Train Object Detector Using R-CNN Deep Learning

César Martín Guijarro & Pablo García García

Firstly, we'll download the CIfar10 dataset and its pretrained convolutional neural network from the given URL.

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

Then, we load the data:

```
[trainingImages, trainingLabels, testImages, testLabels] =
helperCIFAR10Data.load(cifar10Data);
disp(size(trainingImages));
```

```
          32          32           3       50000
```

Now we can check the different categories of the images that the dataset contains.

```
numImageCategories = 10;
disp(categories(trainingLabels));
```

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

Let's see some of the images of the dataset:

```
figure;
thumbnails = trainingImages(:,:,:,1:100);
montage(thumbnails);
```

Now we are going to create the CNN. Later, in out slide presentation we will explain how the CNN works.

```
[height, width, numChannels, ~] = size(trainingImages);
imageSize = [height width numChannels];
inputLayer = imageInputLayer(imageSize)
```

```
inputLayer =
  ImageInputLayer with properties:

                      Name: ''
                 InputSize: [32 32 3]
       SplitComplexInputs: 0

   Hyperparameters
```

```
          DataAugmentation: 'none'
             Normalization: 'zerocenter'
    NormalizationDimension: 'auto'
                      Mean: []
```

```
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)
];

finalLayers = [
    fullyConnectedLayer(64)
    reluLayer
    fullyConnectedLayer(numImageCategories)
    softmaxLayer
    classificationLayer
];

layers = [
    inputLayer
    middleLayers
    finalLayers
];

layers(2).Weights = 0.0001 * randn([filterSize numChannels numFilters]);
```

Now we are able to train the net. We'll be using a NVIDIA GPU to make this process faster.

```
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);

cifar10Net = trainNetwork(trainingImages, trainingLabels, layers, opts);
```

```
Training on single GPU.
Initializing input data normalization.
```

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Mini-batch Loss	Base Learning Rate
1	1	00:00:08	8.59%	2.3037	0.0010
1	50	00:00:09	24.22%	2.0412	0.0010
1	100	00:00:10	28.91%	1.7514	0.0010
1	150	00:00:11	46.88%	1.6048	0.0010
1	200	00:00:12	45.31%	1.6095	0.0010
1	250	00:00:13	34.38%	1.7327	0.0010
1	300	00:00:13	50.78%	1.4226	0.0010
1	350	00:00:14	49.22%	1.3921	0.0010
2	400	00:00:15	35.94%	1.6275	0.0010
2	450	00:00:16	46.88%	1.3082	0.0010
2	500	00:00:17	49.22%	1.4088	0.0010
2	550	00:00:17	59.38%	1.2249	0.0010
2	600	00:00:18	50.78%	1.3892	0.0010
2	650	00:00:19	57.03%	1.2395	0.0010
2	700	00:00:20	49.22%	1.4112	0.0010
2	750	00:00:20	52.34%	1.4075	0.0010
3	800	00:00:21	56.25%	1.2624	0.0010
3	850	00:00:22	57.03%	1.2232	0.0010
3	900	00:00:23	60.94%	1.1718	0.0010
3	950	00:00:23	61.72%	1.0258	0.0010
3	1000	00:00:24	59.38%	1.1980	0.0010
3	1050	00:00:25	53.12%	1.2636	0.0010
3	1100	00:00:26	52.34%	1.3793	0.0010
3	1150	00:00:26	60.16%	1.0044	0.0010
4	1200	00:00:27	63.28%	1.0616	0.0010
4	1250	00:00:28	61.72%	1.2183	0.0010
4	1300	00:00:29	65.62%	0.9688	0.0010
4	1350	00:00:29	69.53%	0.9060	0.0010
4	1400	00:00:30	61.72%	1.0383	0.0010
4	1450	00:00:31	65.62%	0.9773	0.0010
4	1500	00:00:32	64.84%	1.1022	0.0010
4	1550	00:00:32	66.41%	1.0626	0.0010
5	1600	00:00:33	65.62%	0.9074	0.0010
5	1650	00:00:34	67.19%	1.0650	0.0010
5	1700	00:00:35	58.59%	1.1035	0.0010
5	1750	00:00:35	60.94%	0.9998	0.0010
5	1800	00:00:36	56.25%	1.1109	0.0010
5	1850	00:00:37	66.41%	0.9760	0.0010
5	1900	00:00:38	54.69%	1.2310	0.0010
5	1950	00:00:38	64.06%	1.0031	0.0010
6	2000	00:00:39	60.94%	1.1367	0.0010
6	2050	00:00:40	64.84%	0.8786	0.0010
6	2100	00:00:40	64.84%	1.0510	0.0010
6	2150	00:00:41	70.31%	0.9055	0.0010
6	2200	00:00:42	63.28%	0.9582	0.0010
6	2250	00:00:43	64.06%	0.9657	0.0010
6	2300	00:00:43	68.75%	0.8389	0.0010
7	2350	00:00:44	65.62%	0.9936	0.0010
7	2400	00:00:45	75.00%	0.7526	0.0010
7	2450	00:00:46	72.66%	0.8487	0.0010
7	2500	00:00:46	67.97%	0.8530	0.0010
7	2550	00:00:47	63.28%	1.0280	0.0010
7	2600	00:00:48	72.66%	0.8459	0.0010
7	2650	00:00:48	67.19%	1.0060	0.0010
7	2700	00:00:49	64.84%	0.9763	0.0010
8	2750	00:00:50	62.50%	1.0739	0.0010
8	2800	00:00:51	69.53%	0.9466	0.0010
8	2850	00:00:51	77.34%	0.7462	0.0010

