Computer Vision - Project n.3

Eugenio Ceschia - IN2000099

Problem Statement

It is required, given a dataset that contains images from 15 different categories, to implement an image classifier using Convolutional Neural Networks.
Doing so, the assignment requires to follow an 'incremental' approach, starting from a shallow network with default parameters, and adding more and more improvements with more complex techniques, discussing their effectiveness on the result.
After finishing with these tasks, it is also required to implement Transfer Learning on a pre-trained network and comment on the results.

Description of the Approach

I followed an incremental approach, as the assignment suggested.
First I trained a shallow network using almost all the default parameters, then after analyzing the results, I proceeded to implement some better strategies, such as batch normalization, data augmentation and dropout.
As the final task, AlexNet has been used as a pre-trained network in a transfer learning approach, manifesting other improvements that will be shown in the Results section.
In the process, standard Matlab libraries have been used, except for the DAG implementation when dealing with multiclass SVMs, implemented following [Platt, 2000] paper.

Implementation Choices

Task 1

I started the training with default parameters other than the ones specified by the assignment. In this case the *learning rate* was set by default to 0.01.
Following an empirical procedure I started decreasing exponentially the learning rate to see the possible improvements obtained, resulting in a final value of 0.0005. For this first task no automatic early stopping had been used, only manual stopping had been employed when the training was clearly plateauing.

Task 2

Data augmentation

I reflected images left-to-right using the *imageDataAugmenter* Matlab function, which randomly reflects images with a probability of 50% in every Batch.

Batch normalization

For Batch Normalization the standard Matlab Batch Normalization layer has been used.

Modifying layers and parameters

I first enlarged successive convolutional filters (3x3 - 5x5- 7x7) as suggested and to further improve results I decided to add two more convolutional layers with 9x9 and 11x11 filter sizes.
The learning rate had been incremented after the employing of Batch Normalization layers as suggested in [Ioffe and Szegedy, 2015].
Dropout layer had been used just once, at first with a probability of 0.5 (default parameter). Decreasing it to 0.4 showed the best improvement in results.
For mini-batches size I started with a value of 32, successively increasing it. I found out that a size of 128 led to the best performances.

Ensemble of networks

I tried to implement three ensembles, of 5, 7 and 10 networks respectively.
The networks' layout remained the same through all the process, using the one from the preceding task, which proved to be the best to that point. I set *ValidationPatience* to a value of 2 in order to avoid single increments in Loss that could lead to a worse overall performance.
I also changed training and validation datasets before every network training, by randomly splitting the dataset given with the requirements. The output class had been chosen by averaging the scores of every predictor and taking the maximum value for every observation.

Cropping and rotating

I randomly cropped train dataset images to a size of 48x48 and added them to the original dataset.
This resulted in a bigger dataset with respect to the other situations.
In fact, the imageDataAugmenter performs “online augmentation” of the dataset, not providing cropping functionalities. Hence, I had to perform “offline augmentation” just for the cropping, and then rotate and mirror images during training.

Task 3

Fine Tuning

AlexNet has been used as a pre-trained network to implement transfer learning with.
Since its input layer required 227x227x3 images, a rescaling was necessary. To obtain a 3-channel image from a grey-scale one I simply copied the image three times to simulate an RGB image.
Setting a high learning rate for the last FC layer allowed to train it quicker and to use just 6 epochs.
In order to freeze the weights of the other layers an initial Learning Rate of 0.0001 has been chosen.

