[bboxes,score] = detect(rcnn,im);
    if isempty(score) == 1
        label = 'NULL';
        score = 0;
        bboxes = [35 35 100 100];
    end
    ann = [];
    boxColor = [];
    for i = 1:length(score)
        ann{i} = sprintf('%s: %f', label(i), score(i));
        if label(i) == "PL"
            boxColor = [boxColor;([255 0 0])];
        else
            boxColor = [boxColor;([255 255 0])];
        end
    end
    outputImage = insertObjectAnnotation(image, 'rectangle',...
    bboxes, ann,...
    'Color',boxColor);
    step(videoPlayer, outputImage);
    step(videoFWriter,outputImage);
end
release(videoFReader)
release(videoPlayer)
release(videoFWriter)
%Train Fast R-CNN for New Sign Detection
%Changes categories from 10 to 3 -> PL,PR,Background
doTraining = true;
if doTraining
    % training options
    options = trainingOptions('sgdm', ...
        'MiniBatchSize', 30, ...
        'InitialLearnRate', 1e-3, ...
        'LearnRateSchedule', 'piecewise', ...
        'LearnRateDropFactor', 0.1, ...
        'LearnRateDropPeriod', 100, ...
        'MaxEpochs', 10, ...
        'VerboseFrequency', 200);
    % Train an R-CNN object detector. This will take several minutes.
    rcnn = trainFasterRCNNObjectDetector(TruthTable, cifar10Net, options, ...
    'NegativeOverlapRange', [0 .3], 'PositiveOverlapRange',[.6 1])
else
    % pre-trained network for the example.
    load('rcnnStopSigns.mat','rcnn')
end
Starting parallel pool (parpool) using the 'local' profile ...
connected to 4 workers.
*************************************************************
***********
Training a Faster R-CNN Object Detector for the following object classes:

* PL
* PR

Step 1 of 4: Training a Region Proposal Network (RPN).
Training on single GPU.
|=====================================================
================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   | Accuracy   | RMSE       | Rate     |
|=====================================================
================================|
```

```
|       1 |           1 |       00:00:00 |       60.00% |         1.08 |          0.0010 |
|       3 |         200 |       00:01:21 |       93.33% |         1.03 |          0.0010 |
|       6 |         400 |       00:02:44 |       83.33% |         0.81 |          0.0010 |
|       9 |         600 |       00:04:06 |       86.67% |         0.82 |          0.0010 |
|      10 |         740 |       00:05:04 |       83.33% |         0.82 |          0.0010 |
|=========================================================
=================================|

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
**************************************************************
*****
Training a Fast R-CNN Object Detector for the following object classes:

* PL
* PR

--> Extracting region proposals from 74 training images...done.

Training on single GPU.
|=========================================================
=================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base
Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     RMSE     |      Rate      |
|=========================================================
=================================|
|       1 |           1 |       00:00:00 |       13.33% |         0.63 |          0.0010 |
|       3 |         200 |       00:00:58 |       86.67% |         0.57 |          0.0010 |
|       6 |         400 |       00:01:56 |       93.33% |         0.65 |          0.0010 |
|       9 |         600 |       00:02:54 |       96.67% |         0.50 |          0.0010 |
|      10 |         730 |       00:03:33 |      100.00% |         0.58 |          0.0010 |
|=========================================================
=================================|

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Training on single GPU.
|=========================================================
=================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base
Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     RMSE     |      Rate      |
|=========================================================
=================================|
|       1 |           1 |       00:00:00 |       73.33% |         0.77 |          0.0010 |
|       3 |         200 |       00:01:01 |       96.67% |         0.89 |          0.0010 |
|       6 |         400 |       00:02:02 |       96.67% |         0.90 |          0.0010 |
|       9 |         600 |       00:03:04 |       93.33% |         0.99 |          0.0010 |
|      10 |         740 |       00:03:47 |      100.00% |         0.68 |          0.0010 |
|=========================================================
=================================|

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
**************************************************************
*****
Training a Fast R-CNN Object Detector for the following object classes:

* PL
* PR

--> Extracting region proposals from 74 training images...done.

Training on single GPU.
|=========================================================
=================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base
Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     RMSE     |      Rate      |
|=========================================================
=================================|
|       1 |           1 |       00:00:00 |       86.67% |         0.66 |          0.0010 |
|       3 |         200 |       00:00:39 |       96.67% |         0.52 |          0.0010 |
|       6 |         400 |       00:01:18 |      100.00% |         0.41 |          0.0010 |
|       9 |         600 |       00:01:57 |      100.00% |         0.34 |          0.0010 |
|      10 |         740 |       00:02:25 |       96.67% |         0.47 |          0.0010 |
|=========================================================
=================================|

Finished training Faster R-CNN object detector.


rcnn = 

  <a href="matlab:helpPopup fasterRCNNObjectDetector" style="font-
weight:bold">fasterRCNNObjectDetector</a> with properties:

                ModelName: 'PL'
                  Network: [1×1 vision.cnn.FastRCNN]
    RegionProposalNetwork: [1×1 vision.cnn.RegionProposalNetwork]
              MinBoxSizes: [37 17]
          BoxPyramidScale: 2
     NumBoxPyramidLevels: 6
               ClassNames: {'PL'  'PR'  'Background'}
            MinObjectSize: [5 5]

IdleTimeout has been reached.
Parallel pool using the 'local' profile is shutting down.
save rcnn
summary(rcnn)
{Undefined function 'summary' for input arguments of type
'fasterRCNNObjectDetector'.
}
rcnn

rcnn = 

  <a href="matlab:helpPopup fasterRCNNObjectDetector" style="font-
weight:bold">fasterRCNNObjectDetector</a> with properties:

                ModelName: 'PL'
                  Network: [1×1 vision.cnn.FastRCNN]
    RegionProposalNetwork: [1×1 vision.cnn.RegionProposalNetwork]
              MinBoxSizes: [37 17]
          BoxPyramidScale: 2
     NumBoxPyramidLevels: 6
               ClassNames: {'PL'  'PR'  'Background'}
            MinObjectSize: [5 5]

release(videoFReader)
release(videoPlayer)
release(videoFWriter)
release(videoFReader)
release(videoPlayer)
release(videoFWriter)
release(videoFReader)
release(videoPlayer)
release(videoFWriter)
release(videoFReader)
release(videoPlayer)
release(videoFWriter)
save rcnn
```