Deep Learning (MATLAB)

Contents

1 The CIFAR-10 dataset **2**

2 Convolutional Neural Networks (CNNs / ConvNets) **2**

2.1 Architecture Overview 2

2.2 Layers used to build ConvNets 4

2.2.1 Convolutional Layer 4

2.2.2 Pooling Layer 5

3 Deep Learning Softwares **7**

4 Tasks **7**

5 Codes **7**

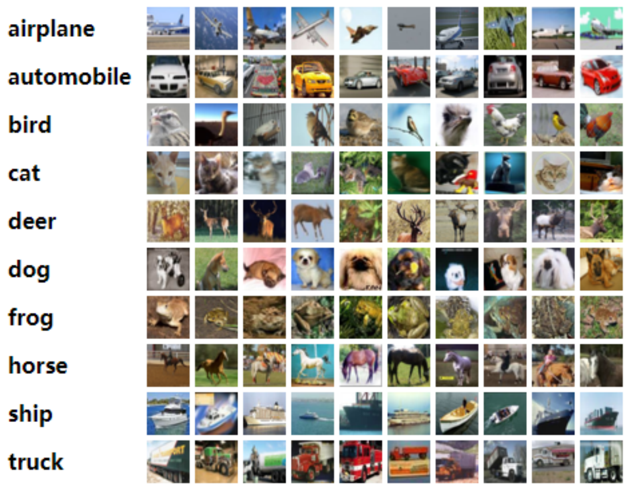
6 Results **13**

7 Reference **13**

1 The CIFAR-10 dataset

The CIFAR-10 dataset (`http://www.cs.toronto.edu/~kriz/cifar.html`) consists of 60000 32×32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. Here are the classes in the dataset, as well as 10 random images from each:



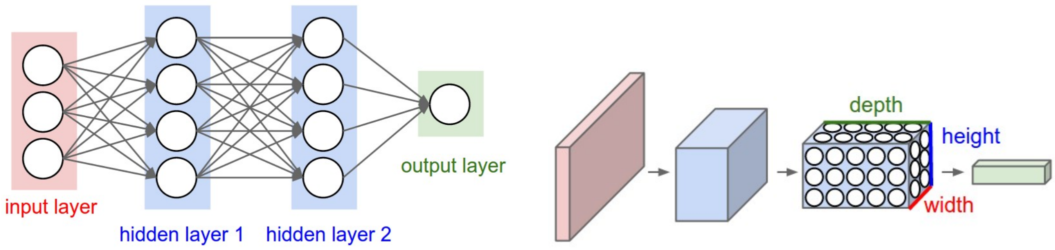
The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