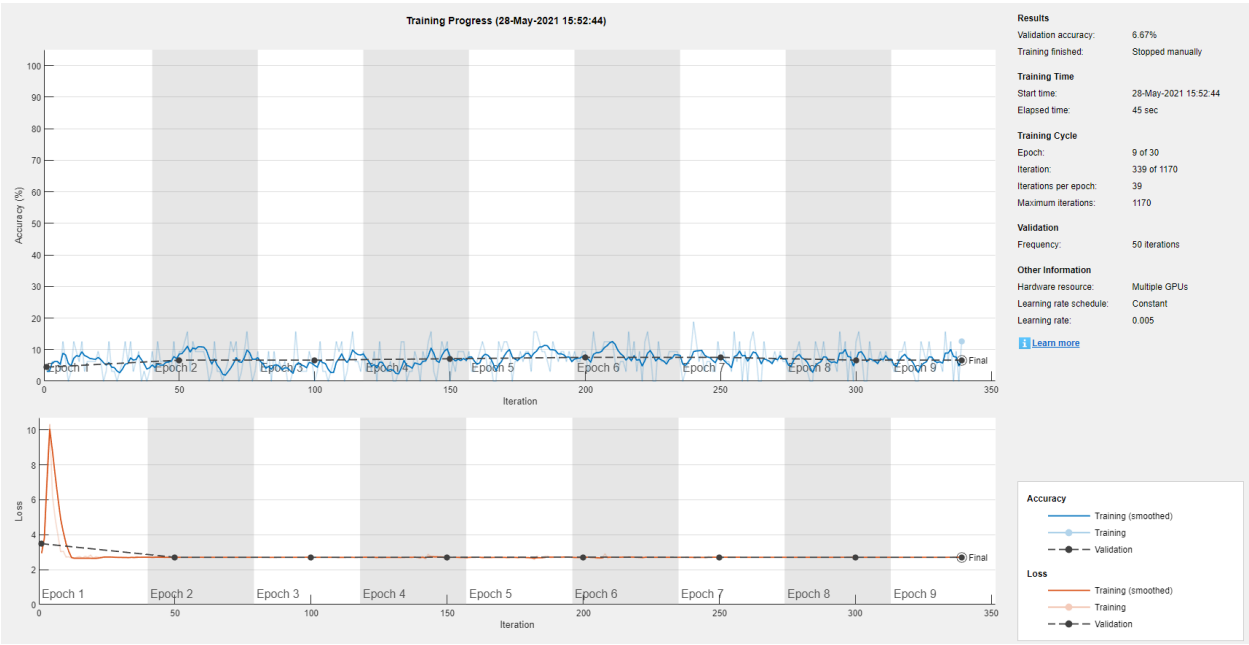
Multiclass SVM

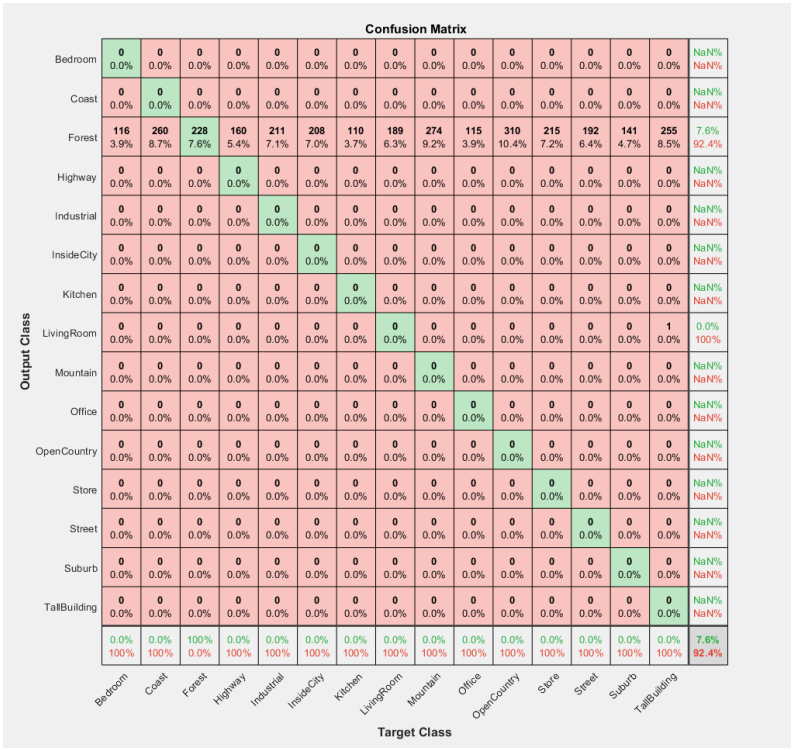
To implement DAG approach for multiclass classification with SVM I used a recursive function that, probably, is poorly optimized for the task, resulting in a predicting time of around 2 minutes with 15 classes and around 3000 observations.
I did not implement Error Correcting Output Code, instead I used Matlab function *fitcecoc* with its default parameters.

Results

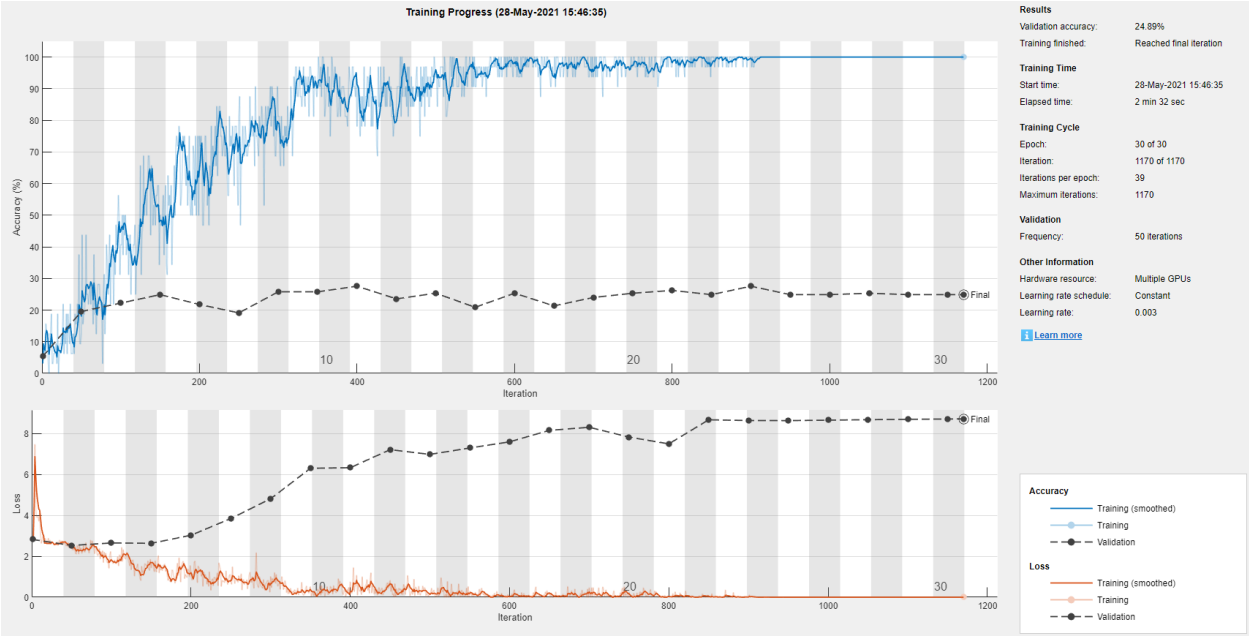
Task 1

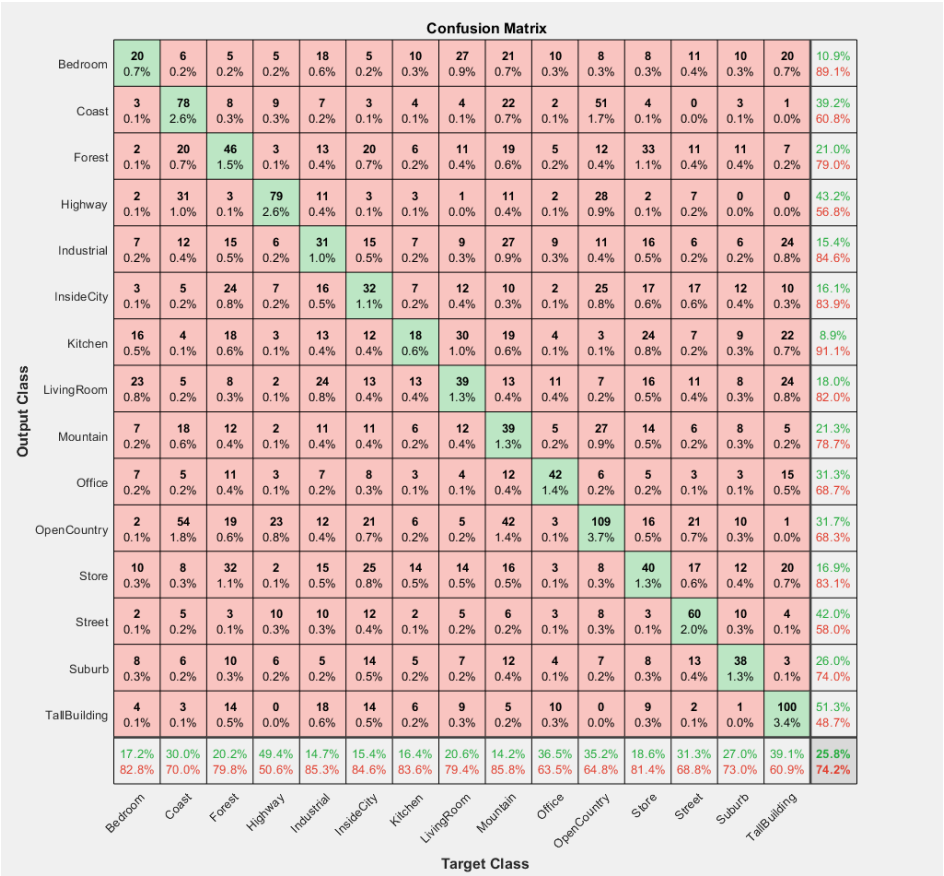
Learning rates from 0.01 to 0.005 were not useful in network training, I report here just the plot for 0.005, which led to a test accuracy of 7.6%.
Manual stopping had been used as it was clear no improvement could happen in this instance.



