IMAGE PROCESSING USING RCNN CLASSIFIERS

Table of Contents

AUTHOR

S Sai Suryateja Vellore Institute of Technology,Vellore

Load, Store and Explore the data

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

Divide the dataset into Training, Validation and Testing

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

List the image categories

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

```
ans =

  10×1 cell array
```

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

Properties of CIFAR10 dataset

Create the image input layer for 32x32x3 CIFAR-10 images.

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

imageSize = [height width numChannels];
inputLayer = imageInputLayer(imageSize)
```

```
inputLayer =

  ImageInputLayer with properties:

                 Name: ''
            InputSize: [32 32 3]

   Hyperparameters
    DataAugmentation: 'none'
       Normalization: 'zerocenter'
        AverageImage: []
```

CNN Classifier - Middle Layer

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)
]
```

```
middleLayers =

  9x1 Layer 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]
```

Final Layer of CNN

```
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
]
```

```
finalLayers =

  5x1 Layer 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 last layers

```
layers = [
    inputLayer
    middleLayers
    finalLayers
    ]
```

```
layers =

  15x1 Layer 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
```

Initialise first convolution layer weight

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

Train CNN using CIFAR-10 Data

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
    cifar10Net = trainNetwork(trainingImages, trainingLabels, layers,
 opts);
else
    load('rcnnStopSigns.mat','cifar10Net')
end
```

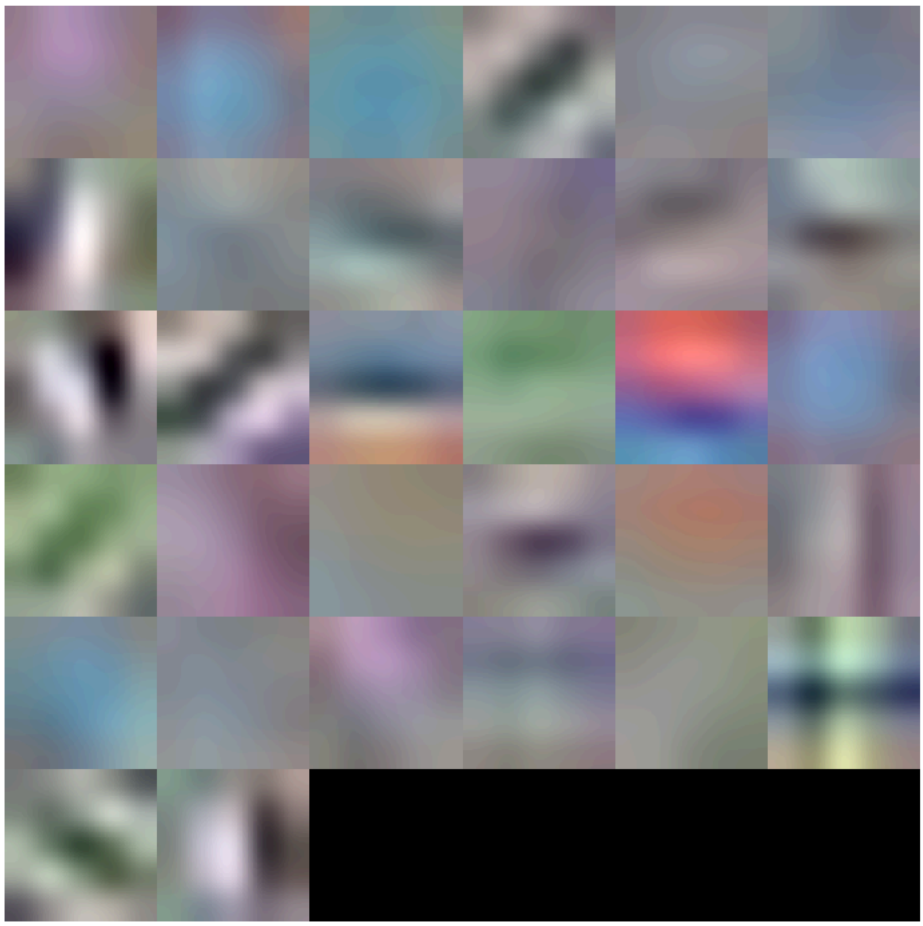
Validate CIFAR-10 Network training

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(1)
montage(w)
% Run the network on the test set.
YTest = classify(cifar10Net, testImages);
% Calculate the accuracy.
accuracy = sum(YTest == testLabels)/numel(testLabels)
```

```
accuracy =

    0.7456
```



Published with MATLAB® R2019a