IMAGE CLASSIFIER

Table of Contents

INTRODUCTION

A Convolutional Neural Network (CNN) is a powerful machine learning technique from the field of deep learning. CNNs are trained using large collections of diverse images. From these large collections, CNNs can learn rich feature representations for a wide range of images. These feature representations often outperform hand-crafted features such as HOG, LBP, or SURF. An easy way to leverage the power of CNNs, without investing time and effort into training, is to use a pretrained CNN as a feature extractor. In this example, images from a Flowers Dataset[5] are classified into categories using a multiclass linear SVM trained with CNN features extracted from the images. This approach to image category classification follows the standard practice of training an off-the-shelf classifier using features extracted from images. For example, the Image Category Classification Using Bag of Features example uses SURF features within a bag of features framework to train a multiclass SVM. The difference here is that instead of using image features such as HOG or SURF, features are extracted using a CNN.

AUTHOR

S Sai Suryateja, Vellore Institute of Technology,Vellore

Note: This example requires Deep Learning Toolbox™, Statistics and Machine Learning Toolbox™, and Deep Learning Toolbox™ Model for ResNet-50 Network . Using a CUDA-capable NVIDIA™ GPU with compute capability 3.0 or higher is highly recommended for running this example. Use of a GPU requires the Parallel Computing Toolbox™.

Location of the compressed data set

```
url = 'http://download.tensorflow.org/example_images/
flower_photos.tgz';
```

Store the output in a temporary folder

```
downloadFolder = tempdir;
filename = fullfile(downloadFolder,'flower_dataset.tgz');
```

Uncompressed data set

```
imageFolder = fullfile(downloadFolder,'flower_photos');

if ~exist(imageFolder,'dir') % download only once
    disp('Downloading Flower Dataset (218 MB)...');
    websave(filename,url);
    untar(filename,downloadFolder)
end
```

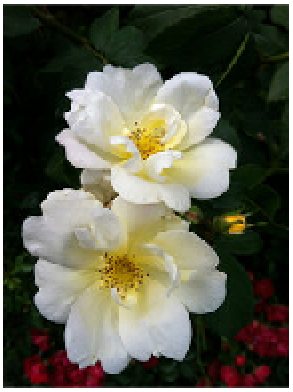
Load the Dataset

```
imds =
 imageDatastore(imageFolder,'LabelSource','foldernames','IncludeSubfolders',true);
```

Find the first instance of an image for each category

```
daisy = find(imds.Labels == 'daisy', 1);
figure(1),imshow(readimage(imds,daisy));
dandelion = find(imds.Labels == 'dandelion', 1);
```

```
figure(2),imshow(readimage(imds,dandelion));
roses = find(imds.Labels == 'roses', 1);
figure(3),imshow(readimage(imds,roses));
sunflowers = find(imds.Labels == 'sunflowers', 1);
figure(4),imshow(readimage(imds,sunflowers));
tulips = find(imds.Labels == 'tulips', 1);
figure(5),imshow(readimage(imds,tulips));
```



Count the amount of images in each category

```
tbl = countEachLabel(imds)
```

```
tbl =

  5×2 table

      Label        Count
    __________     _____

    daisy           633
    dandelion       898
    roses           641
    sunflowers      699
    tulips          799
```

Determine the smallest amount of images in a category

```
minSetCount = min(tbl{:,2});
```

Limit the number of images to reduce the time it takes run this example.

```
maxNumImages = 100;
minSetCount = min(maxNumImages,minSetCount);
```

Use splitEachLabel method to trim the set.

```
imds = splitEachLabel(imds, minSetCount, 'randomize');
```

Notice that each set now has exactly the same number of images.

```
countEachLabel(imds)
```