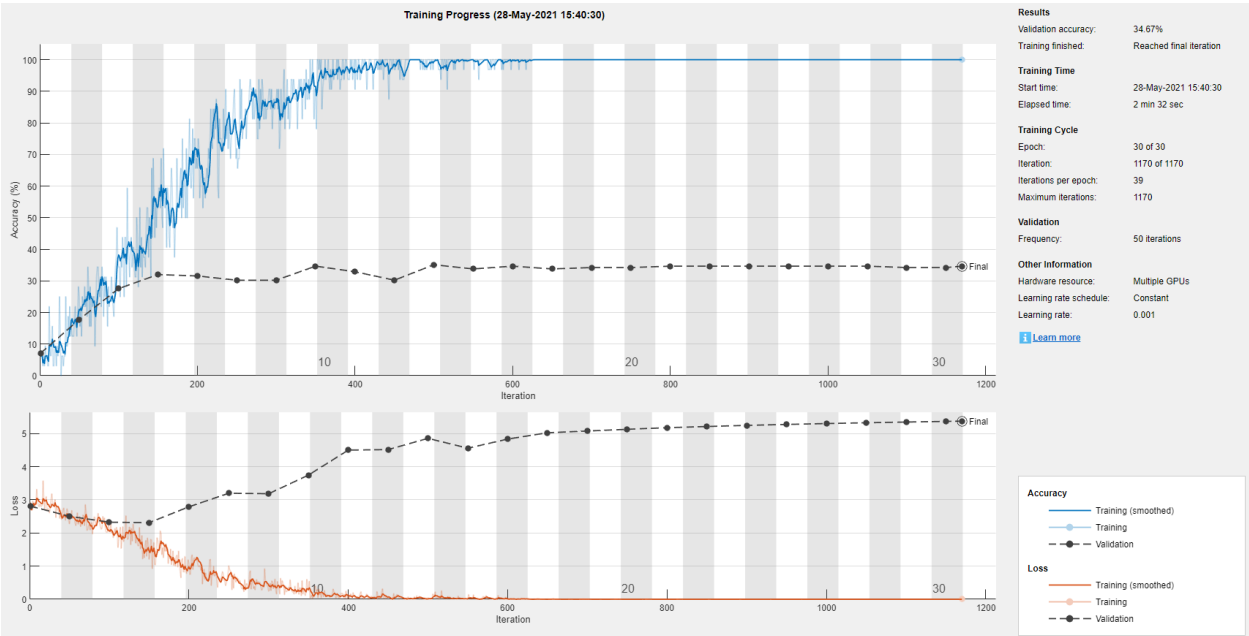


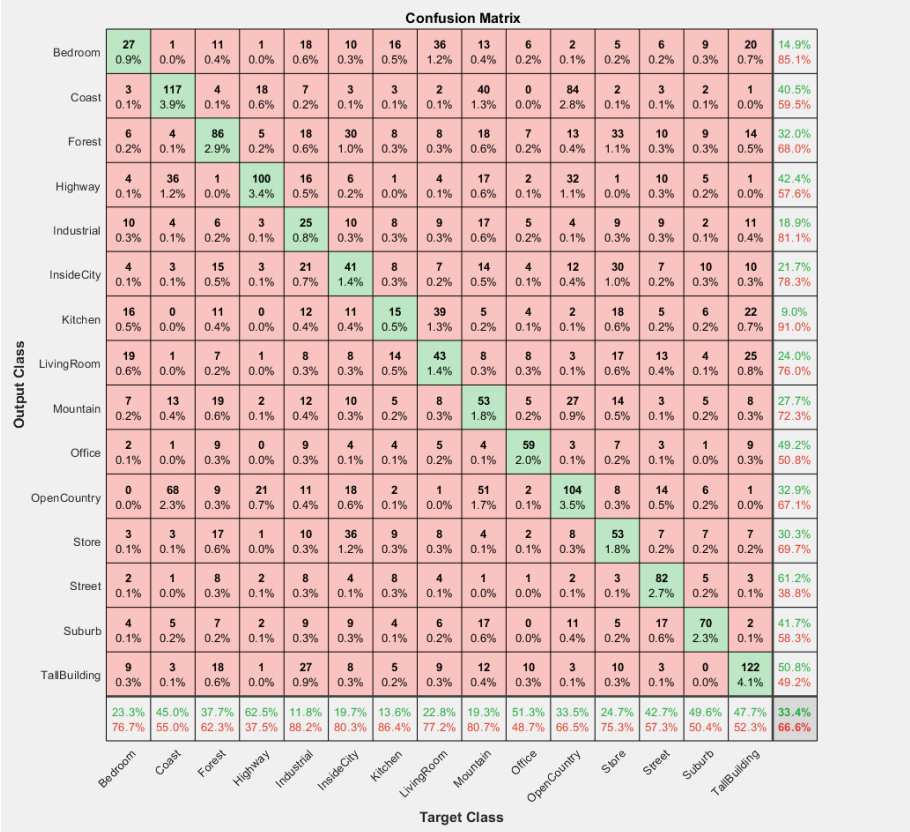
The first learning rate which led to an acceptable result was 0.003.
Here, even if the loss on validation was increasing I still decided not to stop the training and try to use smaller values for LR in the next tries.
The resulting test accuracy was 25.8%.