8	2900	00:00:52	73.44%	0.8096	0.0010
8	2950	00:00:53	68.75%	0.8545	0.0010
8	3000	00:00:54	75.78%	0.7059	0.0010
8	3050	00:00:54	66.41%	1.0119	0.0010
8	3100	00:00:55	75.78%	0.6426	0.0010
9	3150	00:00:56	75.00%	0.8256	0.0001
9	3200	00:00:57	71.88%	0.8012	0.0001
9	3250	00:00:57	78.12%	0.6230	0.0001
9	3300	00:00:58	75.78%	0.7534	0.0001
9	3350	00:00:59	73.44%	0.6201	0.0001
9	3400	00:01:00	78.12%	0.5866	0.0001
9	3450	00:01:00	78.12%	0.6060	0.0001
9	3500	00:01:01	82.03%	0.5479	0.0001
10	3550	00:01:02	77.34%	0.6012	0.0001
10	3600	00:01:03	73.44%	0.8093	0.0001
10	3650	00:01:03	77.34%	0.7267	0.0001
10	3700	00:01:04	72.66%	0.6763	0.0001
10	3750	00:01:05	75.78%	0.6993	0.0001
10	3800	00:01:06	76.56%	0.6848	0.0001
10	3850	00:01:07	75.00%	0.8749	0.0001
10	3900	00:01:07	78.12%	0.7080	0.0001
11	3950	00:01:08	72.66%	0.8838	0.0001
11	4000	00:01:09	75.00%	0.6583	0.0001
11	4050	00:01:10	74.22%	0.7222	0.0001
11	4100	00:01:10	77.34%	0.7884	0.0001
11	4150	00:01:11	70.31%	0.7545	0.0001
11	4200	00:01:12	71.88%	0.7922	0.0001
11	4250	00:01:12	78.91%	0.5906	0.0001
12	4300	00:01:13	68.75%	0.7395	0.0001
12	4350	00:01:14	76.56%	0.5622	0.0001
12	4400	00:01:15	78.12%	0.6263	0.0001
12	4450	00:01:15	78.91%	0.6004	0.0001
12	4500	00:01:16	67.19%	0.9125	0.0001
12	4550	00:01:17	78.91%	0.5927	0.0001
12	4600	00:01:18	75.78%	0.6726	0.0001
12	4650	00:01:18	74.22%	0.8036	0.0001
13	4700	00:01:19	74.22%	0.7746	0.0001
13	4750	00:01:20	74.22%	0.6979	0.0001
13	4800	00:01:20	79.69%	0.5740	0.0001
13	4850	00:01:21	78.91%	0.6090	0.0001
13	4900	00:01:22	71.88%	0.7117	0.0001
13	4950	00:01:23	82.81%	0.5817	0.0001
13	5000	00:01:23	71.88%	0.8572	0.0001
13	5050	00:01:24	85.94%	0.4858	0.0001
14	5100	00:01:25	78.12%	0.6789	0.0001
14	5150	00:01:26	72.66%	0.7179	0.0001
14	5200	00:01:26	81.25%	0.5365	0.0001
14	5250	00:01:27	78.12%	0.6634	0.0001
14	5300	00:01:28	78.91%	0.5539	0.0001
14	5350	00:01:29	78.91%	0.5589	0.0001
14	5400	00:01:29	79.69%	0.5655	0.0001
14	5450	00:01:30	84.38%	0.5278	0.0001
15	5500	00:01:31	79.69%	0.5470	0.0001
15	5550	00:01:31	75.00%	0.7370	0.0001
15	5600	00:01:32	78.12%	0.6769	0.0001
15	5650	00:01:33	75.00%	0.6318	0.0001
15	5700	00:01:34	78.91%	0.6598	0.0001
15	5750	00:01:34	77.34%	0.6373	0.0001
15	5800	00:01:35	78.91%	0.8177	0.0001
15	5850	00:01:36	78.12%	0.6712	0.0001
16	5900	00:01:37	71.88%	0.8234	0.0001
16	5950	00:01:37	79.69%	0.6079	0.0001
16	6000	00:01:38	75.78%	0.6762	0.0001
16	6050	00:01:39	78.91%	0.7349	0.0001

16	6100	00:01:39	71.09%	0.6916	0.0001
16	6150	00:01:40	71.88%	0.7394	0.0001
16	6200	00:01:41	79.69%	0.5514	0.0001
17	6250	00:01:42	75.00%	0.6506	1.0000e-05
17	6300	00:01:42	78.91%	0.5271	1.0000e-05
17	6350	00:01:43	80.47%	0.5964	1.0000e-05
17	6400	00:01:44	78.12%	0.5737	1.0000e-05
17	6450	00:01:45	72.66%	0.8468	1.0000e-05
17	6500	00:01:45	79.69%	0.5616	1.0000e-05
17	6550	00:01:46	78.12%	0.6396	1.0000e-05
17	6600	00:01:47	77.34%	0.7440	1.0000e-05
18	6650	00:01:47	75.78%	0.7174	1.0000e-05
18	6700	00:01:48	76.56%	0.6764	1.0000e-05
18	6750	00:01:49	79.69%	0.5438	1.0000e-05
18	6800	00:01:50	82.03%	0.5570	1.0000e-05
18	6850	00:01:50	74.22%	0.7061	1.0000e-05
18	6900	00:01:51	83.59%	0.5440	1.0000e-05
18	6950	00:01:52	71.09%	0.8034	1.0000e-05
18	7000	00:01:53	85.94%	0.4745	1.0000e-05
19	7050	00:01:53	79.69%	0.6143	1.0000e-05
19	7100	00:01:54	74.22%	0.6699	1.0000e-05
19	7150	00:01:55	85.16%	0.5160	1.0000e-05
19	7200	00:01:56	82.03%	0.5920	1.0000e-05
19	7250	00:01:56	82.03%	0.5086	1.0000e-05
19	7300	00:01:57	78.91%	0.5292	1.0000e-05
19	7350	00:01:58	83.59%	0.5131	1.0000e-05
19	7400	00:01:58	84.38%	0.4854	1.0000e-05
20	7450	00:01:59	79.69%	0.5242	1.0000e-05
20	7500	00:02:00	74.22%	0.7193	1.0000e-05
20	7550	00:02:01	82.03%	0.6419	1.0000e-05
20	7600	00:02:01	76.56%	0.6057	1.0000e-05
20	7650	00:02:02	80.47%	0.5943	1.0000e-05
20	7700	00:02:03	78.91%	0.6293	1.0000e-05
20	7750	00:02:04	78.91%	0.7870	1.0000e-05
20	7800	00:02:04	79.69%	0.6409	1.0000e-05
21	7850	00:02:05	73.44%	0.7965	1.0000e-05
21	7900	00:02:06	78.91%	0.6008	1.0000e-05
21	7950	00:02:06	75.00%	0.6832	1.0000e-05
21	8000	00:02:07	78.91%	0.7300	1.0000e-05
21	8050	00:02:08	73.44%	0.6359	1.0000e-05
21	8100	00:02:09	73.44%	0.6968	1.0000e-05
21	8150	00:02:09	82.03%	0.5241	1.0000e-05
22	8200	00:02:10	77.34%	0.6524	1.0000e-05
22	8250	00:02:11	79.69%	0.5401	1.0000e-05
22	8300	00:02:12	81.25%	0.5829	1.0000e-05
22	8350	00:02:12	79.69%	0.5560	1.0000e-05
22	8400	00:02:13	72.66%	0.8347	1.0000e-05
22	8450	00:02:14	80.47%	0.5515	1.0000e-05
22	8500	00:02:15	79.69%	0.6365	1.0000e-05
22	8550	00:02:15	77.34%	0.7440	1.0000e-05
23	8600	00:02:16	75.78%	0.7043	1.0000e-05
23	8650	00:02:17	75.78%	0.6643	1.0000e-05
23	8700	00:02:17	82.81%	0.5390	1.0000e-05
23	8750	00:02:18	82.03%	0.5475	1.0000e-05
23	8800	00:02:19	74.22%	0.6992	1.0000e-05
23	8850	00:02:20	84.38%	0.5418	1.0000e-05
23	8900	00:02:20	71.09%	0.7999	1.0000e-05
23	8950	00:02:21	85.94%	0.4732	1.0000e-05
24	9000	00:02:22	78.91%	0.6097	1.0000e-05
24	9050	00:02:23	74.22%	0.6614	1.0000e-05
24	9100	00:02:23	85.94%	0.5104	1.0000e-05
24	9150	00:02:24	82.03%	0.5862	1.0000e-05
24	9200	00:02:25	82.03%	0.5037	1.0000e-05
24	9250	00:02:26	80.47%	0.5267	1.0000e-05