2 Convolutional Neural Networks (CNNs / ConvNets)

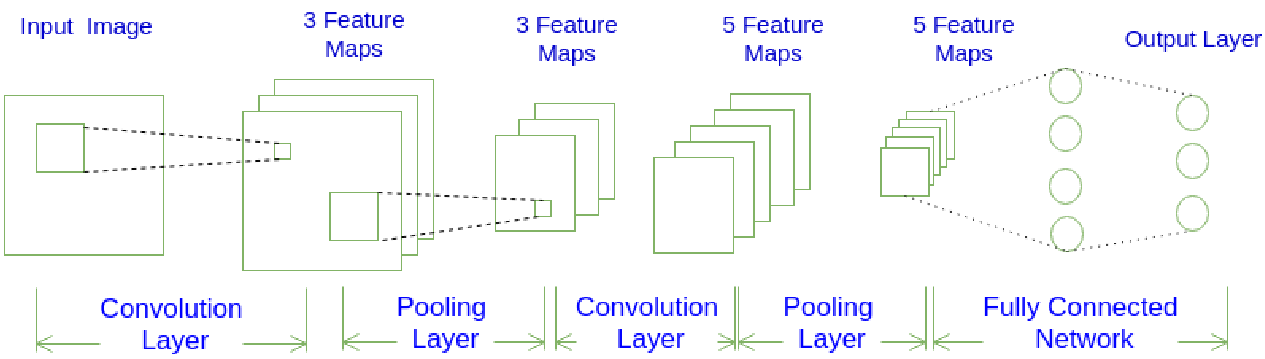
2.1 Architecture Overview

Regular Neural Nets don't scale well to full images. In CIFAR-10, images are only of size $32 \times 32 \times 3$ (32 wide, 32 high, 3 color channels), so a single fully-connected neuron in a first hidden layer of a regular Neural Network would have $32 * 32 * 3 = 3072$ weights. This amount still seems manageable, but clearly this fully-connected structure does not scale to larger images. For example, an image of more respectable size, e.g. $200 \times 200 \times 3$, would lead to neurons that have $200*200*3 = 120{,}000$ weights. Moreover, we would almost certainly want to have several such neurons, so the parameters would add up quickly! Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth. (Note that the word depth here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network.) For example, the input images in CIFAR-10 are an input volume of activations, and the volume has dimensions $32 \times 32 \times 3$ (width, height, depth respectively). As we will soon see, the neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would for CIFAR-10 have dimensions $1 \times 1 \times 10$, because by the end of the ConvNet architecture we will reduce the full image into a single vector of class scores, arranged along the depth dimension. Here is a visualization:



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).



2.2 Layers used to build ConvNets

a simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: **Convolutional Layer**, **Pooling Layer**, and **Fully-Connected Layer** (exactly as seen in regular Neural Networks). We will stack these layers to form a full ConvNet architecture.

Example Architecture: Overview. We will go into more details below, but a simple ConvNet for CIFAR-10 classification could have the architecture [**INPUT - CONV - RELU - POOL - FC**]. In more detail:

- INPUT $[32 \times 32 \times 3]$ will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as $[32 \times 32 \times 12]$ if we decided to use 12 filters.
- RELU layer will apply an elementwise activation function, such as the $max(0, x)$ thresholding at zero. This leaves the size of the volume unchanged ($[32 \times 32 \times 12]$).
- POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as $[16 \times 16 \times 12]$.
- FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size $[1 \times 1 \times 10]$, where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