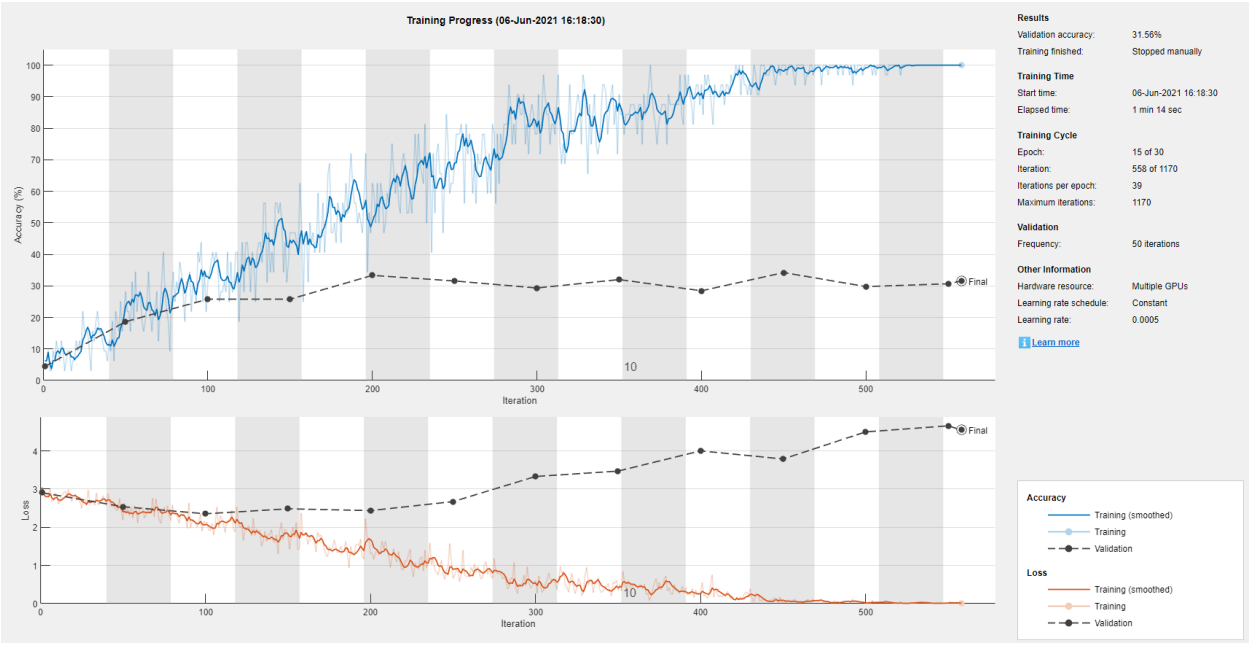


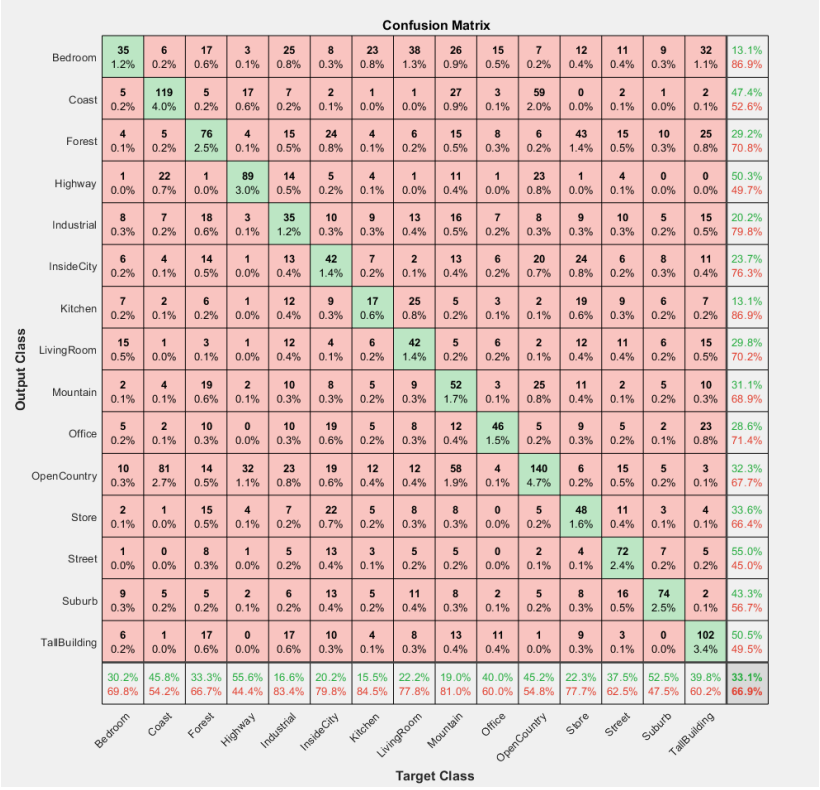
The first learning rate that led to an accuracy of around 30% was 0.001.
From the plot it is clear how 30 epochs were far more than necessary, however, as above I decided not to stop the training and decrease the rate once more.
Here test accuracy reached 33.4%.



