CNN Classifier

Giovanni Battilana, Marzia Paschini, Marco Sgobino

January 12, 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:

Baseline Layout	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

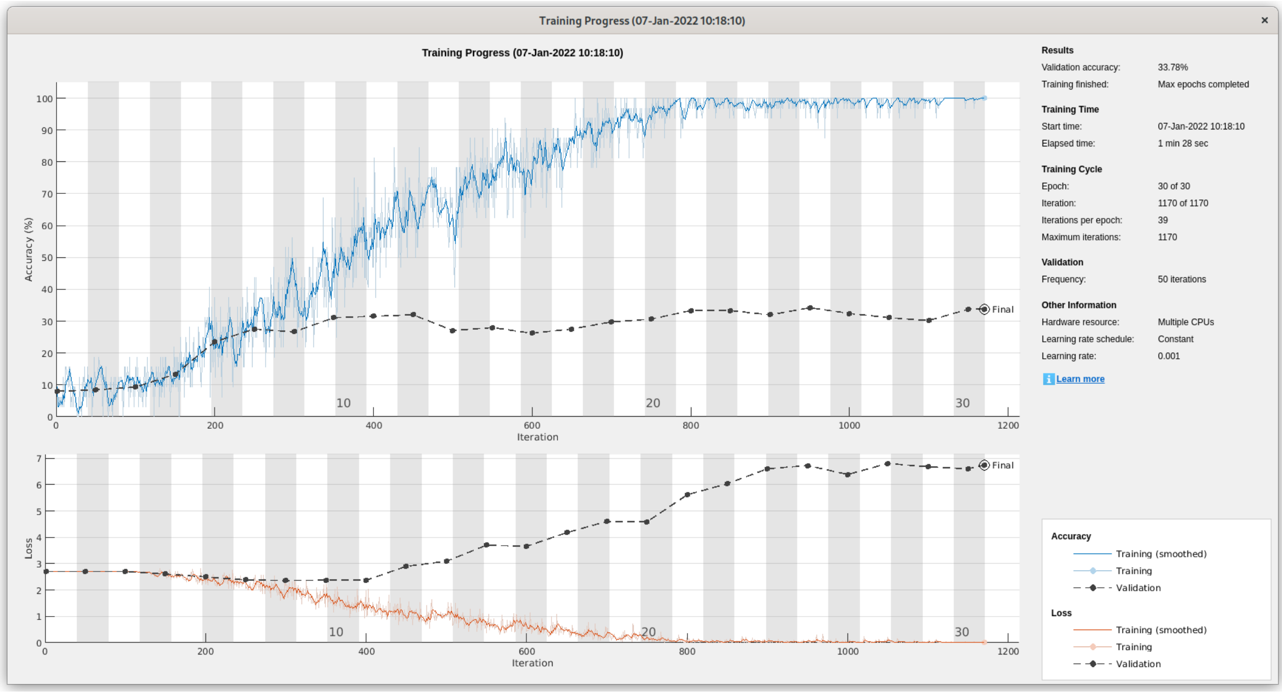

Figure 1: Overfitting of the network when `MaxEpoch` is the default value of 30.

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 training progress and confusion matrix

The overall training progress for the baseline network is shown in Figure 2.

