CNN Classifier

Giovanni Battilana, Marzia Paschini, Marco Sgobino

January 6, 2022

Contents

1 Assignment general description

This project requires the implementation of an image classifier based on convolutional neural networks. The provided dataset (from [Lazebnik et al., 2006]), contains 15 categories (office, kitchen, living room, bedroom, store, industrial, tall building, inside city, street, highway, coast, open country, mountain, forest, suburb), and is already divided in training set and test set.

2 Goal of this paper

This paper had 3 goals of increasing difficulty:

1. building a *shallow network* with a layout specified from the Teacher in order to have a *baseline* network to allow further comparisons. Such baseline should be improved in the subsequent sections;

2. improving the previous baseline network and compare results with the baseline. Comments on strategies adopted should be provided;

3. adopting *transfer learning* based on the pre-trained network *AlexNet*.

3 Adopted tools

Students adopted the programming language and numeric computing environment **MATLAB** as both text editor and simulation software to describe, implement and build the trained networks.

Specific MATLAB Toolbox were needed – the project should require *Deep Learning Toolbox*, *Computer Vision Toolbox* and *Image Processing Toolbox*. Optionally, to improve processing speed during the convolutional neural network build phase, students installed and enabled the *Parallel Computing Toolbox*.

4 Training a baseline network

The first step was the building a baseline network (a shallow network), whose layout had been assigned by the teacher. In practice, this phase required to strictly follow given requirements in order to obtain *an overall test accuracy of around* 30%.

4.1 Importing the dataset

The training dataset comprises 1500 images, of 15 categories (office, kitchen, living room, bedroom, store, industrial, tall building, inside city, street, highway, coast, open country, mountain, forest, suburb). Each photo lies inside a directory, whose name denotes the category. Similarly, the test dataset comprises 2985 pictures lying in the same categories.
In order to import the dataset into an *image datastore* object, students used following instructions,

```
TrainDatasetPath = fullfile('dataset',
    'train');

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

Each image in the dataset had its own – non common – size. Moreover, images do not share a common aspect ratio. In order to obtain a shared size and aspect ratio, students have chosen to follow the simple approach of *rescaling the whole image independently along the vertical and the horizontal axis*, in order to achieve the proper size. As will be shown later, the chosen aspect ratio will be 1 : 1 (a square).

4.2 Requirements

Layout of the network is described in the following Table:

#	type	size
1	Image input	$64 \times 64 \times 1$ images
2	Convolution	8 3×3 convolutions, stride 1
3	ReLU	
4	Max Pooling	2×2 max pooling, stride 2
5	Convolution	16 3×3 convolutions, stride 1
6	ReLU	
7	Max Pooling	2×2 max pooling, stride 2
8	Convolution	32 3×3 convolutions, stride 1
9	ReLU	
10	Fully Connected	15
11	Softmax	softmax
12	Classification	crossentropyex

Other non-optional requirements were the following ones,

- using input images of size 64×64 (as shown in layer 1 of previous Table);
- use 85% of the provided dataset for the training set and the remaining portion for the validation set, so that each label is properly split with the same percentage;
- employ **stochastic gradient descent with momentum** as the optimization algorithm;
- adopt *minibatches* of size 32;
- initial weights should be drawn from a Gaussian distribution with a mean of 0 and a standard deviation of 0.01. Initial bias values should be set to 0;
- choose a stopping criterion;

4.3 Splitting the dataset

To split the dataset as requested, students wrote and ran the following code:

```
trainQuota=0.85;
[imdsTrain, imdsValidation] =
    splitEachLabel(imds, trainQuota,...
    'randomize');
```

These instructions were sufficient to obtain two, distinct, datasets – the first one for training, the second one for validation. Images should be elected and put in the datasets according to a random process; despite that, `splitEachLabel` function should assure that the same quota of 0.85 for training sets was maintained across all 15 labels.

4.4 Resizing images

In order to resize images to match the requested size of 64×64, students ran:

```
imds.ReadFcn = @(x)imresize(imread(x),...
                            [64 64]);
```

This command should overload the `ReadFcn` function inside `imds`, so that instead of simply reading images they should be resized too.

4.5 Layers definition

Layers layout is specified in the provided Table. In order to obtain such layout, the students organized the network layout information in a single object named `layers`, such as