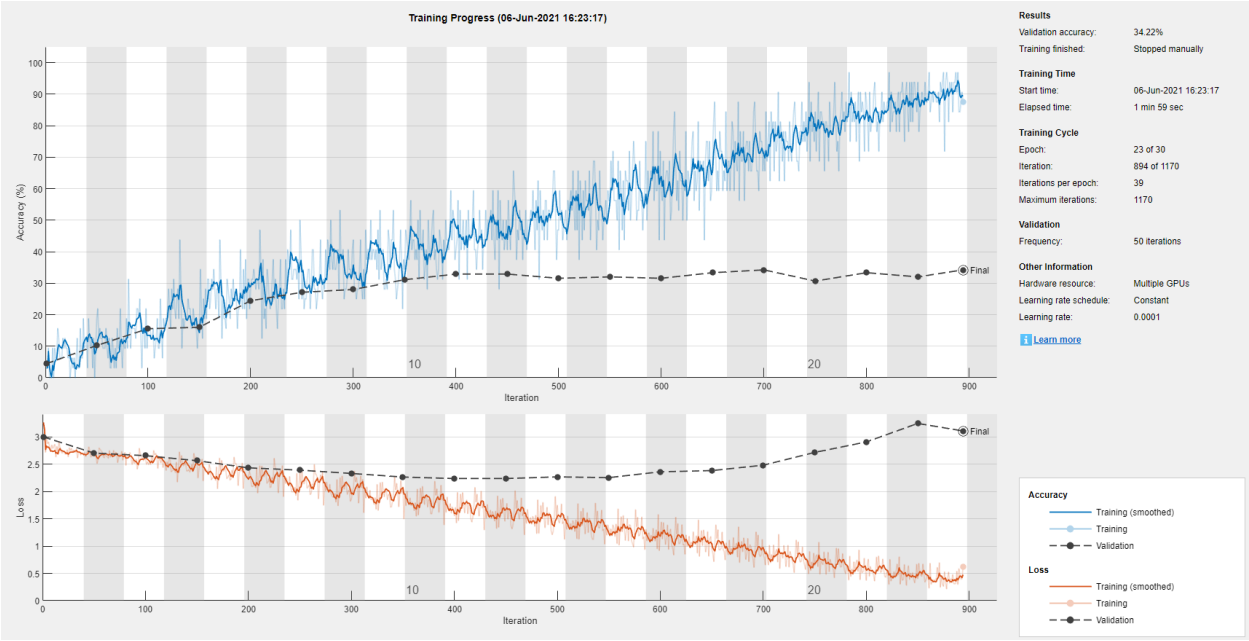


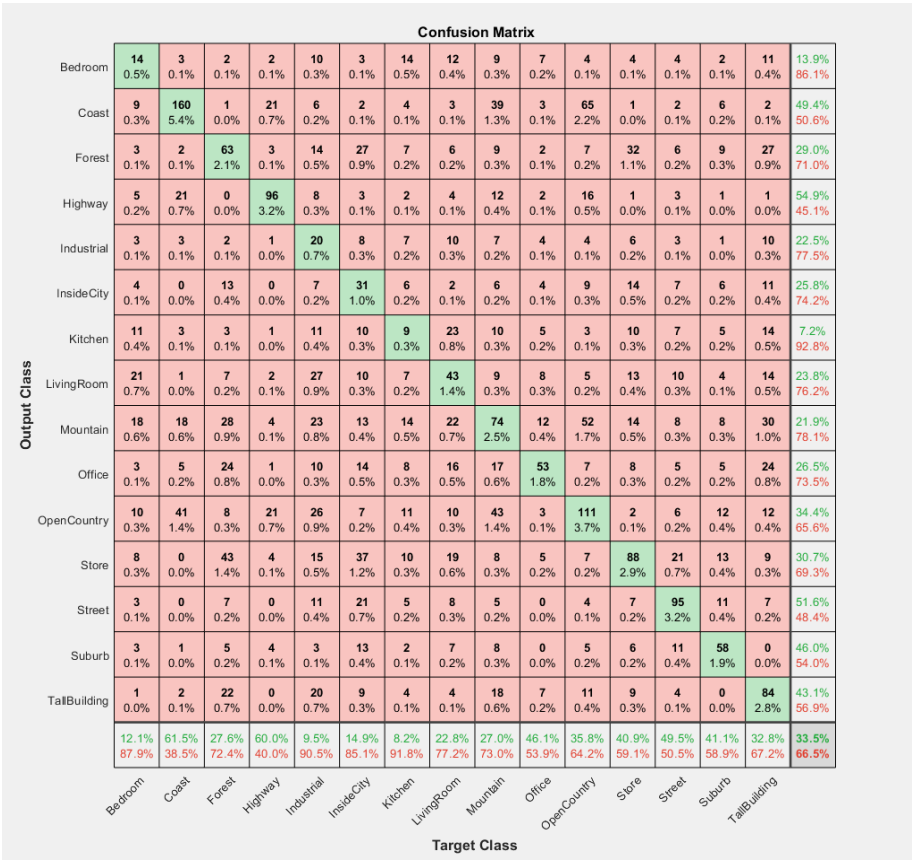
Again, also with a learning rate of 0.0005 30 epochs were more than necessary, this time I stopped the train manually as the loss was increasing quickly.
Here test accuracy reached 33.1%.





In the end, a learning rate of 0.0001 did not improve the result, leading to a test accuracy of 33.5%. Here the training required circa 10 epochs more to reach previous results, and again was stopped when the loss started increasing.

