CNN Classifier

Giovanni Battilana, Marzia Paschini, Marco Sgobino

January 10, 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 the reference from which subsequent improvements are evaluated;

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*. Comments on performance improvements should be given.

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, the group has 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 of 1 : 1 (a square).

4.2 Requirements

Layout of the network is described in the following Table:

#	layer type	size and parameters
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)` 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);
```

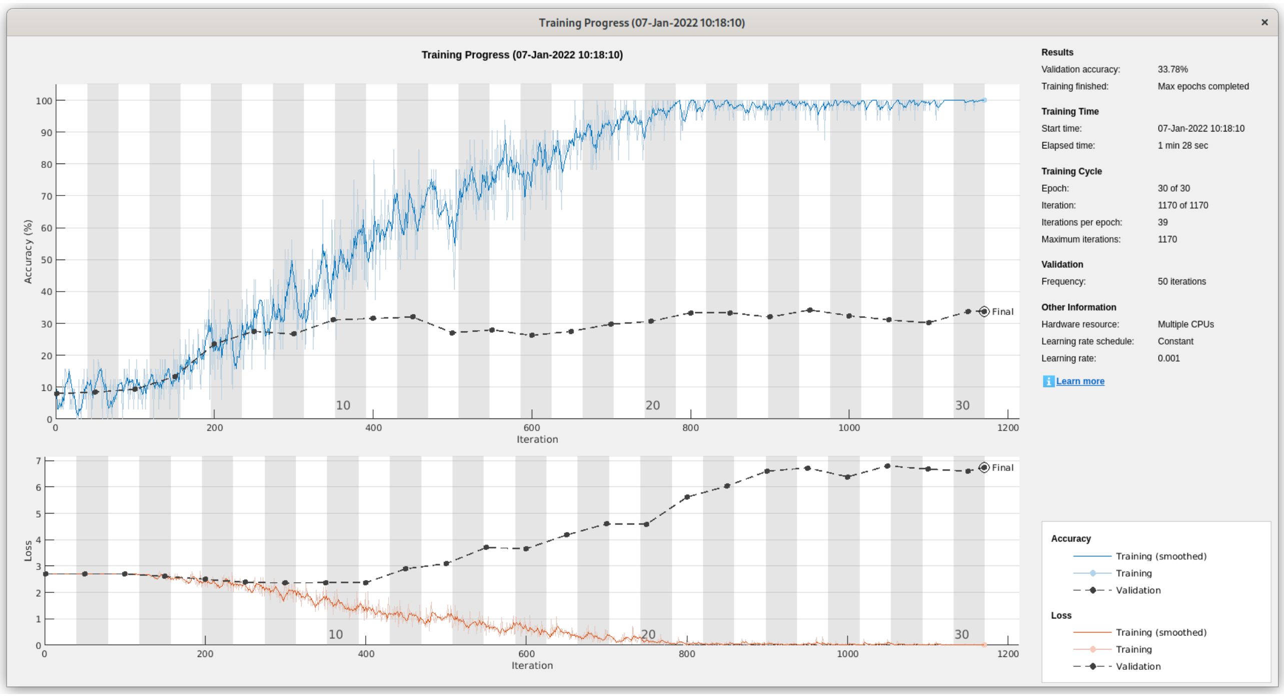
4.8 Choice of `MaxEpoch` and stopping criterion

Training the network with an initial learning rate of 0.001 and leaving the default number of epochs unaltered led to evident overfitting of the network, as shown in Figure 1.
With the goal of avoiding overfitting, a lower `MaxEpoch` value (for instance 8) should be adopted.

4.9 Fine-tuning initial learning rate

Fine-tune the initial learning rate required manual trials along with manual inspection. 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%. Since the default `MaxEpoch` value of 30 led to overfitting, a lower number should be set.
In particular, testing the network 5 times, with an optimal initial learning rate of 0.0005, 0.001, 0.0015 and 8 epochs, collecting the accuracy of each run and averaging the results returned the following Table,

Initial learning rate	Average of validation accuracy	Average of test set accuracy	Epochs
0.0005	0.2258	0.1970	8
0.001	0.2766	0.2623	8
0.0015	0.2800	0.2919	8