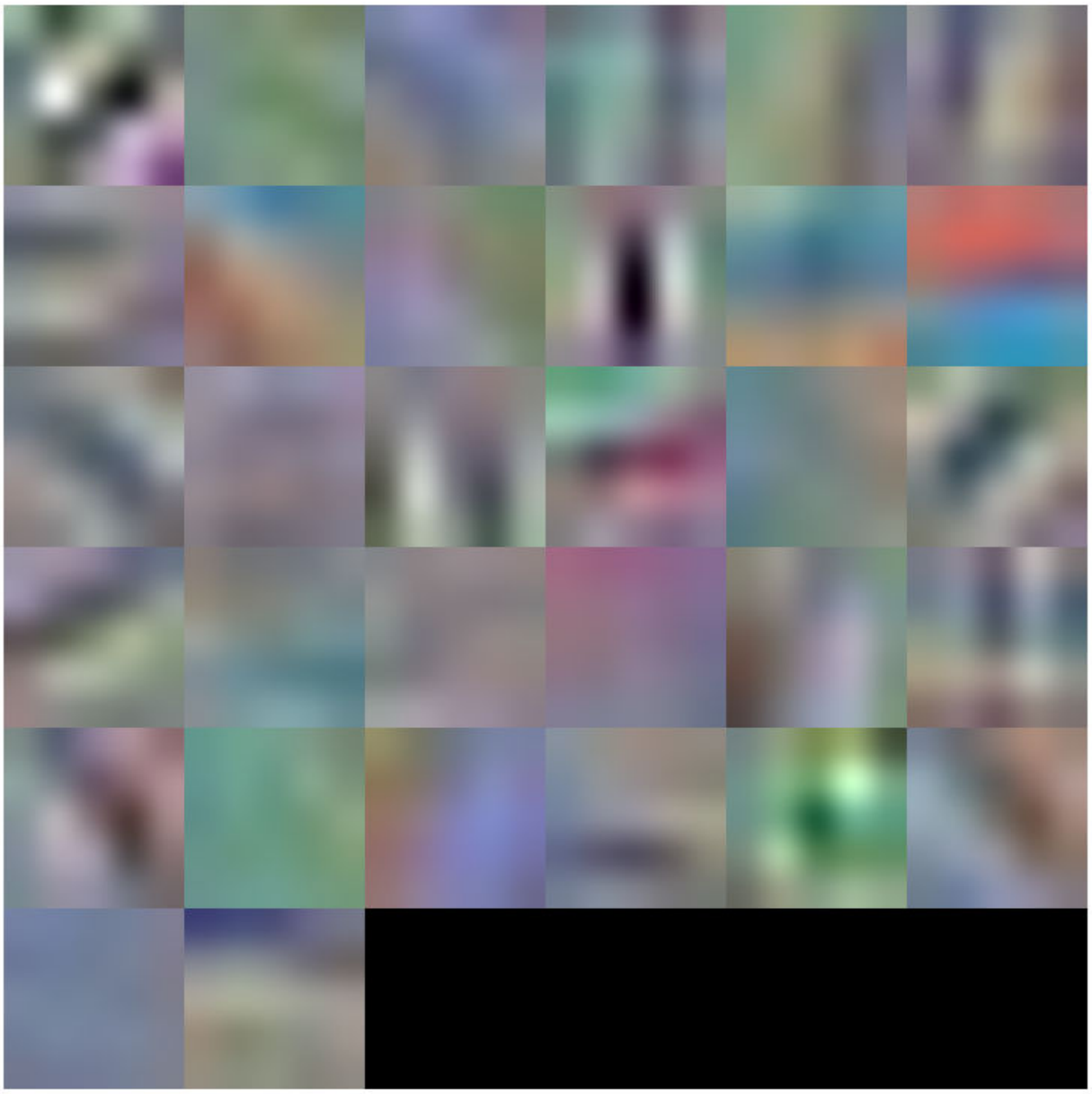
24	9300	00:02:26	84.38%	0.5114	1.0000e-05
24	9350	00:02:27	85.16%	0.4812	1.0000e-05
25	9400	00:02:28	78.91%	0.5150	1.0000e-06
25	9450	00:02:28	75.78%	0.7051	1.0000e-06
25	9500	00:02:29	83.59%	0.6348	1.0000e-06
25	9550	00:02:30	75.78%	0.5958	1.0000e-06
25	9600	00:02:31	81.25%	0.5870	1.0000e-06
25	9650	00:02:31	79.69%	0.6219	1.0000e-06
25	9700	00:02:32	77.34%	0.7773	1.0000e-06
25	9750	00:02:33	78.91%	0.6312	1.0000e-06
26	9800	00:02:34	72.66%	0.7920	1.0000e-06
26	9850	00:02:34	78.91%	0.5862	1.0000e-06
26	9900	00:02:35	75.00%	0.6861	1.0000e-06
26	9950	00:02:36	79.69%	0.7238	1.0000e-06
26	10000	00:02:37	73.44%	0.6216	1.0000e-06
26	10050	00:02:37	73.44%	0.6845	1.0000e-06
26	10100	00:02:38	82.03%	0.5174	1.0000e-06
27	10150	00:02:39	78.12%	0.6520	1.0000e-06
27	10200	00:02:40	79.69%	0.5348	1.0000e-06
27	10250	00:02:40	80.47%	0.5798	1.0000e-06
27	10300	00:02:41	80.47%	0.5488	1.0000e-06
27	10350	00:02:42	72.66%	0.8264	1.0000e-06
27	10400	00:02:42	81.25%	0.5435	1.0000e-06
27	10450	00:02:43	78.91%	0.6291	1.0000e-06
27	10500	00:02:44	77.34%	0.7377	1.0000e-06
28	10550	00:02:45	75.00%	0.6986	1.0000e-06
28	10600	00:02:45	77.34%	0.6585	1.0000e-06
28	10650	00:02:46	84.38%	0.5394	1.0000e-06
28	10700	00:02:47	82.03%	0.5417	1.0000e-06
28	10750	00:02:48	73.44%	0.6997	1.0000e-06
28	10800	00:02:48	84.38%	0.5357	1.0000e-06
28	10850	00:02:49	73.44%	0.7938	1.0000e-06
28	10900	00:02:50	85.94%	0.4696	1.0000e-06
29	10950	00:02:51	78.91%	0.6088	1.0000e-06
29	11000	00:02:51	74.22%	0.6581	1.0000e-06
29	11050	00:02:52	85.16%	0.5073	1.0000e-06
29	11100	00:02:53	82.03%	0.5782	1.0000e-06
29	11150	00:02:53	81.25%	0.5024	1.0000e-06
29	11200	00:02:54	77.34%	0.5271	1.0000e-06
29	11250	00:02:55	84.38%	0.5057	1.0000e-06
29	11300	00:02:56	87.50%	0.4766	1.0000e-06
30	11350	00:02:56	79.69%	0.5152	1.0000e-06
30	11400	00:02:57	75.00%	0.7047	1.0000e-06
30	11450	00:02:58	82.81%	0.6316	1.0000e-06
30	11500	00:02:59	75.78%	0.5971	1.0000e-06
30	11550	00:02:59	81.25%	0.5840	1.0000e-06
30	11600	00:03:00	78.91%	0.6208	1.0000e-06
30	11650	00:03:01	78.12%	0.7774	1.0000e-06
30	11700	00:03:01	78.91%	0.6318	1.0000e-06
31	11750	00:03:02	72.66%	0.7893	1.0000e-06
31	11800	00:03:03	78.91%	0.5859	1.0000e-06
31	11850	00:03:04	75.00%	0.6850	1.0000e-06
31	11900	00:03:04	79.69%	0.7213	1.0000e-06
31	11950	00:03:05	73.44%	0.6209	1.0000e-06
31	12000	00:03:06	73.44%	0.6839	1.0000e-06
31	12050	00:03:07	82.03%	0.5178	1.0000e-06
32	12100	00:03:07	78.12%	0.6516	1.0000e-06
32	12150	00:03:08	80.47%	0.5348	1.0000e-06
32	12200	00:03:09	80.47%	0.5784	1.0000e-06
32	12250	00:03:10	80.47%	0.5488	1.0000e-06
32	12300	00:03:10	72.66%	0.8255	1.0000e-06
32	12350	00:03:11	81.25%	0.5429	1.0000e-06
32	12400	00:03:12	78.91%	0.6289	1.0000e-06
32	12450	00:03:12	77.34%	0.7376	1.0000e-06