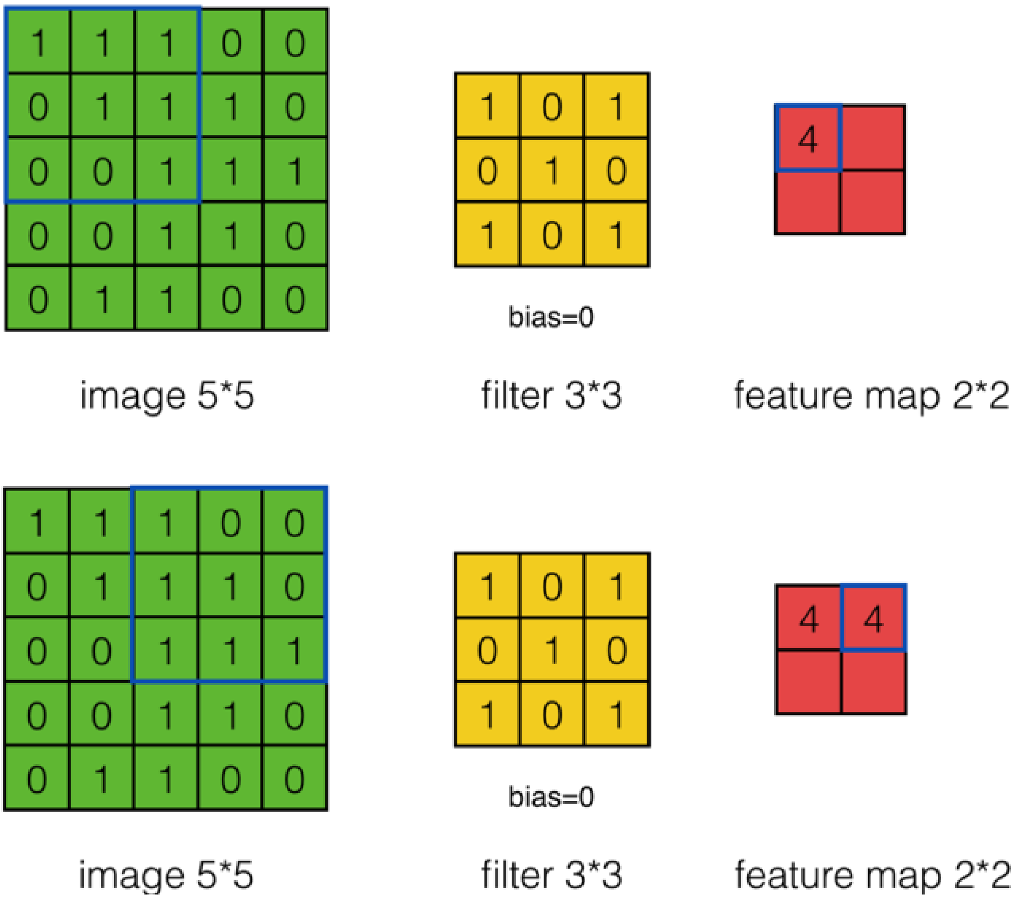
2.2.1 Convolutional Layer

To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.

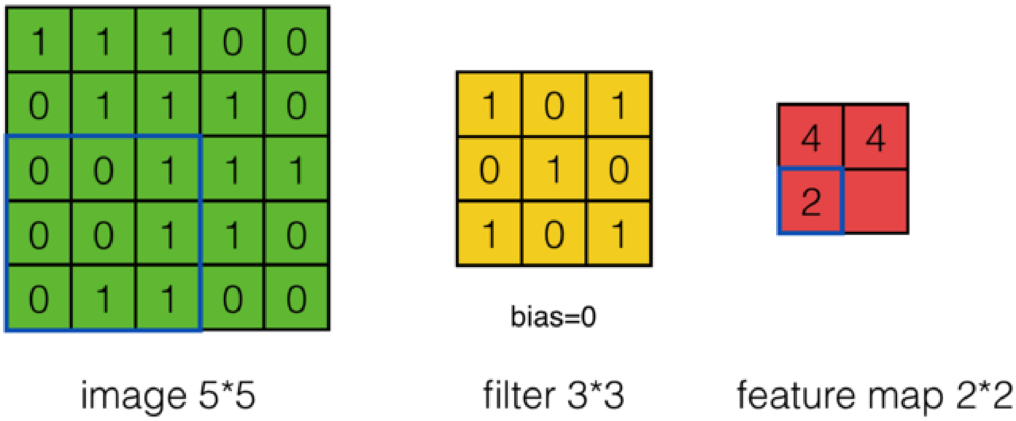
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

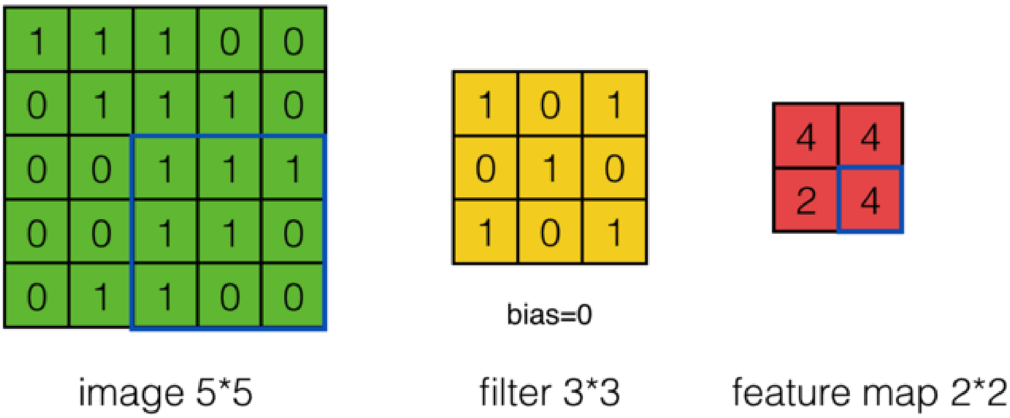
A common setting of the hyperparameters is $F = 3$,$S = 1$,$P = 1$. However, there are common conventions and rules of thumb that motivate these hyperparameters.