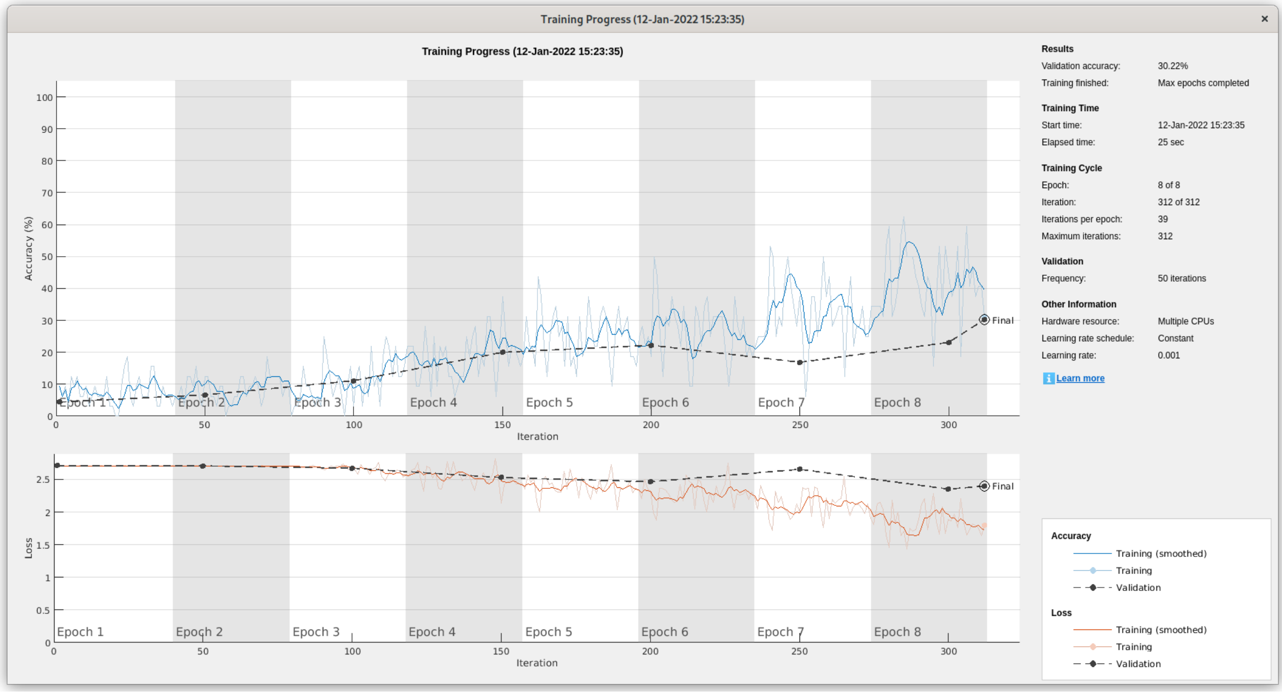

Figure 2: Training progress for the baseline network.

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 3,

```
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)
```