33	12500	00:03:13	75.78%	0.6972	1.0000e-07
33	12550	00:03:14	76.56%	0.6580	1.0000e-07
33	12600	00:03:15	84.38%	0.5390	1.0000e-07
33	12650	00:03:15	82.03%	0.5415	1.0000e-07
33	12700	00:03:16	73.44%	0.6985	1.0000e-07
33	12750	00:03:17	85.16%	0.5355	1.0000e-07
33	12800	00:03:18	73.44%	0.7932	1.0000e-07
33	12850	00:03:18	86.72%	0.4687	1.0000e-07
34	12900	00:03:19	78.91%	0.6083	1.0000e-07
34	12950	00:03:20	74.22%	0.6582	1.0000e-07
34	13000	00:03:20	85.16%	0.5070	1.0000e-07
34	13050	00:03:21	82.03%	0.5777	1.0000e-07
34	13100	00:03:22	81.25%	0.5017	1.0000e-07
34	13150	00:03:23	77.34%	0.5265	1.0000e-07
34	13200	00:03:23	84.38%	0.5058	1.0000e-07
34	13250	00:03:24	87.50%	0.4764	1.0000e-07
35	13300	00:03:25	79.69%	0.5143	1.0000e-07
35	13350	00:03:26	75.00%	0.7027	1.0000e-07
35	13400	00:03:26	82.81%	0.6307	1.0000e-07
35	13450	00:03:27	75.78%	0.5967	1.0000e-07
35	13500	00:03:28	82.03%	0.5825	1.0000e-07
35	13550	00:03:29	78.91%	0.6188	1.0000e-07
35	13600	00:03:29	78.12%	0.7762	1.0000e-07
35	13650	00:03:30	78.91%	0.6312	1.0000e-07
36	13700	00:03:31	72.66%	0.7890	1.0000e-07
36	13750	00:03:32	79.69%	0.5846	1.0000e-07
36	13800	00:03:32	75.00%	0.6850	1.0000e-07
36	13850	00:03:33	79.69%	0.7214	1.0000e-07
36	13900	00:03:34	73.44%	0.6193	1.0000e-07
36	13950	00:03:34	73.44%	0.6829	1.0000e-07
36	14000	00:03:35	82.03%	0.5178	1.0000e-07
37	14050	00:03:36	78.12%	0.6512	1.0000e-07
37	14100	00:03:37	80.47%	0.5347	1.0000e-07
37	14150	00:03:37	80.47%	0.5785	1.0000e-07
37	14200	00:03:38	80.47%	0.5474	1.0000e-07
37	14250	00:03:39	72.66%	0.8246	1.0000e-07
37	14300	00:03:40	81.25%	0.5415	1.0000e-07
37	14350	00:03:40	78.91%	0.6285	1.0000e-07
37	14400	00:03:41	77.34%	0.7370	1.0000e-07
38	14450	00:03:42	75.78%	0.6969	1.0000e-07
38	14500	00:03:42	76.56%	0.6577	1.0000e-07
38	14550	00:03:43	84.38%	0.5390	1.0000e-07
38	14600	00:03:44	82.03%	0.5413	1.0000e-07
38	14650	00:03:45	73.44%	0.6984	1.0000e-07
38	14700	00:03:45	85.16%	0.5353	1.0000e-07
38	14750	00:03:46	73.44%	0.7933	1.0000e-07
38	14800	00:03:47	86.72%	0.4686	1.0000e-07
39	14850	00:03:48	78.91%	0.6080	1.0000e-07
39	14900	00:03:48	74.22%	0.6581	1.0000e-07
39	14950	00:03:49	85.16%	0.5070	1.0000e-07
39	15000	00:03:50	82.03%	0.5777	1.0000e-07
39	15050	00:03:50	81.25%	0.5018	1.0000e-07
39	15100	00:03:51	77.34%	0.5267	1.0000e-07
39	15150	00:03:52	84.38%	0.5058	1.0000e-07
39	15200	00:03:53	87.50%	0.4765	1.0000e-07
40	15250	00:03:53	79.69%	0.5143	1.0000e-07
40	15300	00:03:54	75.00%	0.7027	1.0000e-07
40	15350	00:03:55	82.81%	0.6304	1.0000e-07
40	15400	00:03:56	75.78%	0.5968	1.0000e-07
40	15450	00:03:56	82.03%	0.5824	1.0000e-07
40	15500	00:03:57	79.69%	0.6189	1.0000e-07
40	15550	00:03:58	78.91%	0.7765	1.0000e-07
40	15600	00:03:58	78.91%	0.6312	1.0000e-07

```
|=========================================================================================|
Training finished: Max epochs completed.
```

Once the training has finished, we can see what is happening in some of the middle layers. In this case, it seems as it its trying to detect borders.

```
w = cifar10Net.Layers(2).Weights;
w = rescale(w);

figure
montage(w);
```



Now we will check if the network has been trained correctly:

```
YTest = classify(cifar10Net, testImages);
accuracy = sum(YTest == testLabels)/numel(testLabels)
```

```
accuracy = 0.7072
```