```
layers = [
    imageInputLayer([64 64 1],'Name','input')

    convolution2dLayer(3,8,...
        'Padding','same',...
        'Stride', [1 1],...
        'Name','conv_1',...
        'WeightsInitializer',...
        @(sz) randn(sz)*0.01,...
        'BiasInitializer',...
        @(sz) zeros(sz))

    reluLayer('Name','relu_1')

    maxPooling2dLayer(2,'Stride',2,...
        'Name','maxpool_1')

    convolution2dLayer(3,16,...
        'Padding','same',...
        'Stride', [1 1],...
        'Name','conv_2',...
        'WeightsInitializer',...
        @(sz) randn(sz)*0.01,...
        'BiasInitializer',...
        @(sz) zeros(sz))

    reluLayer('Name','relu_2')

    maxPooling2dLayer(2,'Stride',2,...
        'Name','maxpool_2')

    convolution2dLayer(3,32,...
        'Padding','same',...
        'Stride', [1 1],...
        'Name','conv_3',...
        'WeightsInitializer',...
        @(sz) randn(sz)*0.01,...
        'BiasInitializer',...
        @(sz) zeros(sz))

    reluLayer('Name','relu_3')

    fullyConnectedLayer(15,...
        'Name','fc_1',...
        'WeightsInitializer',...
        @(sz) randn(sz)*0.01,...
        'BiasInitializer',...
        @(sz) zeros(sz))

    softmaxLayer('Name','softmax')
    classificationLayer('Name','output')
];
```

A peculiar aspects of this representation is the weights initialization, which is handled by the function handle `@(sz) randn(sz)*0.01`, which should yield proper random numbers for normally-distributed weights initialization with standard deviation 0.01.
Similarly, `@(sz) zeros(sz)*0.01` will generate the null vectors required to initialize the biases.
A name should be assigned to each layer, while the defined network may be visually inspected with the commands

```
lgraph = layerGraph(layers);
analyzeNetwork(lgraph)
```

4.6 Network training options

Options for training the network should be provided. In order to satisfy the requirements, students set the following options in an object,

```
options = trainingOptions('sgdm', ...
    'InitialLearnRate', InitialLearningRate, ...
    'ValidationData',imdsValidation, ...
    'MiniBatchSize',32, ...
    'ExecutionEnvironment','parallel',...
    'Plots','training-progress'...
);
```

where the quantity `InitialLearningRate` should be properly fine-tuned.
Each option is necessary, with the sole exception of `ExecutionEnvironment`, which enables parallel computing capabilities:

- `sgdm`: the optimization algorithm in use, as in requirements;
- `ValidationData`: specifies the image datastore to use when validating the error, during the

training;

- `MiniBatchSize`: set to 32 as in requirements;
- `Plots`: enables a visual inspection of the training process, useful to spot possible signs of overfitting;

Every other option is left as default.

4.7 Training the network and obtaining the accuracy

The following command will begin the network training with specified layout and options,

```
net = trainNetwork(imdsTrain, layers, options);
```

In order to collect the accuracy with the provided test set, the students implemented the following code,

```
TestDatasetPath = fullfile('dataset', 'test');
imdsTest = imageDatastore(TestDatasetPath, ...
    'IncludeSubfolders', true,...
    'LabelSource', 'foldernames');
imdsTest.ReadFcn = @(x)imresize(imread(x),...
                  [64 64]);

YPredicted = classify(net, imdsTest);
YTest = imdsTest.Labels;

accuracy = sum(YPredicted == YTest) /
    numel(YTest);
```

TODO

4.8 Fine-tuning initial learning rate

In order to fine-tune the initial learning rate, manual trials along with manual inspection was required. Values of 0.1, 0.01, 0.001 and 0.0001 (representatives of various order of magnitude) were tried.

Only values in the neighborhood of 0.001 gave the required performance of around 30%. In particular, testing the network 5 times, and collecting the accuracy TODO

5 Improving the network

6 Adopting transfer learning with AlexNet