2.2.2 Pooling Layer

It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture. Its function is to progressively reduce the spatial size of the representation to reduce the



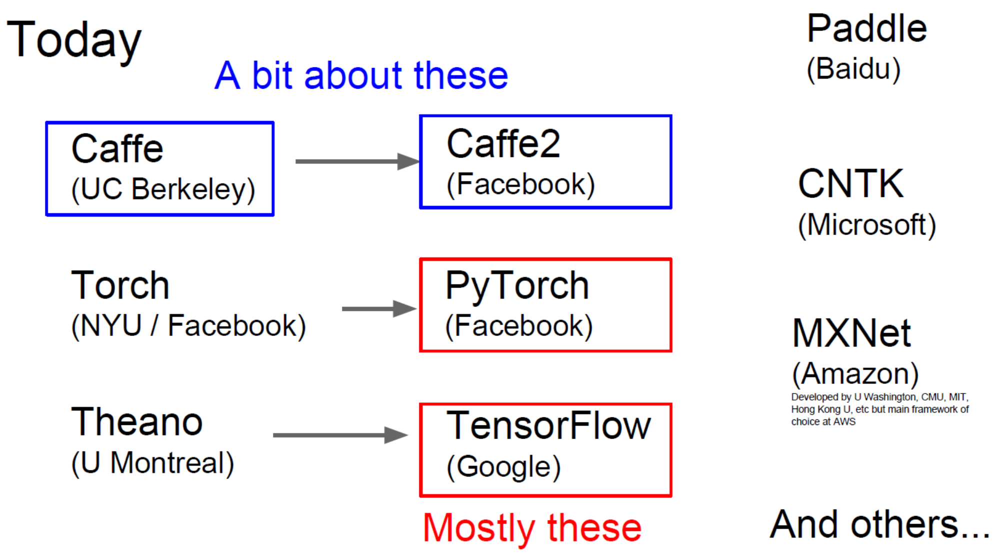


amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the **MAX** operation. The most common form is a pooling layer with filters of size 2×2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations. Every MAX operation would in this case be taking a max over 4 numbers (little 2×2 region in some depth slice). The depth dimension remains unchanged. More generally, the pooling layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D2 = D1$

- Introduces zero parameters since it computes a fixed function of the input
- For Pooling layers, it is not common to pad the input using zero-padding.

3 Deep Learning Softwares



4 Tasks

1. Given the data set in the first section, please implement a convolutional neural network to calculate the accuracy rate. The major steps involved are as follows:
   (a) Reading the input image.
   (b) Preparing filters.
   (c) Conv layer: Convolving each filter with the input image.
   (d) ReLU layer: Applying ReLU activation function on the feature maps (output of conv layer).
   (e) Max Pooling layer: Applying the pooling operation on the output of ReLU layer.
   (f) Stacking conv, ReLU, and max pooling layers
2. You can refer to the codes in `cs231n`. Don't use Keras, TensorFlow, PyTorch, Theano, Caffe, and other deep learning softwares.