Results	
Validation accuracy:	33.78%
Training finished:	Max epochs completed
Training Time	
Start time:	07-Jan-2022 10:18:10
Elapsed time:	1 min 28 sec
Training Cycle	
Epoch:	30 of 30
Iteration:	1170 of 1170
Iterations per epoch:	39
Maximum iterations:	1170
Validation	
Frequency:	50 iterations
Other Information	
Hardware resource:	Multiple CPUs
Learning rate schedule:	Constant
Learning rate:	0.001

Figure 1: Overfitting of the network when `MaxEpoch` is the default value of 30.

Apparently, the baseline network of choice should be the third one with an initial learning rate of 0.0015. However, the comparison is unfair since that network manifests signs of overfitting beginning with the Epoch 8. To compare those two networks the group decided to lower the number of Epochs for the third network, and repeat the training:

Initial learning rate	Average of validation accuracy	Average of test set accuracy	Epochs
0.001	0.2766	0.2623	8
0.0015	0.2800	0.2664	7

where in this case the choice makes no real difference with respect to the average of test set accuracy. To keep the baseline as simple as possible, the students decided to elect the first network as the *baseline*, with parameters as following:

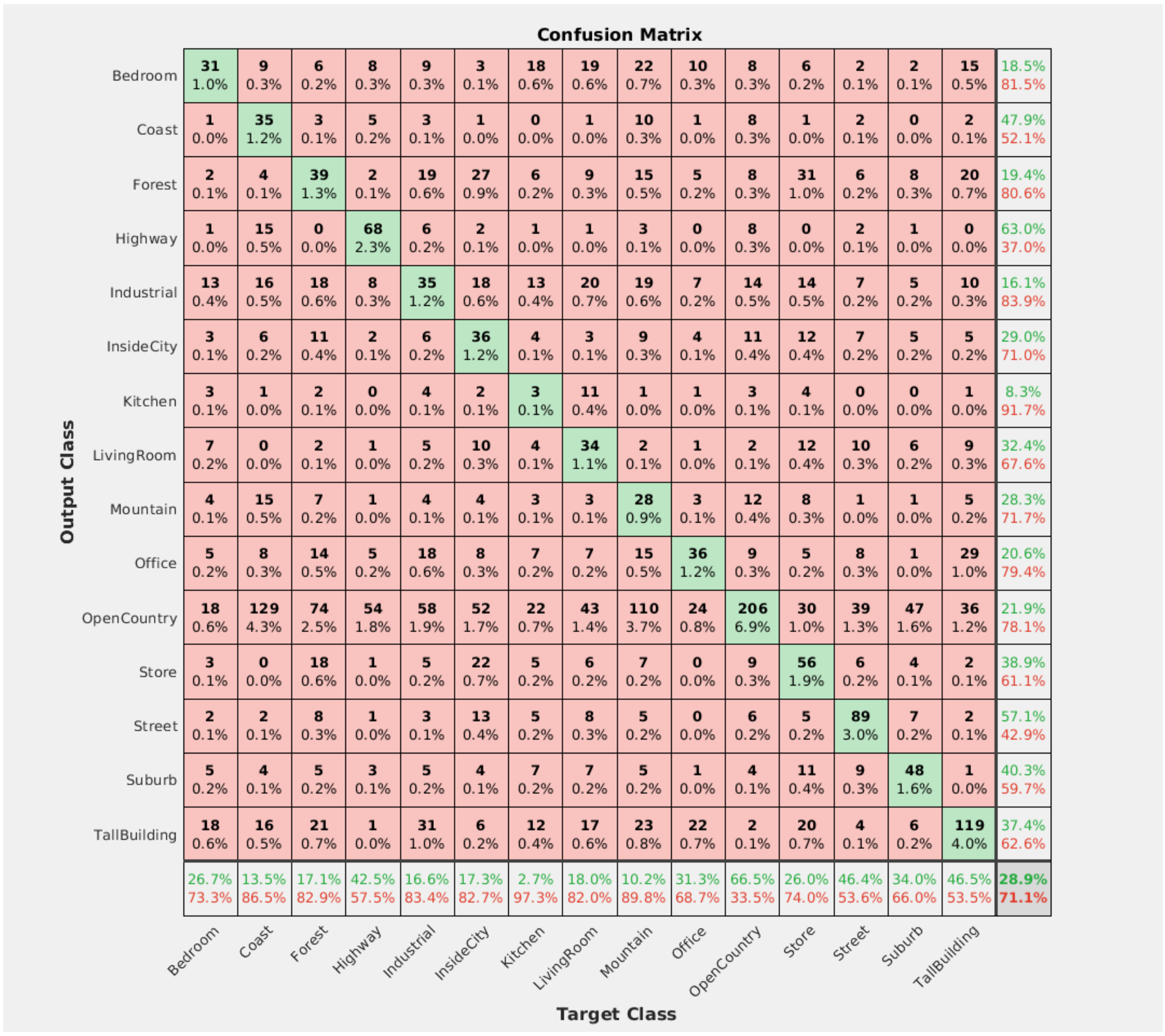
Baseline parameter	value
Optimization Algorithm	sgdm
Epochs (stopping criterion)	8
Initial Learning Rate	0.001
MiniBatch size	32
Weights initialization	Gaussian with $\mu = 0$ and $\sigma = 0.01$
Bias initialization	0

4.10 Baseline confusion matrix

To obtain the *confusion matrix* for the chosen baseline network, the students ran the following code to first train the network again, and then plot a confusion matrix as shown in Figure 2,