Detecting STOP signs

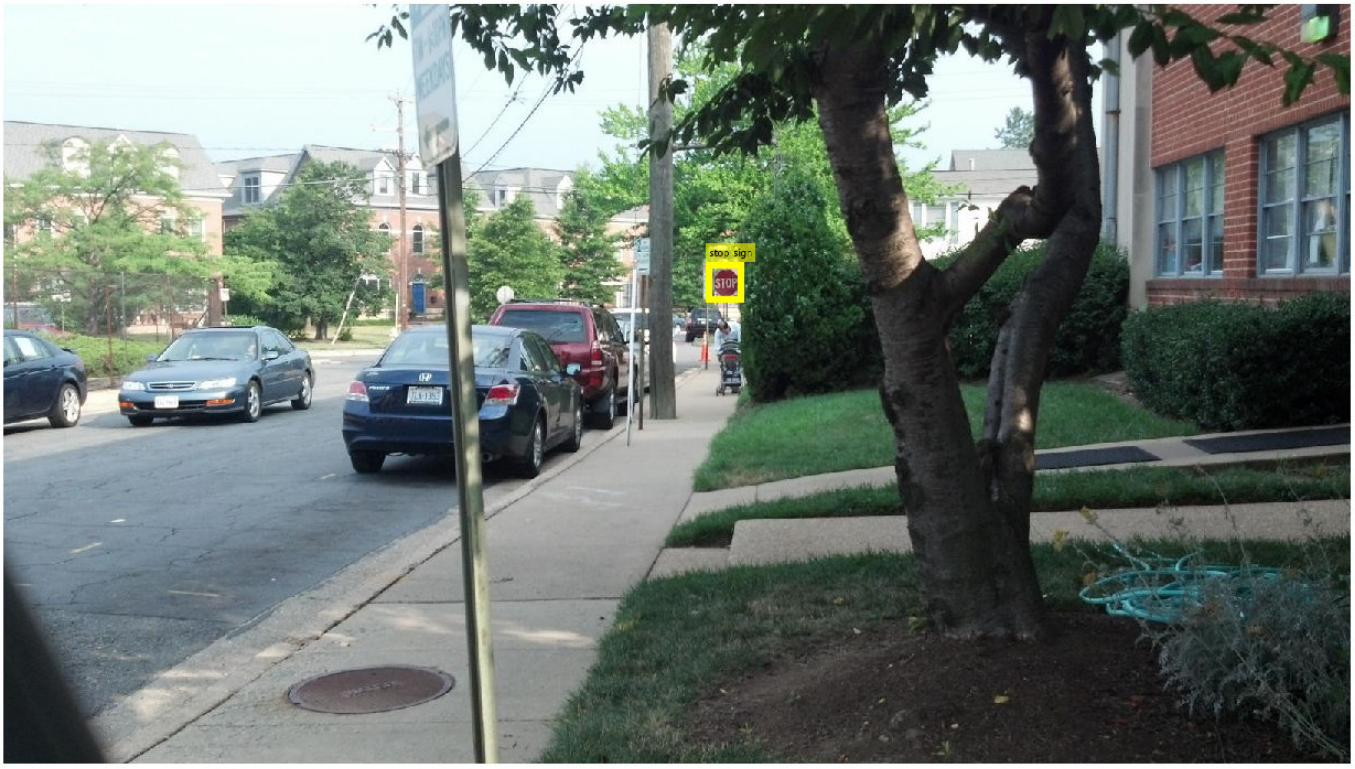
Having correctly trained our CNN, we will try to use it to detect STOP signs. To do so, we need to load only the images we will use. In this case will follow the MATLAB example.

```
data = load('stopSignsAndCars.mat', 'stopSignsAndCars');
stopSignsAndCars = data.stopSignsAndCars;

visiondata = fullfile(toolboxdir('vision'), 'visiondata');
stopSignsAndCars.imageFilename = fullfile(visiondata,
stopSignsAndCars.imageFilename);

summary(stopSignsAndCars)
```

```
Variables:

    imageFilename: 41×1 cell array of character vectors

    stopSign: 41×1 cell

    carRear: 41×1 cell

    carFront: 41×1 cell
```

In this dataset, every object of interest has a bounding box surrounding it, so to detect stop signs we need to load only the images which them. Let's see an example of an image with an stop sign with its boundary box.

```
stopSigns = stopSignsAndCars(:, {'imageFilename','stopSign'});

I = imread(stopSigns.imageFilename{1});
I = insertObjectAnnotation(I,'Rectangle',stopSigns.stopSign{1},'stop
sign','LineWidth',8);

figure
imshow(I)
```



Now we will train our CNN but not from 0 knowledge, it will be a training that modifies the knowledge of the CIFAR-10 network.

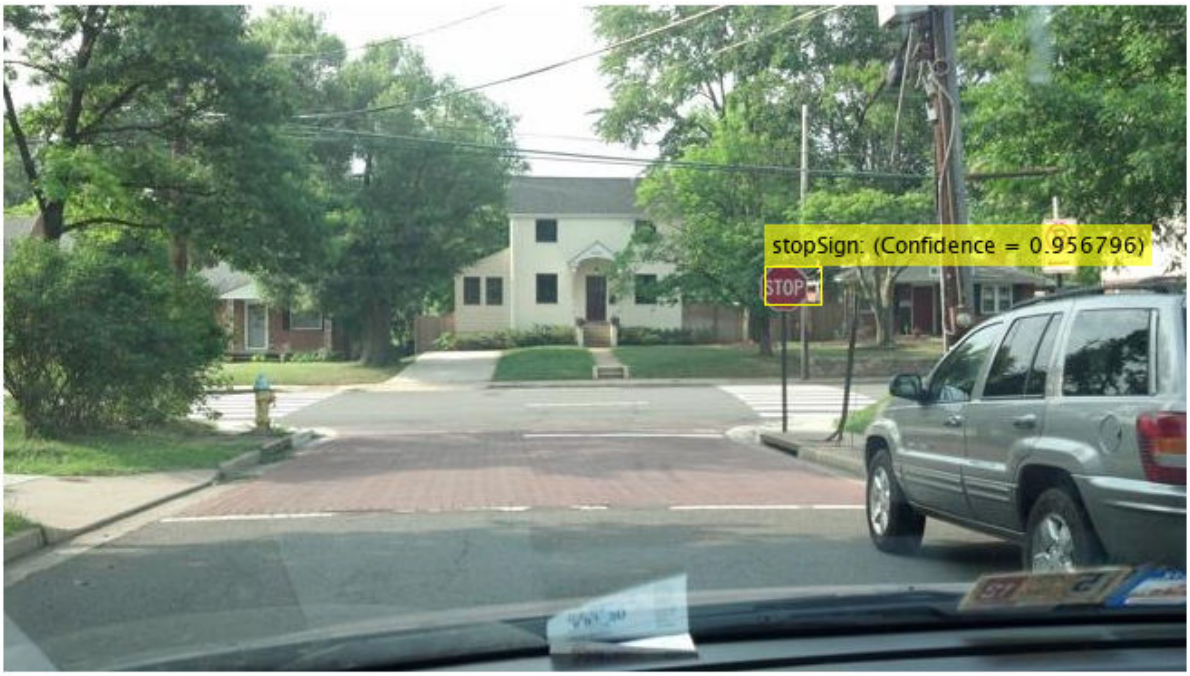
```
options = trainingOptions('sgdm', ...
        'MiniBatchSize', 128, ...
        'InitialLearnRate', 1e-3, ...
        'LearnRateSchedule', 'piecewise', ...
        'LearnRateDropFactor', 0.1, ...
        'LearnRateDropPeriod', 100, ...
        'MaxEpochs', 100, ...
        'Verbose', true);

rcnn = trainRCNNObjectDetector(stopSigns, cifar10Net, options,
'NegativeOverlapRange', [0 0.3], 'PositiveOverlapRange',[0.5 1]);
```

```
Starting parallel pool (parpool) using the 'Processes' profile ...
Parallel pool using the 'Processes' profile is shutting down.

*********************************************************************
Training an R-CNN Object Detector for the following object classes:

* stopSign

--> Extracting region proposals from 41 training images...done.

--> Training a neural network to classify objects in training data...

Training on single GPU.
Initializing input data normalization.
```

```
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:00 |       57.03% |       0.7087 |          0.0010 |
|       6 |          50 |       00:00:19 |       99.22% |       0.0135 |          0.0010 |
|      12 |         100 |       00:00:38 |      100.00% |       0.0125 |          0.0010 |
|      17 |         150 |       00:00:56 |      100.00% |       0.0010 |          0.0010 |
|      23 |         200 |       00:01:14 |       99.22% |       0.0068 |          0.0010 |
|      28 |         250 |       00:01:32 |      100.00% |       0.0005 |          0.0010 |
|      34 |         300 |       00:01:51 |      100.00% |       0.0009 |          0.0010 |
|      39 |         350 |       00:02:09 |      100.00% |   9.9973e-05 |          0.0010 |
|      45 |         400 |       00:02:27 |      100.00% |   8.3369e-05 |          0.0010 |
|      50 |         450 |       00:02:46 |      100.00% |       0.0004 |          0.0010 |
|      56 |         500 |       00:03:04 |      100.00% |       0.0005 |          0.0010 |
|      62 |         550 |       00:03:23 |      100.00% |       0.0009 |          0.0010 |
|      67 |         600 |       00:03:41 |      100.00% |       0.0003 |          0.0010 |
|      73 |         650 |       00:03:59 |      100.00% |       0.0001 |          0.0010 |
|      78 |         700 |       00:04:18 |      100.00% |       0.0001 |          0.0010 |
|      84 |         750 |       00:04:36 |      100.00% |       0.0003 |          0.0010 |
|      89 |         800 |       00:04:54 |      100.00% |       0.0001 |          0.0010 |
|      95 |         850 |       00:05:13 |      100.00% |       0.0002 |          0.0010 |
|     100 |         900 |       00:05:31 |      100.00% |       0.0001 |          0.0010 |
|=========================================================================================|
Training finished: Max epochs completed.

Network training complete.

--> Training bounding box regression models for each object class...100.00%...done.

Detector training complete.
*****************************************************************
```