5 Codes

```
clear;
[trainingImages,trainingLabels,testImages,testLabels] = helperCIFAR10Data.load('');
numImageCategories = 10;
% Create the image input layer for 32x32x3 CIFAR-10 images.
[height,width,numChannels, ~] = size(trainingImages);

imageSize = [height width numChannels];
inputLayer = imageInputLayer(imageSize)
% Convolutional layer parameters
filterSize = [5 5];
numFilters = 32;

middleLayers = [

% The first convolutional layer has a bank of 32 5x5x3 filters. A
% symmetric padding of 2 pixels is added to ensure that image borders
% are included in the processing. This is important to avoid
% information at the borders being washed away too early in the
% network.
convolution2dLayer(filterSize,numFilters,'Padding',2)

% Note that the third dimension of the filter can be omitted because it
% is automatically deduced based on the connectivity of the network. In
% this case because this layer follows the image layer, the third
% dimension must be 3 to match the number of channels in the input
% image.

% Next add the ReLU layer:
reluLayer()

% Follow it with a max pooling layer that has a 3x3 spatial pooling area
% and a stride of 2 pixels. This down-samples the data dimensions from
% 32x32 to 15x15.
maxPooling2dLayer(3,'Stride',2)

% Repeat the 3 core layers to complete the middle of the network.
convolution2dLayer(filterSize,numFilters,'Padding',2)
reluLayer()
maxPooling2dLayer(3, 'Stride',2)

```

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

]% Convolutional layer parameters
filterSize = [5 5];
numFilters = 32;

middleLayers = [

% The first convolutional layer has a bank of 32 5x5x3 filters. A
% symmetric padding of 2 pixels is added to ensure that image borders
% are included in the processing. This is important to avoid
% information at the borders being washed away too early in the
% network.
convolution2dLayer(filterSize,numFilters,'Padding',2)

% Note that the third dimension of the filter can be omitted because it
% is automatically deduced based on the connectivity of the network. In
% this case because this layer follows the image layer, the third
% dimension must be 3 to match the number of channels in the input
% image.

% Next add the ReLU layer:
reluLayer()

% Follow it with a max pooling layer that has a 3x3 spatial pooling area
% and a stride of 2 pixels. This down-samples the data dimensions from
% 32x32 to 15x15.
maxPooling2dLayer(3,'Stride',2)

% Repeat the 3 core layers to complete the middle of the network.
convolution2dLayer(filterSize,numFilters,'Padding',2)
reluLayer()
maxPooling2dLayer(3, 'Stride',2)

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

]
```

```
finalLayers = [

% Add a fully connected layer with 64 output neurons. The output size of
% this layer will be an array with a length of 64.
fullyConnectedLayer(64)

% Add an ReLU non-linearity.
reluLayer

% Add the last fully connected layer. At this point, the network must
% produce 10 signals that can be used to measure whether the input image
% belongs to one category or another. This measurement is made using the
% subsequent loss layers.
fullyConnectedLayer(numImageCategories)

% Add the softmax loss layer and classification layer. The final layers use
% the output of the fully connected layer to compute the categorical
% probability distribution over the image classes. During the training
% process, all the network weights are tuned to minimize the loss over this
% categorical distribution.
softmaxLayer
classificationLayer
]

layers = [
    inputLayer
    middleLayers
    finalLayers
    ]

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

% Set the 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);

% A trained network is loaded from disk to save time when running the
% example. Set this flag to true to train the network.
doTraining = false;

if doTraining
    % Train a network.
    cifar10Net = trainNetwork(trainingImages, trainingLabels, layers, opts);
else
    % Load pre-trained detector for the example.
    load('rcnnStopSigns.mat','cifar10Net')
end

% Extract the first convolutional layer weights
w = cifar10Net.Layers(2).Weights;

% rescale the weights to the range [0, 1] for better visualization
w = rescale(w);

figure
montage(w)

% Run the network on the test set.
YTest = classify(cifar10Net, testImages);

% Calculate the accuracy.
accuracy = sum(YTest == testLabels)/numel(testLabels)

% Load the ground truth data
data = load('stopSignsAndCars.mat', 'stopSignsAndCars');
stopSignsAndCars = data.stopSignsAndCars;

% Update the path to the image files to match the local file system
visiondata = fullfile(toolboxdir('vision'),'visiondata');
stopSignsAndCars.imageFilename = fullfile(visiondata, stopSignsAndCars.imageFilename);

% Display a summary of the ground truth data
```

```
summary(stopSignsAndCars)