```
InitialLearningRate = 0.001;
options = trainingOptions('sgdm', ...
    'InitialLearnRate', InitialLearningRate, ...
    'ValidationData',imdsValidation, ...
    'MiniBatchSize',32, ...
    'MaxEpochs', 8,...
    'ExecutionEnvironment','parallel',...
    'Plots','training-progress'...
);

net = trainNetwork(imdsTrain,layers,options);

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);

figure
plotconfusion(YTest,YPredicted)
```

The confusion matrix shows that there are some classes that are more often misclassified (`kitchen`, `industrial`) and some classes that instead are less often misclassified, such as `highway` and `street`. Overall accuracy stands at the value 28.9%: from this point of view, the baseline should represent an improvement from a completely random classifier, whose performance with respect to this classification task is less than a 7%.



	Bedroom	Coast	Forest	Highway	Industrial	InsideCity	Kitchen	LivingRoom	Mountain	Office	OpenCountry	Store	Street	Suburb	TallBuilding	
Bedroom	31 1.0%	9 0.3%	6 0.2%	8 0.3%	9 0.3%	3 0.1%	18 0.6%	19 0.6%	22 0.7%	10 0.3%	8 0.3%	6 0.2%	2 0.1%	2 0.1%	15 0.5%	18.5% 81.5%
Coast	1 0.0%	35 1.2%	3 0.1%	5 0.2%	3 0.1%	1 0.0%	0 0.0%	1 0.0%	10 0.3%	1 0.0%	8 0.3%	1 0.0%	2 0.1%	0 0.0%	2 0.1%	47.9% 52.1%
Forest	2 0.1%	4 0.1%	39 1.3%	2 0.1%	19 0.6%	27 0.9%	6 0.2%	9 0.3%	15 0.5%	5 0.2%	8 0.3%	31 1.0%	6 0.2%	8 0.3%	20 0.7%	19.4% 80.6%
Highway	1 0.0%	15 0.5%	0 0.0%	68 2.3%	6 0.2%	2 0.1%	1 0.0%	1 0.0%	3 0.1%	0 0.0%	8 0.3%	0 0.0%	2 0.1%	1 0.0%	0 0.0%	63.0% 37.0%
Industrial	13 0.4%	16 0.5%	18 0.6%	8 0.3%	35 1.2%	18 0.6%	13 0.4%	20 0.7%	19 0.6%	7 0.2%	14 0.5%	14 0.5%	7 0.2%	5 0.2%	10 0.3%	16.1% 83.9%
InsideCity	3 0.1%	6 0.2%	11 0.4%	2 0.1%	6 0.2%	36 1.2%	4 0.1%	3 0.1%	9 0.3%	4 0.1%	11 0.4%	12 0.4%	7 0.2%	5 0.2%	5 0.2%	29.0% 71.0%
Kitchen	3 0.1%	1 0.0%	2 0.1%	0 0.0%	4 0.1%	2 0.1%	3 0.1%	11 0.4%	1 0.0%	1 0.0%	3 0.1%	4 0.1%	0 0.0%	0 0.0%	1 0.0%	8.3% 91.7%
LivingRoom	7 0.2%	0 0.0%	2 0.1%	1 0.0%	5 0.2%	10 0.3%	4 0.1%	34 1.1%	2 0.1%	1 0.0%	2 0.1%	12 0.4%	10 0.3%	6 0.2%	9 0.3%	32.4% 67.6%
Mountain	4 0.1%	15 0.5%	7 0.2%	1 0.0%	4 0.1%	4 0.1%	3 0.1%	3 0.1%	28 0.9%	3 0.1%	12 0.4%	8 0.3%	1 0.0%	1 0.0%	5 0.2%	28.3% 71.7%
Office	5 0.2%	8 0.3%	14 0.5%	5 0.2%	18 0.6%	8 0.3%	7 0.2%	7 0.2%	15 0.5%	36 1.2%	9 0.3%	5 0.2%	8 0.3%	1 0.0%	29 1.0%	20.6% 79.4%
OpenCountry	18 0.6%	129 4.3%	74 2.5%	54 1.8%	58 1.9%	52 1.7%	22 0.7%	43 1.4%	110 3.7%	24 0.8%	206 6.9%	30 1.0%	39 1.3%	47 1.6%	36 1.2%	21.9% 78.1%
Store	3 0.1%	0 0.0%	18 0.6%	1 0.0%	5 0.2%	22 0.7%	5 0.2%	6 0.2%	7 0.2%	0 0.0%	9 0.3%	56 1.9%	6 0.2%	4 0.1%	2 0.1%	38.9% 61.1%
Street	2 0.1%	2 0.1%	8 0.3%	1 0.0%	3 0.1%	13 0.4%	5 0.2%	8 0.3%	5 0.2%	0 0.0%	6 0.2%	5 0.2%	89 3.0%	7 0.2%	2 0.1%	57.1% 42.9%
Suburb	5 0.2%	4 0.1%	5 0.2%	3 0.1%	5 0.2%	4 0.1%	7 0.2%	7 0.2%	5 0.2%	1 0.0%	4 0.1%	11 0.4%	9 0.3%	48 1.6%	1 0.0%	40.3% 59.7%
TallBuilding	18 0.6%	16 0.5%	21 0.7%	1 0.0%	31 1.0%	6 0.2%	12 0.4%	17 0.6%	23 0.8%	22 0.7%	2 0.1%	20 0.7%	4 0.1%	6 0.2%	119 4.0%	37.4% 62.6%
	26.7% 73.3%	13.5% 86.5%	17.1% 82.9%	42.5% 57.5%	16.6% 83.4%	17.3% 82.7%	2.7% 97.3%	18.0% 82.0%	10.2% 89.8%	31.3% 68.7%	66.5% 33.5%	26.0% 74.0%	46.4% 53.6%	34.0% 66.0%	46.5% 53.5%	**28.9% 71.1%**

Figure 2: Confusion matrix for the baseline network.

5 Improving the network

The students propose (and have implemented) various methods in order to improve the performance of the Convolutional Neural Network from the baseline ~ 30% to a 60%:

1. *data augmentation*;
2. addition of *batch normalization layers*;
3. changing the *size of the convolutional filters*;
4. use of *dropout layers*.

Individual contribution and net improvement of each of these methods is not investigated – what is provided, instead, is a description of each technique's implementation along with the necessary details, a brief description of the increasing improvements *at each added complexity* and a final overview of all improvements altogether with respect to the baseline.

5.1 Data augmentation

The first improvement of the network is related to *data augmentation*. The students implemented *left-to-right reflections* of the train set, leaving unaltered the test set required for evaluation.
Running code