As we've done before, once the training has finished, we can try the net.

```
testImage = imread('stopSignTest.jpg');
[bboxes,score,label] = detect(rcnn, testImage, 'MiniBatchSize', 128);

[score, idx] = max(score);

bbox = bboxes(idx, :);
annotation = sprintf('%s: (Confidence = %f)', label(idx), score);

outputImage = insertObjectAnnotation(testImage, 'rectangle', bbox, annotation);

figure;
imshow(outputImage);
```



Now we can try the model with some photos taken by us:

```
testImage = imread('imagenes_jpg/stop/1.jpg');
[bboxes, score, label] = detect(rcnn, testImage, 'MiniBatchSize', 128);
[score, idx] = max(score);
bbox = bboxes(idx, :);
annotation = sprintf('%s: (Confidence = %f)', label(idx), score);
outputImage = insertObjectAnnotation(testImage, 'rectangle', bbox, annotation);
figure;
imshow(outputImage);
```

DETECTION OF SIGNS IN OUR OWN PHOTOS

To cover this section we went out to the street and took pictures of different traffic signs, so we can train different CNNs over the Cifar one. We have also made a MATLAB script which generates a dataset with each type of image. In this first example we will try to detect crosswalk signs. To begin with, we show one of our dataset images.

```
load('./imagenes_jpg/peatones/dataset_peatones.mat');
dataset_pedestrian = dataset;

I_pedestrian = imread(dataset_pedestrian.paths{1});
I_pedestrian = insertObjectAnnotation(I_pedestrian, 'Rectangle',
dataset_pedestrian.peatones{1}, 'Pedestrian sign', 'LineWidth', 8);

figure;
imshow(I_pedestrian);
```

Once we've seen an example image, we can train the net. In this case, we will implement a different algorithm than the propussed one, we will use Adam which will be also explained in our slides. Also we will use our NVIDIA GPU to train faster.

```
options = trainingOptions('adam', ...
        'MiniBatchSize', 128, ...
        'InitialLearnRate', 1e-3, ...
        'LearnRateSchedule', 'piecewise', ...
        'LearnRateDropFactor', 0.1, ...
        'LearnRateDropPeriod', 100, ...
        'MaxEpochs', 100, ...
        'Verbose', true, ...
        'ExecutionEnvironment', 'gpu');

rcnn_pedestrian = trainRCNNObjectDetector(dataset_pedestrian, cifar10Net, options,
'NegativeOverlapRange', [0 0.3], 'PositiveOverlapRange',[0.5 1]);
```

```
Starting parallel pool (parpool) using the 'Processes' profile ...
Parallel pool using the 'Processes' profile is shutting down.

*************************************************************
```

```
Training an R-CNN Object Detector for the following object classes:

* rois

--> Extracting region proposals from 33 training images...done.

--> Training a neural network to classify objects in training data...

Initializing input data normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:00 |       31.25% |       0.7435 |          0.0010 |
|       6 |          50 |       00:00:12 |       94.53% |       0.1587 |          0.0010 |
|      12 |         100 |       00:00:25 |      100.00% |       0.0002 |          0.0010 |
|      17 |         150 |       00:00:37 |       99.22% |       0.0070 |          0.0010 |
|      23 |         200 |       00:00:49 |      100.00% |   6.2246e-05 |          0.0010 |
|      28 |         250 |       00:01:01 |      100.00% |       0.0074 |          0.0010 |
|      34 |         300 |       00:01:14 |      100.00% |       0.0014 |          0.0010 |
|      39 |         350 |       00:01:26 |       99.22% |       0.0275 |          0.0010 |
|      45 |         400 |       00:01:38 |      100.00% |       0.0003 |          0.0010 |
|      50 |         450 |       00:01:50 |      100.00% |   6.1628e-05 |          0.0010 |
|      56 |         500 |       00:02:02 |       99.22% |       0.0100 |          0.0010 |
|      62 |         550 |       00:02:14 |      100.00% |       0.0017 |          0.0010 |
|      67 |         600 |       00:02:26 |       99.22% |       0.0890 |          0.0010 |
|      73 |         650 |       00:02:38 |      100.00% |   2.2494e-05 |          0.0010 |
|      78 |         700 |       00:02:51 |      100.00% |       0.0023 |          0.0010 |
|      84 |         750 |       00:03:03 |      100.00% |       0.0001 |          0.0010 |
|      89 |         800 |       00:03:15 |      100.00% |       0.0029 |          0.0010 |
|      95 |         850 |       00:03:27 |      100.00% |   8.0533e-05 |          0.0010 |
|     100 |         900 |       00:03:39 |      100.00% |   6.2596e-06 |          0.0010 |
|=========================================================================================|
Training finished: Max epochs completed.

Network training complete.

--> Training bounding box regression models for each object class...100.00%...done.

Detector training complete.
*************************************************************
```

When the model has finished training, we can try with another photo

```
test_image_pedestrian = imread('./imagenes_maps/1.png');
[bboxes, score, label] = detect(rcnn_pedestrian, test_image_pedestrian,
'MiniBatchSize', 128);

[score, idx] = max(score);
bbox = bboxes(idx, :);
annotation = sprintf('%s: (Confidence = %f)', 'Pedestrian sign', score);
outputImage = insertObjectAnnotation(test_image_pedestrian, 'rectangle', bbox,
annotation);
figure;
imshow(outputImage);
```

Also, we modified the recognition code so if more than one object appears in the image, it is also detected.

```
test_image_pedestrian = imread('./imagenes_maps/11.png');
[bboxes, score, label] = detect(rcnn_pedestrian, test_image_pedestrian,
'MiniBatchSize', 128);

instances = 5;  %specify how many objects to detect

objects = min(instances,size(bboxes,1));
[scores_sorted, idx_sorted] = sort(score, 'descend');
outputImage = test_image_pedestrian;

for i = 1:objects
    bbox = bboxes(idx_sorted(i), :);
    annotation = sprintf('%s: (Confidence = %f)', 'Pedestrian sign',
scores_sorted(i));
```

```
    outputImage = insertObjectAnnotation(outputImage, 'rectangle', bbox,
annotation);
end


figure;
imshow(outputImage);
```