% Only keep the image file names and the stop sign ROI labels
stopSigns = stopSignsAndCars(:, {'imageFilename','stopSign'});

% Display one training image and the ground truth bounding boxes
I = imread(stopSigns.imageFilename{1});
I = insertObjectAnnotation(I,'Rectangle',stopSigns.stopSign{1},'stop sign','LineWidth'
    ,8);

figure
imshow(I)
% A trained detector is loaded from disk to save time when running the
% example. Set this flag to true to train the detector.
doTraining = false;

if doTraining

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

    % Train an R-CNN object detector. This will take several minutes.
    rcnn = trainRCNNObjectDetector(stopSigns, cifar10Net, options, ...
    'NegativeOverlapRange', [0 0.3], 'PositiveOverlapRange',[0.5 1])
else
    % Load pre-trained network for the example.
    load('rcnnStopSigns.mat','rcnn')
end
% Read test image
testImage = imread('stopSignTest.jpg');

% Detect stop signs
[bboxes,score,label] = detect(rcnn,testImage,'MiniBatchSize',128)

```

```
% Display the detection results
[score, idx] = max(score);

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

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

figure
imshow(outputImage)
% The trained network is stored within the R-CNN detector
rcnn.Network


featureMap = activations(rcnn.Network, testImage, 14);

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

stopSignMap = featureMap(:, :, 1);
% Resize stopSignMap for visualization
[height, width, ~] = size(testImage);
stopSignMap = imresize(stopSignMap, [height, width]);

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

figure
imshow(featureMapOnImage)
```

6 Results

7 Reference

- https://www.zybuluo.com/hanbingtao/note/485480
- http://cs231n.github.io/convolutional-networks/#layers

```
inputLayer =

  ImageInputLayer - 属性:

                      Name: ''
                 InputSize: [32 32 3]

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

Figure 1: input layer

```
middleLayers =

  具有以下层的 9x1 Layer 数组:

     1   ''   卷积        32 5x5 卷积: 步幅 [1  1]，填充 [2  2  2  2]
     2   ''   ReLU       ReLU
     3   ''   最大池化     3x3 最大池化: 步幅 [2  2]，填充 [0  0  0  0]
     4   ''   卷积        32 5x5 卷积: 步幅 [1  1]，填充 [2  2  2  2]
     5   ''   ReLU       ReLU
     6   ''   最大池化     3x3 最大池化: 步幅 [2  2]，填充 [0  0  0  0]
     7   ''   卷积        64 5x5 卷积: 步幅 [1  1]，填充 [2  2  2  2]
     8   ''   ReLU       ReLU
     9   ''   最大池化     3x3 最大池化: 步幅 [2  2]，填充 [0  0  0  0]
```

Figure 2: middle layer

```
layers = 

  具有以下层的 15x1 Layer 数组:

     1   ''   图像输入     32x32x3 图像: 'zerocenter' 归一化
     2   ''   卷积         32 5x5 卷积: 步幅 [1  1]，填充 [2  2  2  2]
     3   ''   ReLU        ReLU
     4   ''   最大池化     3x3 最大池化: 步幅 [2  2]，填充 [0  0  0  0]
     5   ''   卷积         32 5x5 卷积: 步幅 [1  1]，填充 [2  2  2  2]
     6   ''   ReLU        ReLU
     7   ''   最大池化     3x3 最大池化: 步幅 [2  2]，填充 [0  0  0  0]
     8   ''   卷积         64 5x5 卷积: 步幅 [1  1]，填充 [2  2  2  2]
     9   ''   ReLU        ReLU
    10   ''   最大池化     3x3 最大池化: 步幅 [2  2]，填充 [0  0  0  0]
    11   ''   全连接       64 全连接层
    12   ''   ReLU        ReLU
    13   ''   全连接       10 全连接层
    14   ''   Softmax     softmax
    15   ''   分类输出     crossentropyex
```

Figure 3: layers

```
accuracy =

    0.7456
```

Figure 4: accuracy