```
aug = imageDataAugmenter("RandXReflection",true);
imageSize = [64 64 1];
auimds = augmentedImageDatastore(imageSize,imdsTrain,'DataAugmentation',aug);
aunet = trainNetwork(auimds,layers,options);
```

with the same options as the baseline, but `MaxEpoch` set to 15 (this kind of training resulted somehow to be slower), returned a validation accuracy of around ~ 39%. Hence, the added complexity represented a slight improvement from the previous step.
Here are reported pictures of both training progress and confusion matrix,
TODO FIGURES

5.2 Batch normalization

The second improvement should consist in adding 3 **Batch Normalization layers** before the ReLU layers. Each layer should be added into `layers` object, by adding line

```
batchNormalizationLayer('Name','BN_1')
```

before any ReLU layer. The name of the new layers should be changed accordingly to the place of those layers.
The added complexity resulted in a faster learning (a lower MaxEpoch number should be set, from 15 to 6) and a much higher accuracy (51.11% on validation set, 48.7% on test data).
TODO FIGURES

5.3 Improving convolution layers

Convolution layers may be improved by increasing the size of the filter as they move towards the output. In practice, this means modifying the convolution layers such that their size increases from 3×3 to 5×5 and 7×7,

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

[...]

convolution2dLayer(5,16,'Padding','same','Stride', [1 1], 'Name','conv_2',...
    'WeightsInitializer',@(sz) randn(sz)*0.01,...
    'BiasInitializer', @(sz) zeros(sz))

[...]

 convolution2dLayer(7,32,'Padding','same','Stride', [1 1], 'Name','conv_3',...
    'WeightsInitializer', @(sz) randn(sz)*0.01,...
    'BiasInitializer', @(sz) zeros(sz))
```

This third addition brought a slight improvement, as the validation accuracy is at 55.11%, but the overall test accuracy stands at 50%.
TODO FIGURES

5.4 Modifying parameters

Editing the parameters should improve the performance of the network.
Parameters that should be subjected to optimization are, *Minibatch Size* (default is 32), *Initial learning rate* (which was previously set to 0.001), *optimization algorithm* ('sgdm' or 'adam'). Consequently, *MaxEpoch* should be set accordingly, with the goal of avoiding overfitting while leaving the network room for enough complexity.

5.5 Adding fully-connected and dropout layers

5.5.1 Minibatch size

5.5.2 Initial learning rate

5.5.3 Optimization algorithm

5.5.4 MaxEpoch

5.6 Adding convolutional layer

6 Adopting transfer learning with AlexNet