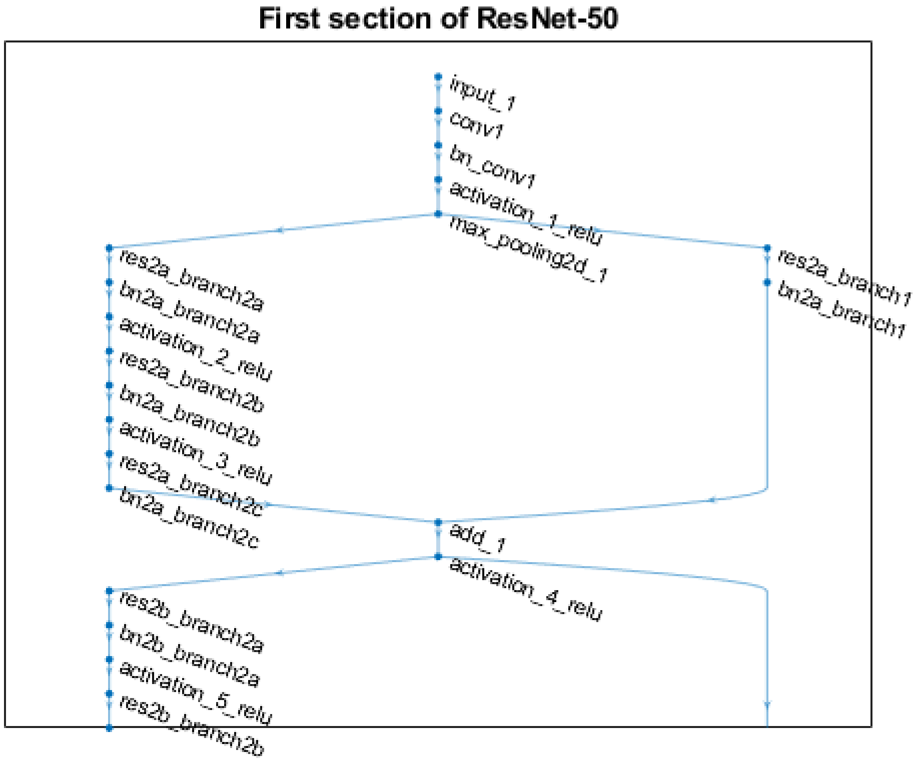
```
ans = 

  5×2 table

      Label        Count
    __________     _____

    daisy           100 
    dandelion       100 
    roses           100 
    sunflowers      100 
    tulips          100 
```

Load pretrained network

```
net = resnet50();
```

Visualize the first section of the network.

```
figure(6),plot(net),title('First section of
 ResNet-50'),set(gca,'YLim',[150 170]);
```



Inspect the first layer

```
net.Layers(1)
```

```
ans = 

  ImageInputLayer with properties:

                Name: 'input_1'
           InputSize: [224 224 3]

   Hyperparameters
    DataAugmentation: 'none'
       Normalization: 'zerocenter'
        AverageImage: [224×224×3 single]
```

Inspect the last layer

```
net.Layers(end)
```

```
ans = 
```

```
  ClassificationOutputLayer with properties:

            Name: 'ClassificationLayer_fc1000'
         Classes: [1000×1 categorical]
      OutputSize: 1000

   Hyperparameters
    LossFunction: 'crossentropyex'
```

Number of class names for ImageNet classification task

```
numel(net.Layers(end).ClassNames)
```

```
ans =

        1000
```

Split the data for Training and Validation

```
[trainingSet, testSet] = splitEachLabel(imds, 0.3, 'randomize');
```

Create augmentedImageDatastore from training and test sets to resize images in imds to the size required by the network.

```
imageSize = net.Layers(1).InputSize;
augmentedTrainingSet = augmentedImageDatastore(imageSize,
 trainingSet, 'ColorPreprocessing', 'gray2rgb');
augmentedTestSet = augmentedImageDatastore(imageSize,
 testSet, 'ColorPreprocessing', 'gray2rgb');
```