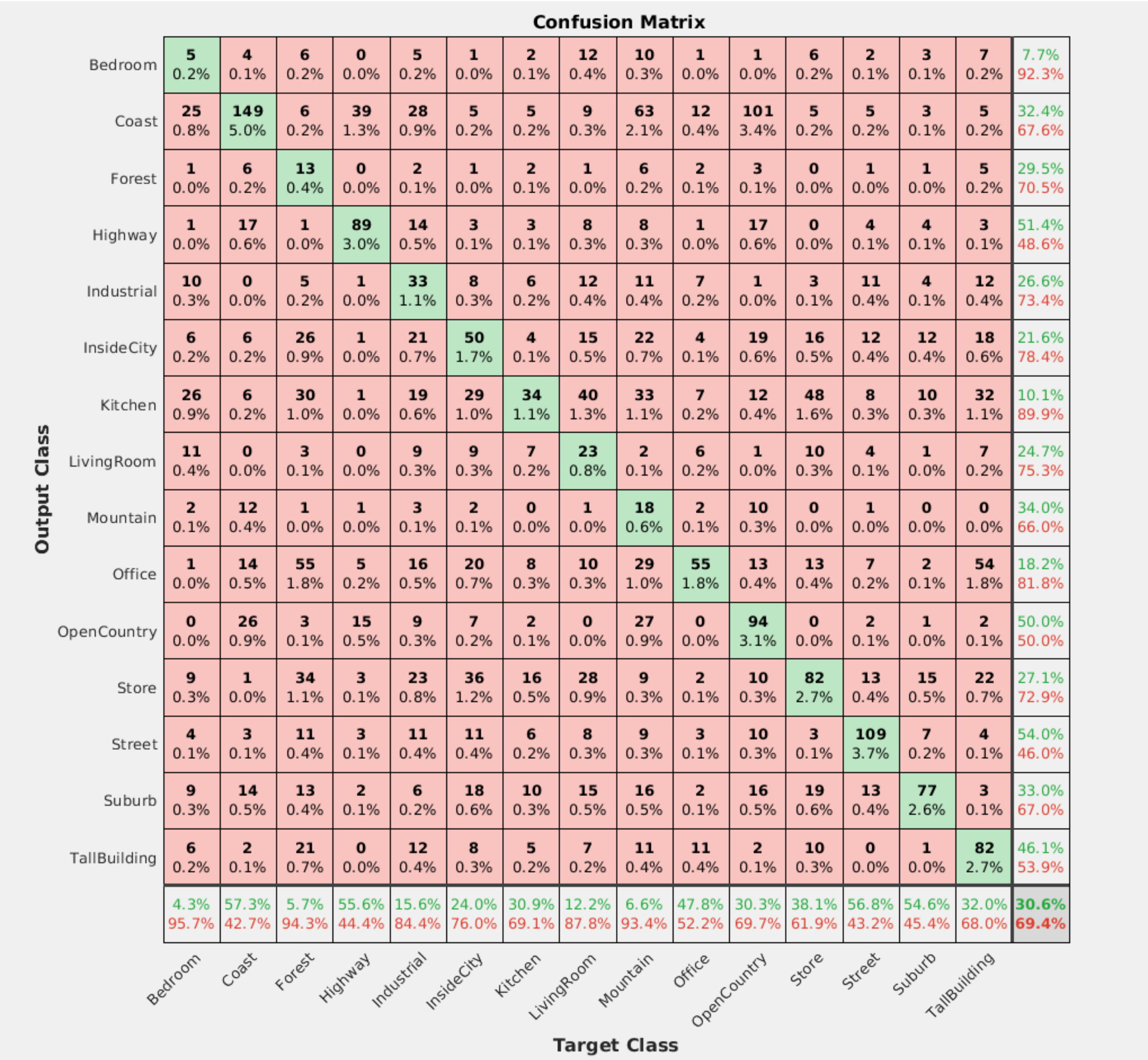

Figure 3: Confusion matrix for the baseline network.

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%.

5 Improving the network

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

1. *data augmentation*;
2. addition of *batch normalization layers*;
3. changing the *size of the convolutional filters*;
4. modifying network layout;

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 $\sim$ 39%. Hence, the added complexity represented a slight improvement from the previous step.

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 position 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).

5.3 Improving convolutional layers

Convolution layers may be improved by increasing the size of the filter as they are placed towards the output. In practice, this means modifying the convolution layers such that their size increases from 3×3 only, to 3×3, 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%.

5.4 Modifying network layout

Editing the layout of the network should improve the overall accuracy.
Since layout of the network changed, *MaxEpoch* should be set accordingly, with the goal of avoiding overfitting while leaving the network room for enough complexity.

5.4.1 Adding fully-connected and dropout layers

A new fully-connected layer, along with a ReLU layer, were added:

```
fullyConnectedLayer(256,'Name','fc_1',...
    'WeightsInitializer', @(sz) randn(sz)*0.01,...
    'BiasInitializer', @(sz) zeros(sz))
reluLayer('Name','relu_4')
```

A dropout layer separating the two fully-connected layers was added too,

```
dropoutLayer(.25, 'Name', 'dropout_1')
```

5.4.2 MaxEpoch

In order to let the network have enough room for complexity, a `MaxEpoch` value of 25 should be set.

5.4.3 Results

Overall network layout is illustrated in the following Table:

Improved Network Layout	Layer type	Size and parameters
1	Image input	$64 \times 64 \times 1$ images
2	Convolution	8 3×3 convolutions, stride 1
3	Batch Normalization	
4	ReLU	
5	Max Pooling	2×2 max pooling, stride 2
6	Convolution	16 5×5 convolutions, stride 1
7	Batch Normalization	
8	ReLU	
9	Max Pooling	2×2 max pooling, stride 2
10	Convolution	32 7×7 convolutions, stride 1
11	Batch Normalization	
12	ReLU	
13	Max Pooling	2×2 max pooling, stride 2
14	Fully Connected	256
15	ReLU	
16	Dropout	25%
17	Fully Connected	15
18	Softmax	softmax
19	Classification	crossentropyex

while training options are listed below,

Improved Network parameter	Value
Optimization Algorithm	sgdm
Epochs (stopping criterion)	25
Initial Learning Rate	0.001
MiniBatch size	32
Weights initialization	Gaussian with $\mu = 0$ and $\sigma = 0.01$
Bias initialization	0

Students reported that these further improvements led to a *validation accuracy* of around 60%, while the *test accuracy* stands at a 57% on average.
Training progress and confusion matrix, along with a comparison with the baseline, will be discussed in Section 5.6.