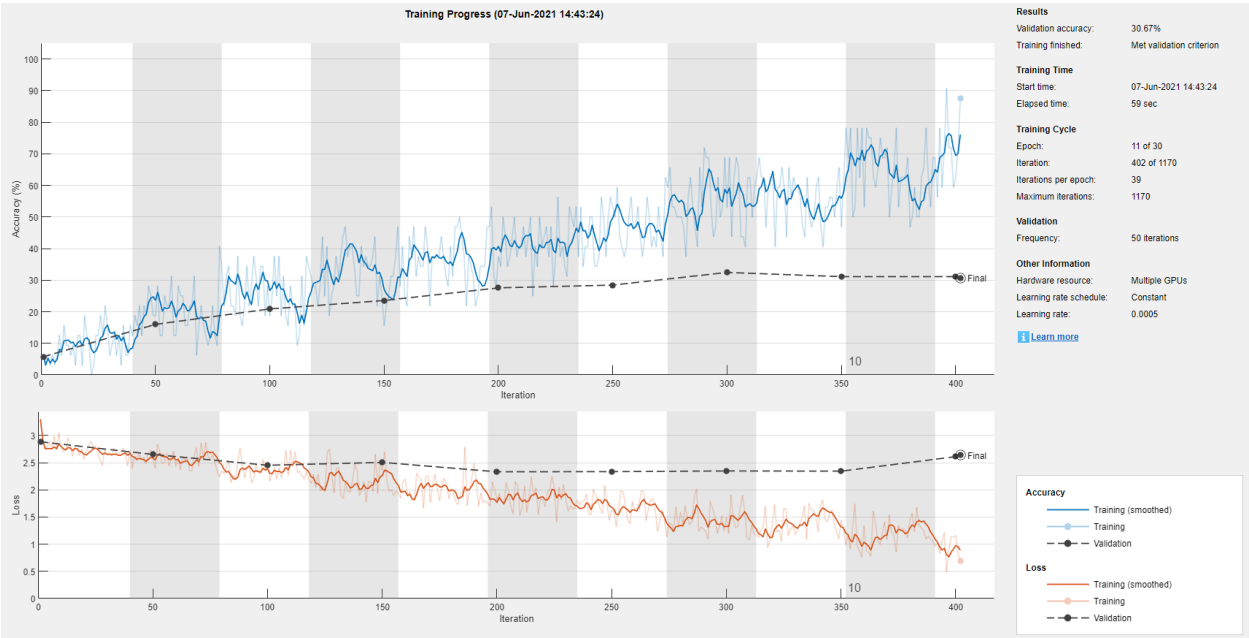


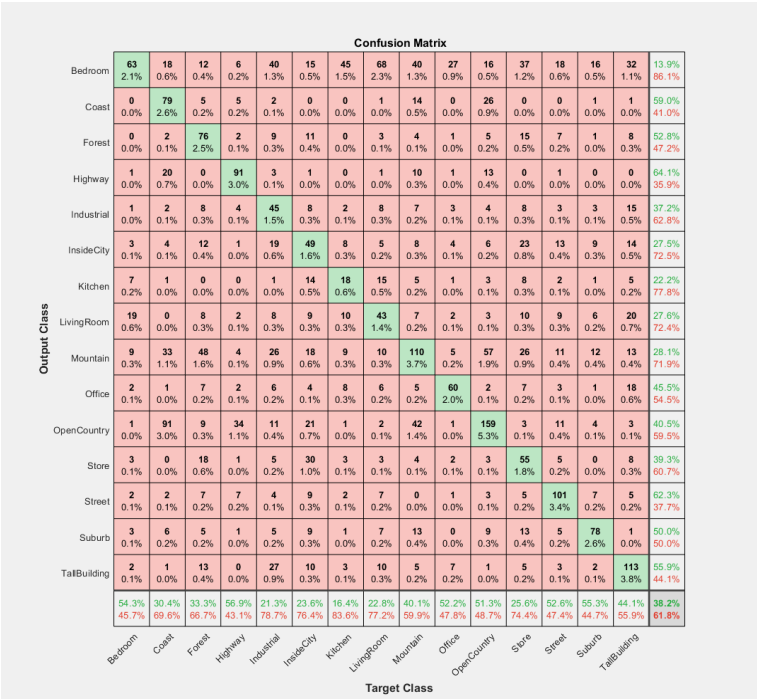


Task 2

a. Augmenting Data

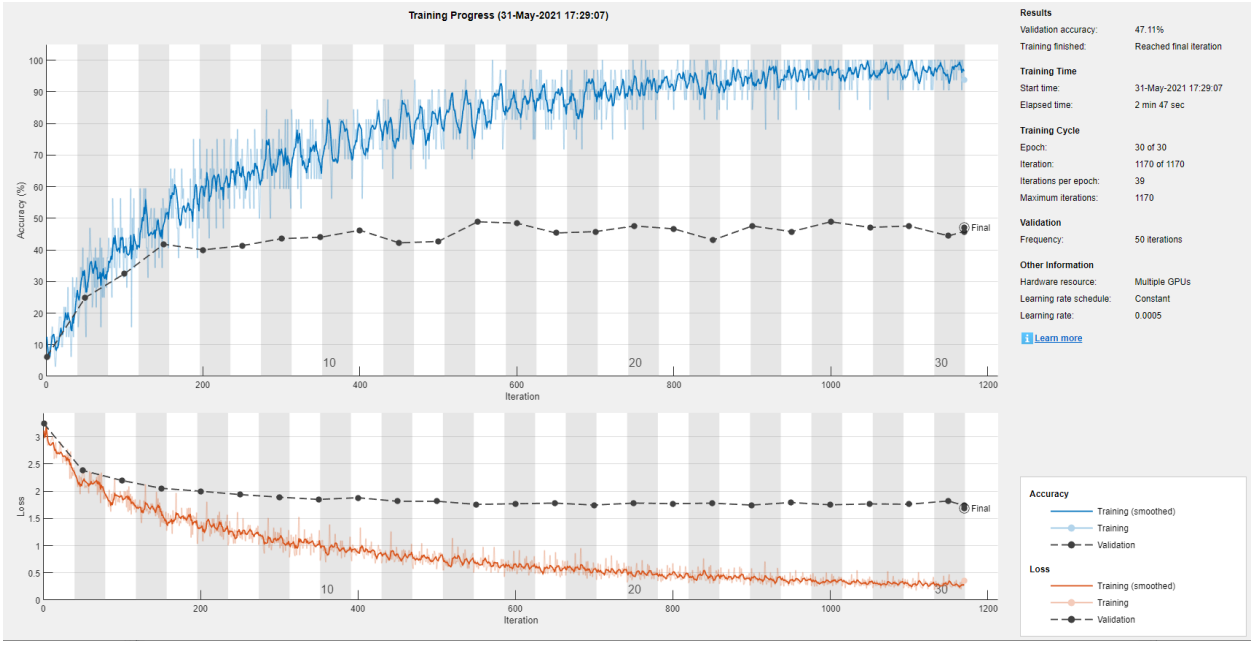
From the previous results, a value of 0.0005 seemed a good compromise for LR.
With the same parameters used in the last task and augmenting the dataset with left-right reflection with probability 50% on the entire dataset , the test accuracy grew to around 40%.