Get the network weights for the second convolutional layer

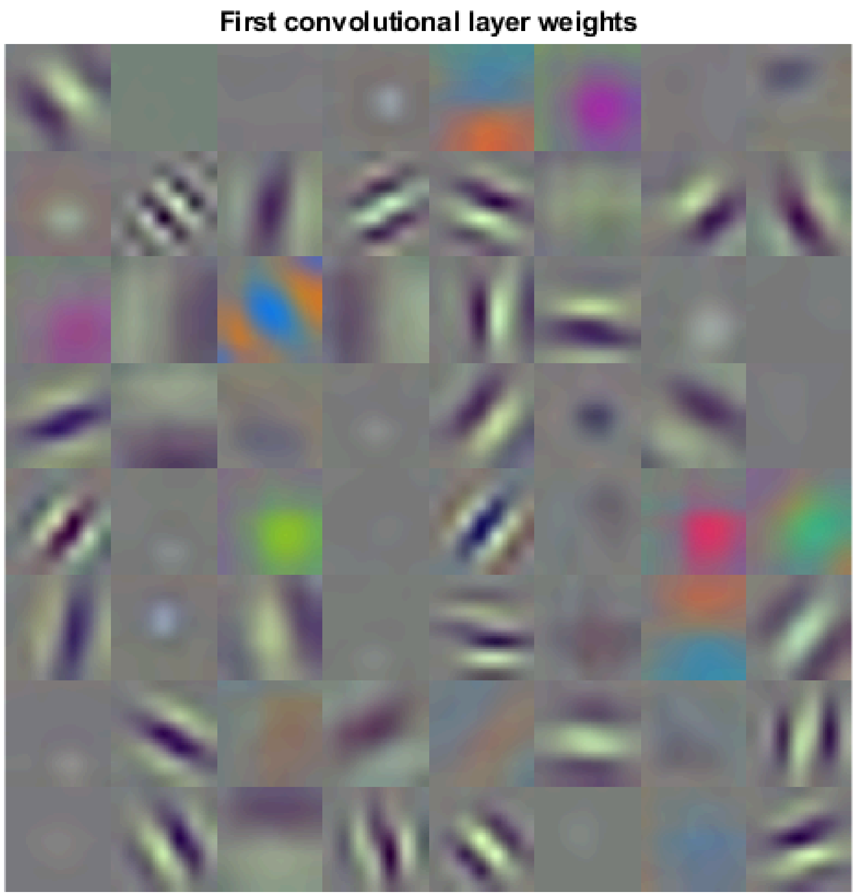
```
w1 = net.Layers(2).Weights;
```

Scale and resize the weights for visualization

```
w1 = mat2gray(w1);
w1 = imresize(w1,5);
```

Display a montage of network weights. There are 96 individual sets of weights in the first layer.

```
figure(7),montage(w1),title('First convolutional layer weights');
featureLayer = 'fc1000';
trainingFeatures = activations(net, augmentedTrainingSet,
 featureLayer, ...
    'MiniBatchSize', 32, 'OutputAs', 'columns');
```



Get training labels from the trainingSet

```
trainingLabels = trainingSet.Labels;
```

Train multiclass SVM classifier using a fast linear solver, and set 'ObservationsIn' to 'columns' to match the arrangement used for training features.

```
classifier = fitcecoc(trainingFeatures, trainingLabels, ...

  'Learners', 'Linear', 'Coding', 'onevsall', 'ObservationsIn', 'columns');
```

Extract test features using the first layer

```
testFeatures = activations(net, augmentedTestSet, featureLayer, ...
    'MiniBatchSize', 32, 'OutputAs', 'columns');
```

Pass CNN image features to trained classifier

```
predictedLabels = predict(classifier,
 testFeatures, 'ObservationsIn', 'columns');
```

Get the known labels

```
testLabels = testSet.Labels;
```

Tabulate the results using a confusion matrix.

```
confMat = confusionmat(testLabels, predictedLabels);
```

Convert confusion matrix into percentage form

```
confMat = bsxfun(@rdivide,confMat,sum(confMat,2))
```

```
confMat =

    0.8714    0.0286    0.0571    0.0286    0.0143
    0.0714    0.7571    0.0143    0.1286    0.0286
    0.0143    0.0286    0.7857    0.0429    0.1286
         0    0.0143    0.0571    0.9286         0
         0         0    0.0571    0.0429    0.9000
```

Display the mean accuracy

```
mean(diag(confMat))
testImage = readimage(testSet,1);
testLabel = testSet.Labels(1)
```

```
ans =

    0.8486


testLabel =

  categorical

     daisy
```

Create augmentedImageDatastore to automatically resize the image when image features are extracted using activations.

```
ds = augmentedImageDatastore(imageSize,
 testImage, 'ColorPreprocessing', 'gray2rgb');
```

Extract image features using the CNN

```
imageFeatures = activations(net, ds,
 featureLayer, 'OutputAs', 'columns');
```

Make a prediction using the classifier

```
predictedLabel = predict(classifier,
 imageFeatures, 'ObservationsIn', 'columns')


predictedLabel =

  categorical

     daisy
```

Published with MATLAB® R2019a