5.5 Parameters and layout enhancements that did not provide benefit

Students chose the simplest network among many others that did not bring much of an improvement. The choice of the simplest network has been determined by three factors:

1. the network should have a higher overall accuracy when compared to others;
2. if not, the network should not see its complexity increased with no tangible benefit performance-wise;
3. if not, the network should be as much comparable as possible to the baseline (it should share the same parameters as much as possible).

With this two fundamental goals in mind, the students discarded other configurations which did not result in significant improvements performance-wise, or introduced higher complexity, with less parameters shared between the improved network and the baseline.
In particular,

- increasing or lowering the size of *MiniBatch* has not been sufficient to improve performance – however, it changed significantly the behavior of the training phase, which required setting the *initial learning rate* to a different value;
- switching the optimization algorithm to *adam* has not been sufficient to show significant improvements, even after fine-tuning parameters. Students tried many different variants of the network with the adam optimization algorithm, however the results were not good enough to justify the switch;
- adding any convolutional layer did not result in performance improvements.

5.6 Comparison with the baseline

The students collected pictures of training progress and relative confusion matrix of both *baseline network* (parameters in Section 4.9) and *improved network* (parameters in Section 5.4.3).
Figure 5 shows that classes that had been better predicted by the baseline network are still among those better predicted by the improved network too (for instance, *Street* and *TallBuilding*), while some other classes that were initially difficult have seen a huge improvement (*Bedroom* class, *Forest* class, *Kitchen* class).
In general, the improved network shows significant better performance when compared to the baseline network (an almost doubled accuracy on both validation and test set), at the cost of added complexity and some more training time required. Other variants of the improved network were tried by varying parameters and layout as described in Section 5.5, but no significant accuracy improvements justified the applied changes.