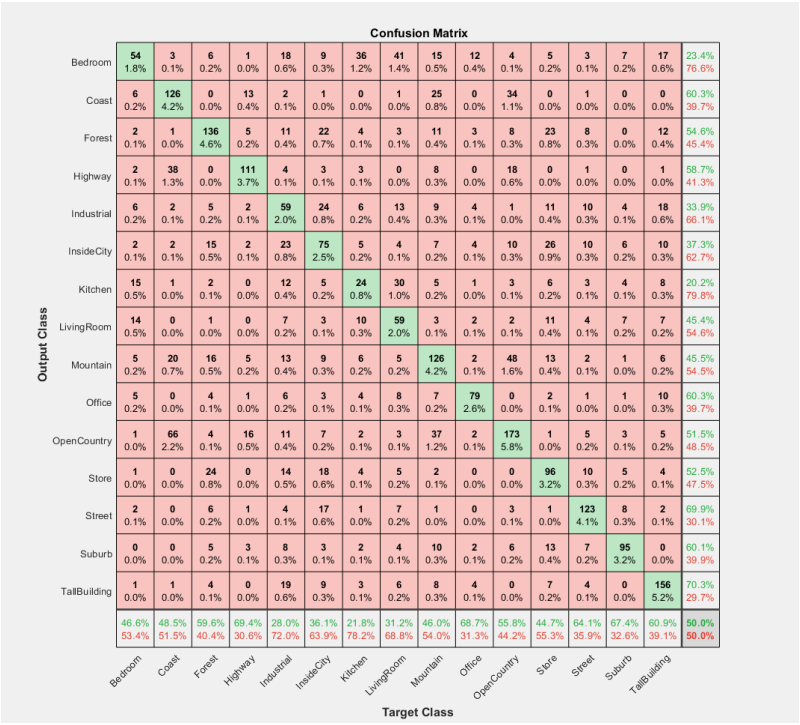




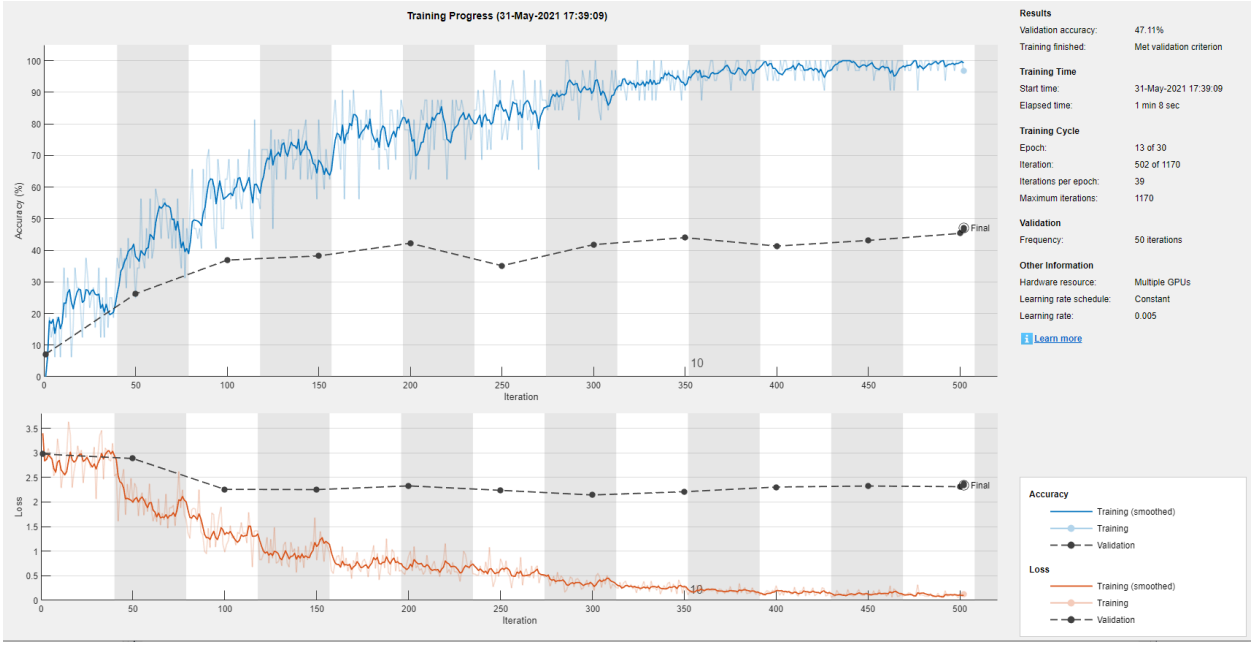
b. Adding Batch Normalization Layer

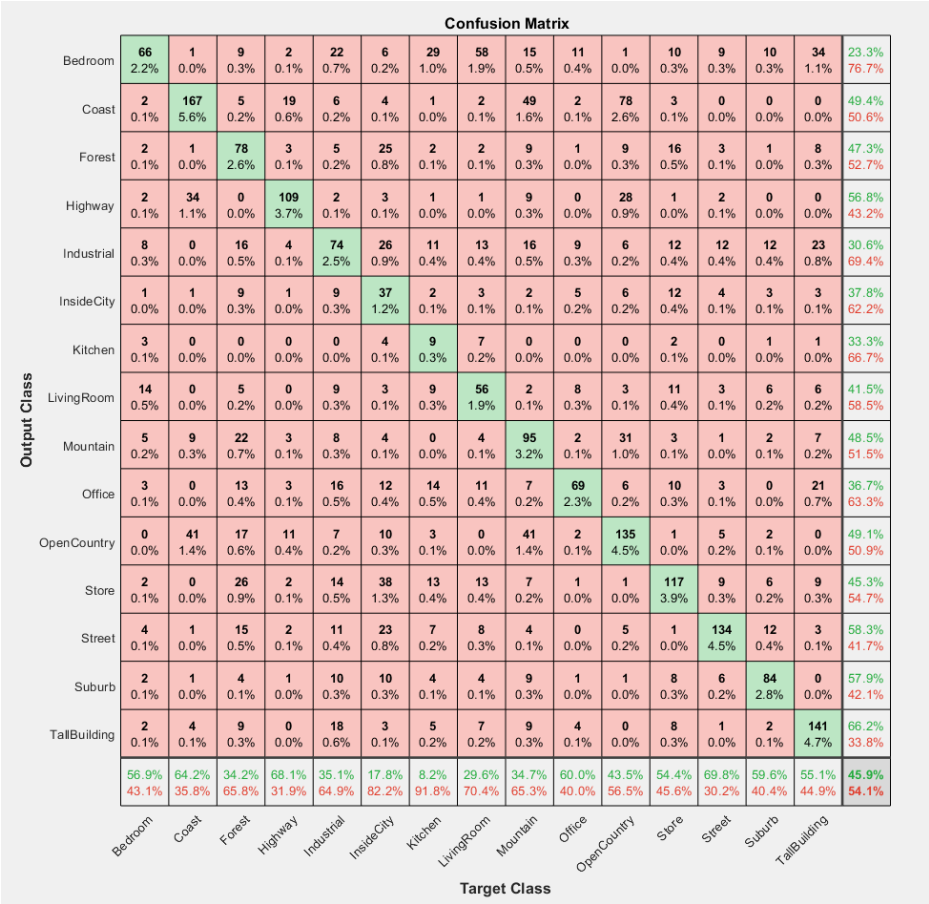
Adding Batch Normalization Layers resulted in an increase of circa 10% on test-accuracy from the previous results given the same Learning Rate (0.0005). However, as specified in [Ioffe and Szegedy, 2015], adding Batch Normalization layers allows to increase the Learning Rate without seeing any decrease in results.