Maybe a too many objetcs are detected, when they shouldn't. To fix that we can set a confidence threshold and show only those whose confidence threshold is bigger than the set one. Here is the alternative version with a minimum score threshold:

```
test_image_pedestrian = imread('./imagenes_jpg/direccion_prohibida/3.jpg');
[bboxes, score, label] = detect(rcnn_pedestrian, test_image_pedestrian,
'MiniBatchSize', 128);

% Set the minimum score threshold
min_score_threshold = 0.7;

% Filter out detections below the threshold
valid_indices = find(score >= min_score_threshold);
bboxes = bboxes(valid_indices, :);
score = score(valid_indices);
label = label(valid_indices);

objects = numel(valid_indices);
```

```
% Sort the scores
[scores_sorted, idx_sorted] = sort(score, 'descend');
outputImage = test_image_pedestrian;

for i = 1:objects
    bbox = bboxes(idx_sorted(i), :);
    annotation = sprintf('%s: (Confidence = %f)', 'Pedestrian sign',
scores_sorted(i));
    outputImage = insertObjectAnnotation(outputImage, 'rectangle', bbox,
annotation);
end

figure;
imshow(outputImage);
```

Heatmap

As shown by the Matlab example, this can also be used to process the entire image and gain information about what the Network is actually seing.

```
% The trained network is stored within the R-CNN detector
rcnn_pedestrian.Network
```

```
ans =
  SeriesNetwork with properties:

         Layers: [15×1 nnet.cnn.layer.Layer]
     InputNames: {'imageinput'}
    OutputNames: {'rcnnClassification'}
```

As a way to debug, we can extract the activations of the networks. The information we need is stored in different dimentions.

```
test_image_pedestrian = imread('./imagenes_maps/9.png');
featureMap = activations(rcnn_pedestrian.Network, test_image_pedestrian, 14);

% The softmax activations are stored in a 3-D array.
size(featureMap)
```

```
ans = 1×3
   107   149     2
```

In this case the pedestrian signs are stored in the third dimention.

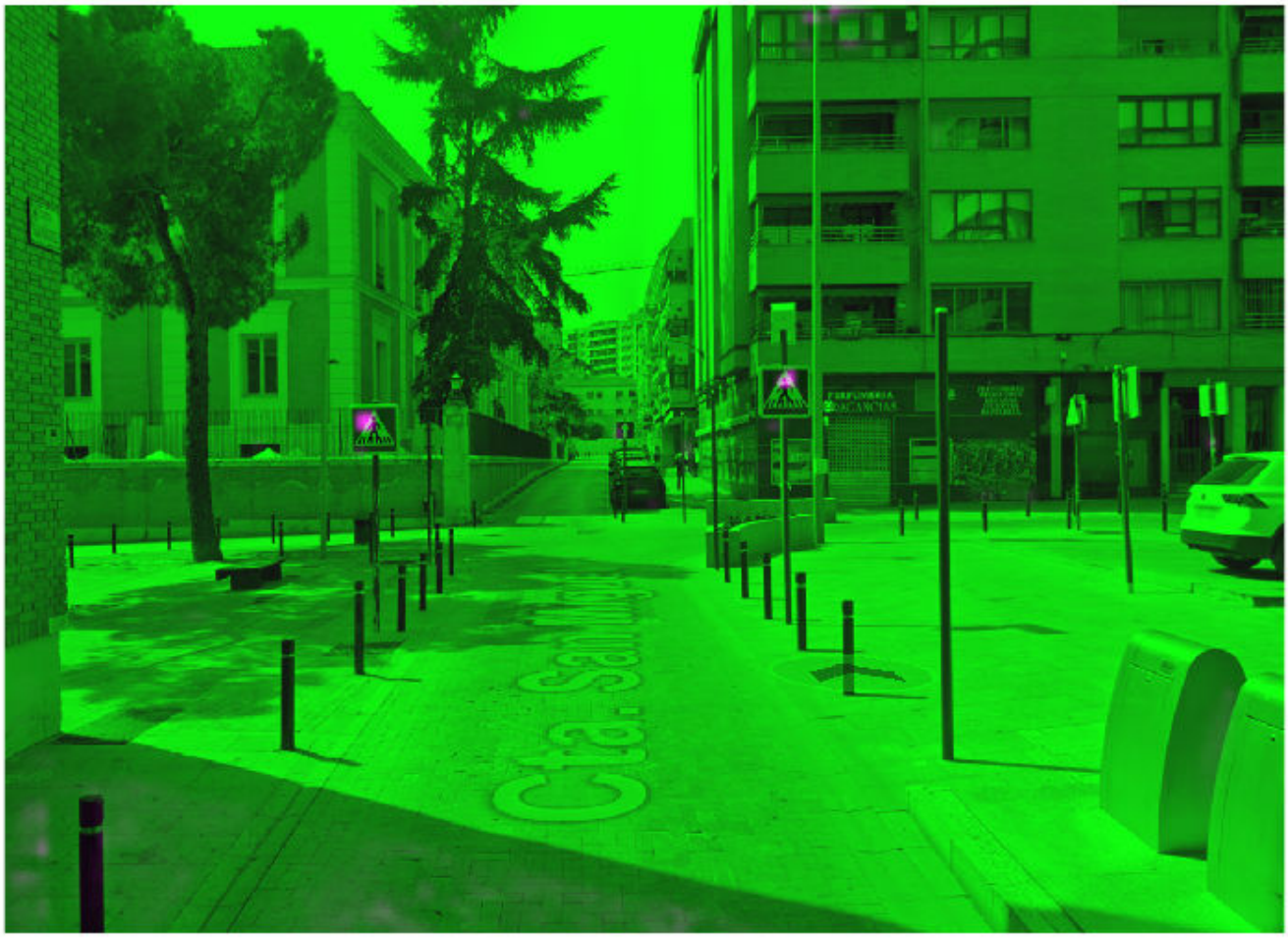
```
rcnn_pedestrian.ClassNames
```

The pedestrian sign feature map is stored in the first channel.

```
pedestrianSignMap = featureMap(:, :, 1);
```

Even though this is a "rough aproximation" of what the Network is seeing, it can

```
% Resize stopSignMap for visualization
[height, width, ~] = size(test_image_pedestrian);
pedestrianSignMap = imresize(pedestrianSignMap, [height, width]);

% Visualize the feature map superimposed on the test image.
featureMapOnImage = imfuse(test_image_pedestrian, pedestrianSignMap);

figure
imshow(featureMapOnImage)
```



BAD EXAMPLE: TRAFFIC LIGHT DETECTION

Not everything is perfect, CNNs not always work perfectly. Detecting 3D objects with strange backgrounds is a very hard task. To show that, we have also taken as many photos of traffic lights as of pedestrian signs (more or less), and despite the fact that for pedestrian signs it worked perfectly, now we will see that it is not the case with traffic lights. Maybe it is caused by the fact that photos can be taken in different angles, now what we want to detect is a 3D object and its apperance depends on the angle the photo is taken, not as the sign, independently where you take the photo from, it will always look like a blue square with a white triangle inside. Also the backgrounds of the traffic lights didn't help.

```
load('./imagenes_jpg/semaforo/dataset_semaforo.mat');
dataset_traffic_lights = dataset;

I_traffic_lights = imread(dataset_traffic_lights.paths{1});
I_traffic_lights = insertObjectAnnotation(I_traffic_lights, 'Rectangle',
dataset_traffic_lights.semaforo{1}, 'Traffic light', 'LineWidth', 8);

figure;
imshow(I_traffic_lights);
```