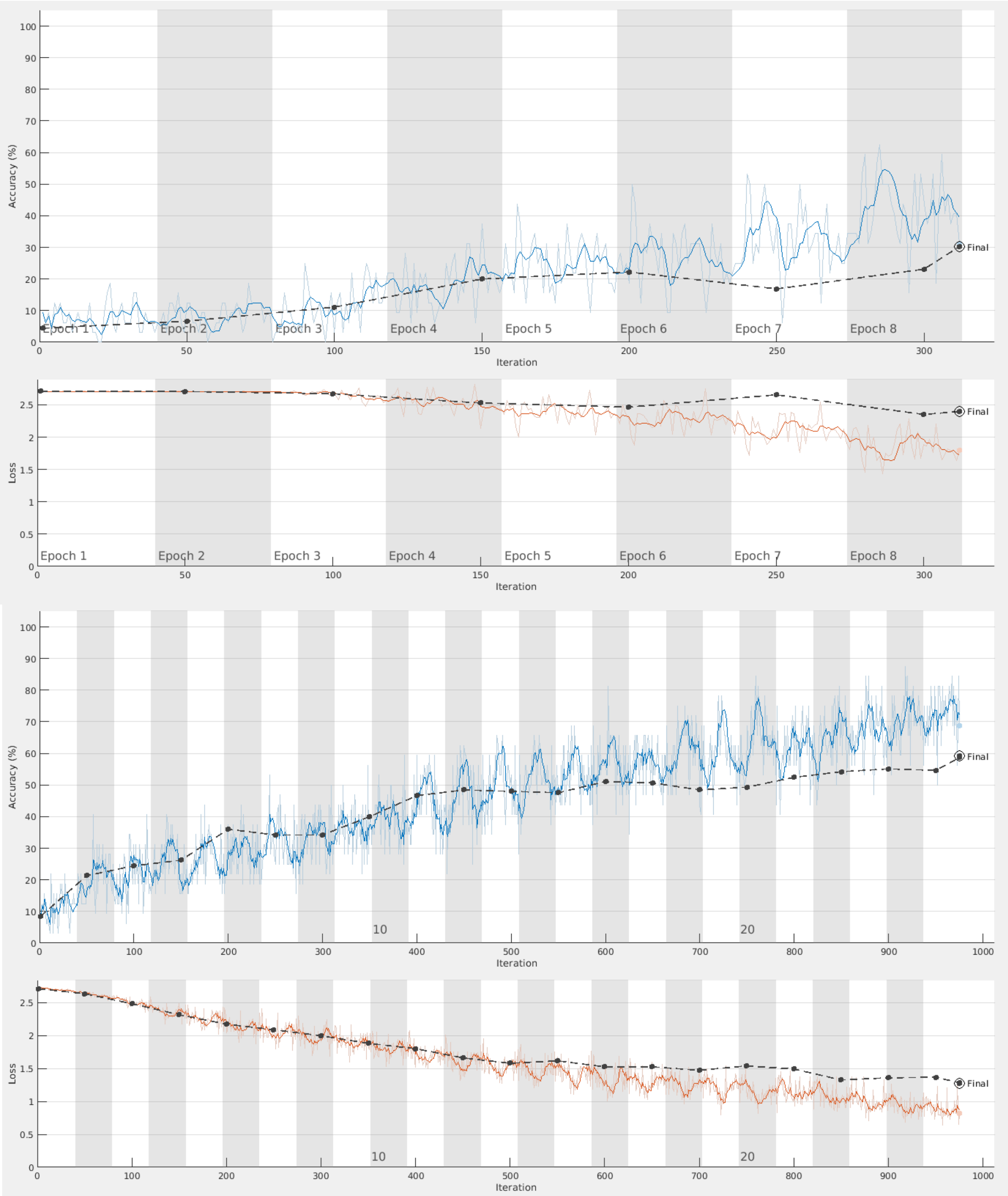

Figure 4: Comparison between training progress of the baseline network (on top) and training progress of the improved network (on bottom).

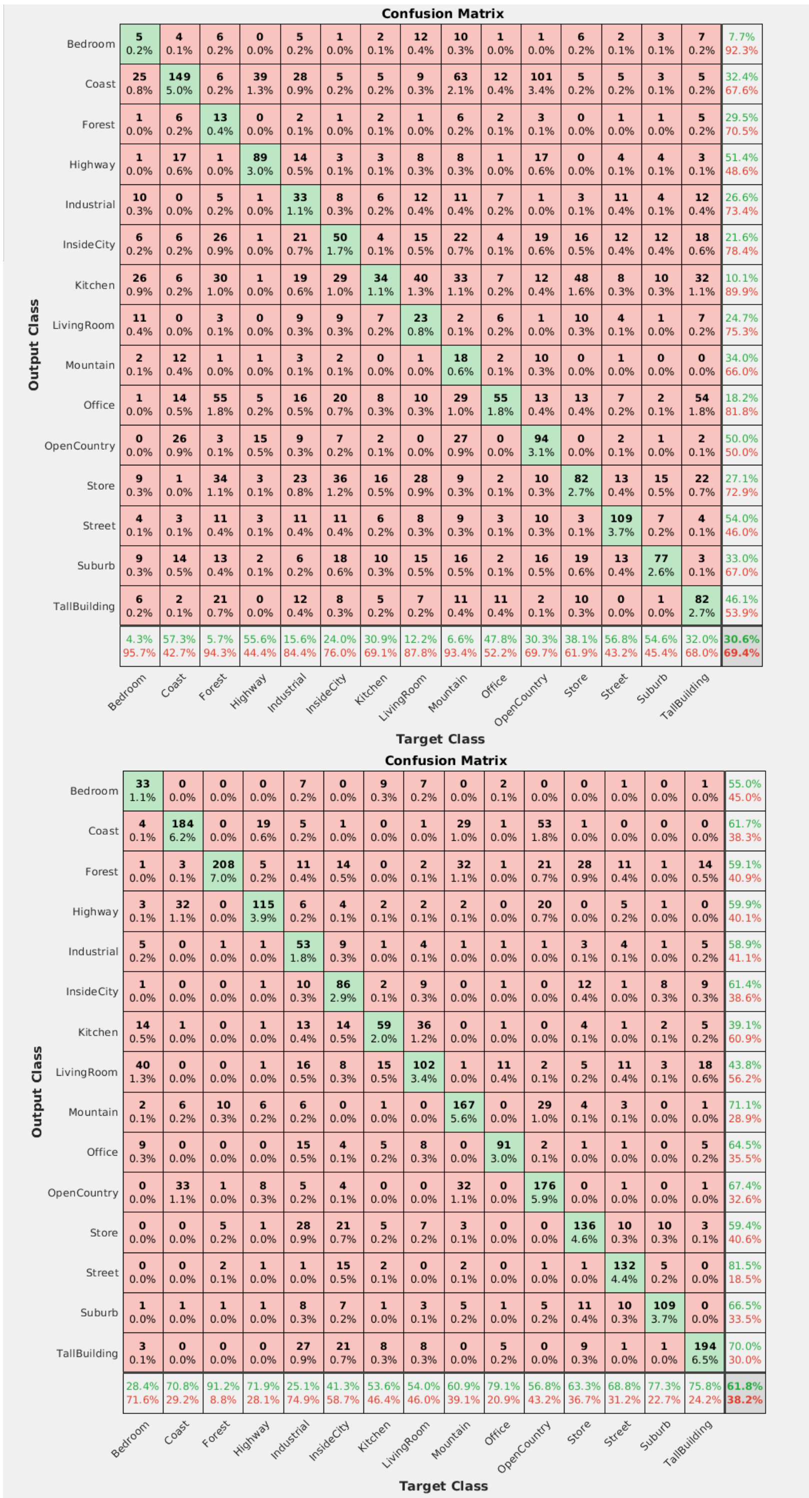

Figure 5: Comparison between confusion matrices of the baseline network (top) and the improved network (bottom).

5.7 Ensamble network based on previous improvements

The students implemented an **ensamble network**, composed of 5 networks of the same layout as in Section 5.4.3. In order to assign the class, a *voting* system has been built: each network of the ensamble casts a vote (a class), and the overall output is the *most common predicted class* among the ensamble.

The code related to the ensamble prediction is the following one,

```
% Evaluating ensamble network accuracy

EnsambleNum=5;
for i=1:EnsambleNum
    YPred = classify(net(i),imdsTest);
    for j=1:size(YPred)
        YPredicted(j,i) = YPred(j);
    end
end

categ = categorical(categories(YTest));
for i=1:size(YPred)
    [v, argmax] = max(countcats(YPredicted(i,:)));
    predicted_output(i) = categ(argmax);
end

accuracy = sum(predicted_output == YTest)/numel(YTest);
figure
plotconfusion(YTest, predicted_output')
```

where `net(i)` is a network belonging to an array of 5 trained networks. As a result, the above code will plot a confusion matrix for the ensamble network.

5.8 Justifying the added complexity

Ensamble training is very heavy on resources. Time training multiple networks is significantly higher than the time required for training a single network, let alone the time necessary for both training and classifying with an ensamble of networks. Hence, the students questioned themselves whether an ensamble of more networks (for instance, 10 networks or more) could bring any improvement with respect to an ensamble of 5 or less.

In order to answer the question, the students collected test accuracy of ensamble networks of different number B of networks, and compared them with the average of the single networks in the ensamble when taken alone. Each ensamble network has been trained separately, and results has been collected in the following table.

B	Single networks average accuracy	Ensamble test set accuracy	Net improvement
3	0.5763	0.5950	1.87%
5	0.5863	0.6280	4.17%
7	0.5868	0.6340	4.72%
10	0.5841	0.6410	5.69%
15	0.5871	0.6470	5.99%

Basically, training an ensamble of $B = 5$ networks should be enough to provide tangible benefits, with even higher improvements over the single networks when training a $B = 10$ ensamble network (although the improvements are not so pronounced).

Training the network with $B = 15$ only introduced a moderate improvement, thus such a high number of networks is not strictly required and would instead offer more disadvantages than advantages (for instance, training and classification time) with respect to an ensamble network having only 5 or 10 networks.

At every run, ensamble networks gained more accuracy on the test set with respect to a single network. In addition, the single network performance is very unreliable, largely depending on the randomness of the learning phase, with test set accuracy values ranging from 55% to over 60% – on contrary, the ensamble network accuracy has proven to be much more reliable over different runs. On this particular task, ensamble learning has proven to be effective – however, its increased complexity and time required for training and classification cannot be undervalued. A good trade-off between net improvement and increased complexity could be a choice of $B = 5$ networks for the ensamble network, which should bring the accuracy of the student's model well-above 60%.

6 Adopting transfer learning with AlexNet