Once we've seen an example image, we can train the net, also with Adam and our NVIDIA GPU to finish the training earlier.

```
rcnn_traffic_lights = trainRCNNObjectDetector(dataset_traffic_lights, cifar10Net,
options, 'NegativeOverlapRange', [0 0.3], 'PositiveOverlapRange',[0.5 1]);
```

```
Starting parallel pool (parpool) using the 'Processes' profile ...
Connected to parallel pool with 6 workers.
*************************************************************
Training an R-CNN Object Detector for the following object classes:

* rois

--> Extracting region proposals from 31 training images...done.

--> Training a neural network to classify objects in training data...

Initializing input data normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:01 |       64.84% |       0.6583 |          0.0010 |
|      25 |          50 |       00:00:06 |      100.00% |       0.0109 |          0.0010 |
```

```
|      50 |         100 |       00:00:10 |      100.00% |       0.0008 |          0.0010 |
|      75 |         150 |       00:00:15 |      100.00% |       0.0007 |          0.0010 |
|     100 |         200 |       00:00:20 |      100.00% |       0.0101 |          0.0010 |
|=========================================================================================|
Training finished: Max epochs completed.

Network training complete.

--> Training bounding box regression models for each object class...100.00%...done.

Detector training complete.
*************************************************************
```

When the model has finished training, we can try with another photo or any Google Maps photo. As we can see, it will get confused very easy.

```
test_image_traffic_lights = imread('imagenes_maps/2.png');
[bboxes, score, label] = detect(rcnn_traffic_lights, test_image_traffic_lights,
'MiniBatchSize', 128);
[score, idx] = max(score);
bbox = bboxes(idx, :);
annotation = sprintf('%s: (Confidence = %f)', 'Traffic light', score);
outputImage = insertObjectAnnotation(test_image_traffic_lights, 'rectangle', bbox,
annotation);
figure;
imshow(outputImage)
```

WRONG WAY SIGN

```
load('./imagenes_jpg/direccion_prohibida/dataset_direccion_prohibida.mat');
dataset_wrong_way = dataset;

I_wrong_way = imread(dataset_wrong_way.paths{1});
I_wrong_way = insertObjectAnnotation(I_wrong_way, 'Rectangle',
dataset_wrong_way.direccion_prohibida{1}, 'Wrong Way sign', 'LineWidth', 8);

figure;
imshow(I_wrong_way);
```

Once we've seen an example image, we can train the net. In this case, we will implement a different algorithm than the propussed one, we will use Adam. Also we will use our NVIDIA GPU to train faster

```
rcnn_wrong_way = trainRCNNObjectDetector(dataset_wrong_way, cifar10Net, options,
'NegativeOverlapRange', [0 0.3], 'PositiveOverlapRange',[0.5 1]);
```

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

* rois
```

```
--> Extracting region proposals from 20 training images...done.

--> Training a neural network to classify objects in training data...

Initializing input data normalization.
|========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|========================================================================================|
|       1 |           1 |       00:00:00 |       41.41% |       0.7635 |          0.0010 |
|      25 |          50 |       00:00:04 |      100.00% |   2.5319e-05 |          0.0010 |
|      50 |         100 |       00:00:08 |      100.00% |   5.7926e-07 |          0.0010 |
|      75 |         150 |       00:00:11 |      100.00% |   5.9001e-06 |          0.0010 |
|     100 |         200 |       00:00:15 |      100.00% |       0.0003 |          0.0010 |
|========================================================================================|
Training finished: Max epochs completed.

Network training complete.

--> Training bounding box regression models for each object class...100.00%...done.

Detector training complete.
*************************************************************
```

When the model has finished training, we can try with another photo or any Google Maps photo.

```
test_image_wrong_way = imread('./imagenes_maps/4.png');
[bboxes, score, label] = detect(rcnn_wrong_way, test_image_wrong_way,
'MiniBatchSize', 128);
[score, idx] = max(score);
bbox = bboxes(idx, :);
annotation = sprintf('%s: (Confidence = %f)', 'Wrong way sign', score);
outputImage = insertObjectAnnotation(test_image_wrong_way, 'rectangle', bbox,
annotation);
figure;
imshow(outputImage)
```

YIELD RIGHT OF WAY

To prove the power of CNNs, as we detected pedestrian signs, we can detect easily any other 2D patterns in the street. Maybe its a very repetitive topic but it is very useful since autonumus driving is the future, and this type of recognition could be implemented in this technology. In this case we will detect 'Ceda el paso' signs. In this case a triangular pattern. With some photos taken by us, we create the dataset and then train the net over the Cifar10 one.

```
load('./imagenes_jpg/ceda/dataset_ceda.mat');
dataset_ceda = dataset;

I_ceda = imread(dataset_ceda.paths{1});
```

```
I_ceda = insertObjectAnnotation(I_ceda, 'Rectangle', dataset_ceda.ceda{1}, 'Yield
sign', 'LineWidth', 8);

figure;
imshow(I_ceda);
```

Then, the training begins.

```
rcnn_ceda = trainRCNNObjectDetector(dataset_ceda, cifar10Net, options,
'NegativeOverlapRange', [0 0.3], 'PositiveOverlapRange',[0.5 1]);
```

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

* ceda

--> Extracting region proposals from 8 training images...done.

--> Training a neural network to classify objects in training data...

Initializing input data normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
```

```
|=========================================================================================|
|       1 |           1 |       00:00:04 |       64.06% |       0.6541 |          0.0010 |
|      50 |          50 |       00:00:13 |      100.00% |   7.7516e-06 |          0.0010 |
|     100 |         100 |       00:00:20 |      100.00% |       0.0002 |          0.0010 |
|=========================================================================================|
Training finished: Max epochs completed.

Network training complete.

--> Training bounding box regression models for each object class...100.00%...done.

Detector training complete.
**************************************************************
```

When the model has finished training, we can try with another photo or any Google Maps photo.

```
test_image_ceda = imread('./imagenes_maps/10.png');
[bboxes, score, label] = detect(rcnn_ceda, test_image_ceda, 'MiniBatchSize', 128);
[score, idx] = max(score);
bbox = bboxes(idx, :);
annotation = sprintf('%s: (Confidence = %f)', 'Yield sign', score);
outputImage = insertObjectAnnotation(test_image_ceda, 'rectangle', bbox,
annotation);
figure;
imshow(outputImage)
```

Detecting more than one type of object:

To detect multiple classes, the usage of more than one network is required. First we set the networks and labels that we want to use.

```
test_image = imread('./imagenes_jpg/direccion_prohibida/3.jpg');

outputImage = test_image;

% Set the minimum score threshold
min_score_threshold = 0.2;

networks = {rcnn_pedestrian;rcnn_wrong_way};
labels = ["Pedestrian sign", "Wrong way sign"]
```

```
labels = 1×2 string
```

```
"Pedestrian …  "Wrong way …
```

Then, we iterate over each of the networks previously set.

```
for i= 1:numel(networks)
    network = networks{i};
    % Filter out detections below the threshold
    [bboxes, score, label] = detect(network, test_image, 'MiniBatchSize', 128);
    valid_indices = find(score >= min_score_threshold);
    bboxes = bboxes(valid_indices, :);
    score = score(valid_indices);
    label = label(valid_indices);

    objects = numel(valid_indices);

    % Sort the scores
    [scores_sorted, idx_sorted] = sort(score, 'descend');


    for j = 1:objects
        labelidx = label(idx);
        bbox = bboxes(idx_sorted(j), :);
        annotation = sprintf('%s: (Confidence = %f)', labels(i), scores_sorted(j));
        outputImage = insertObjectAnnotation(outputImage, 'rectangle', bbox,
annotation);
    end
end

figure;
imshow(outputImage);
```


Pedestrian sign (Confidence = 1.000000)
Wrong way sign (Confidence = 1.000000)