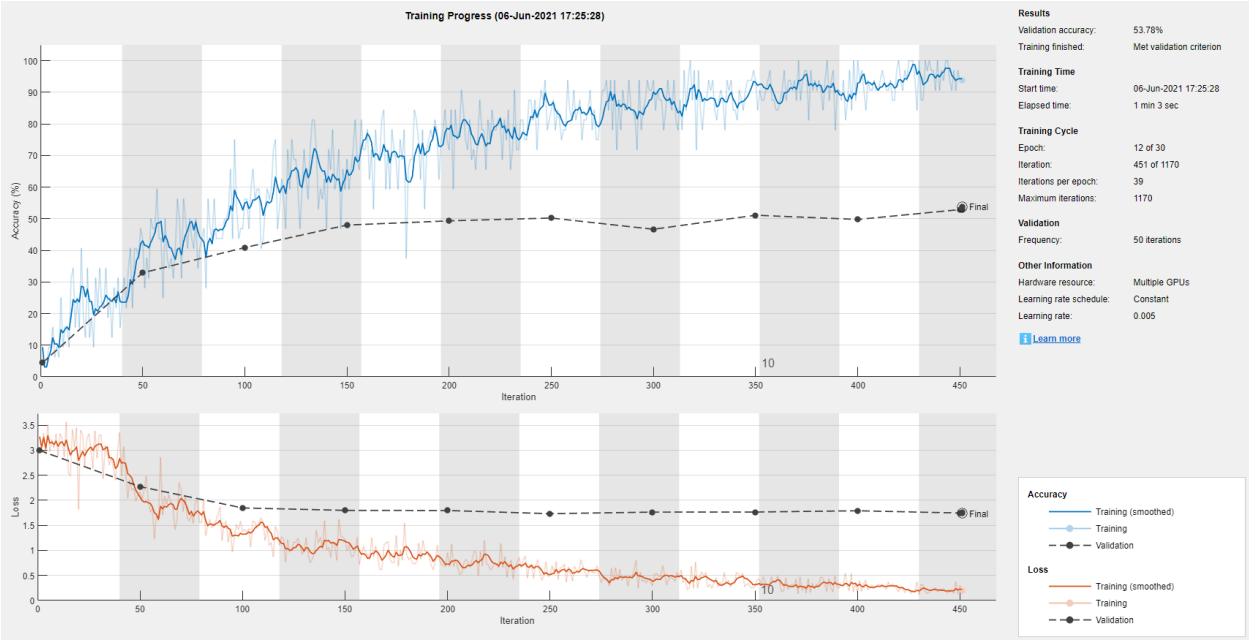


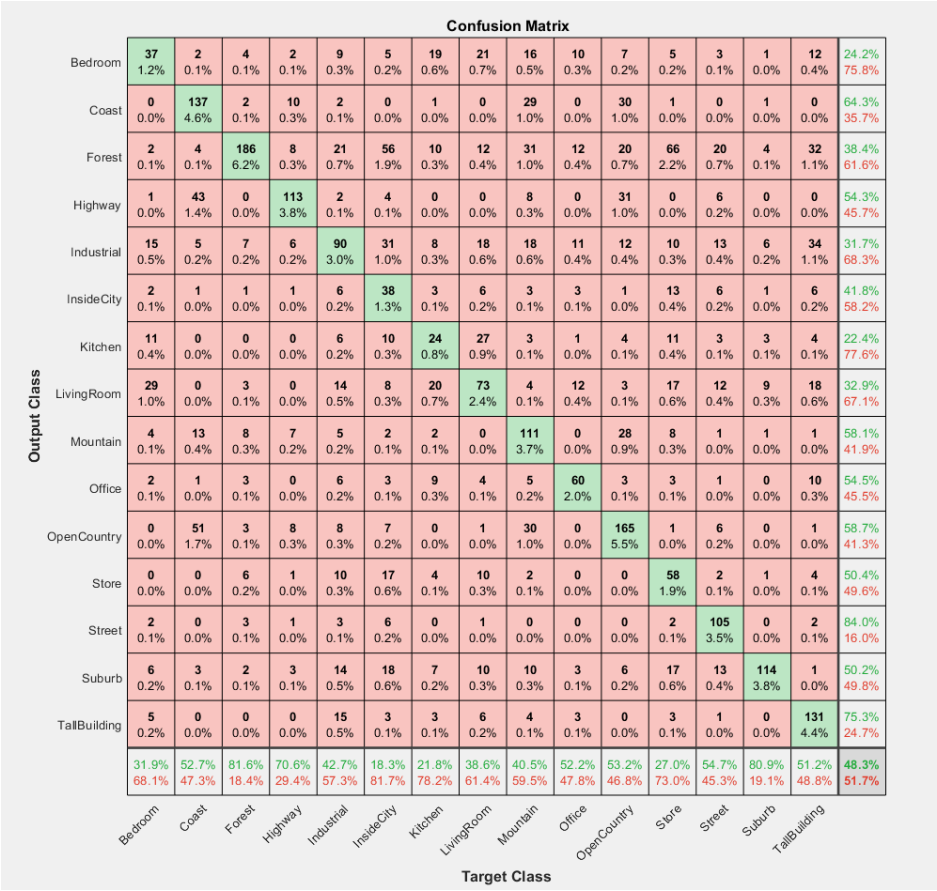
Increasing Learning Rate while maintaining the same Batch Normalization Layers resulted in almost the same performance on accuracy with around half of the training time. (Also, early stopping with Validation Loss was applied, setting *ValidationPatience* parameter to 4).





As stated in [Ioffe and Szegedy, 2015], batch normalization should decrease the need for dropout layers. In fact, employing a dropout layer before the last Fully Connected Layer did not improve accuracy, leading to a value of 48.3%.
Also employing Adam Optimizer did not provide any improvement.

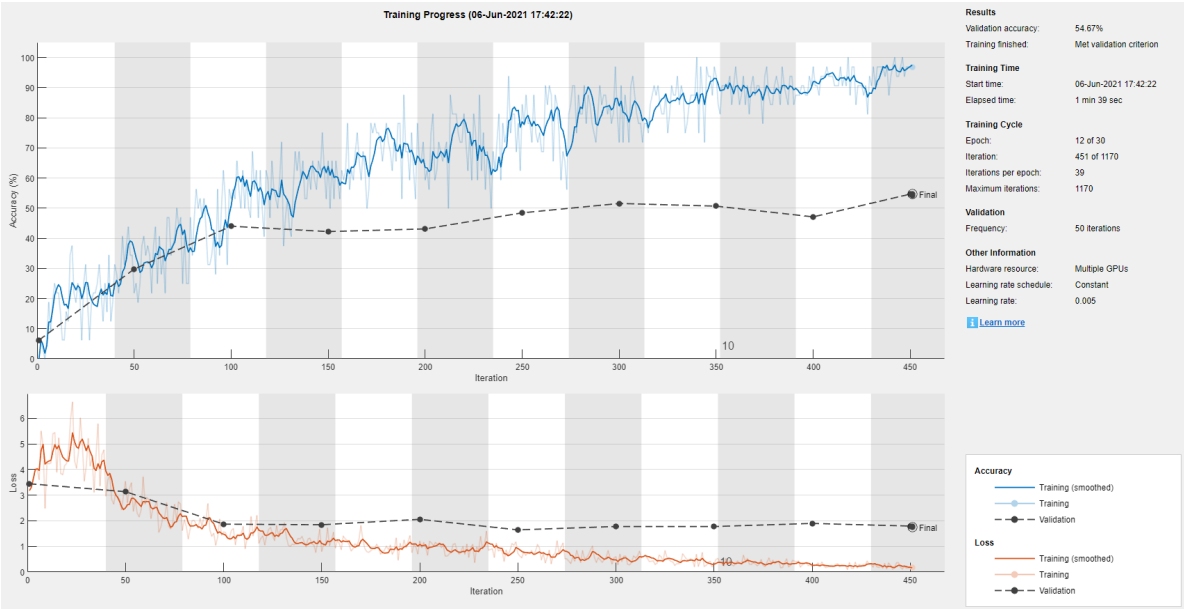


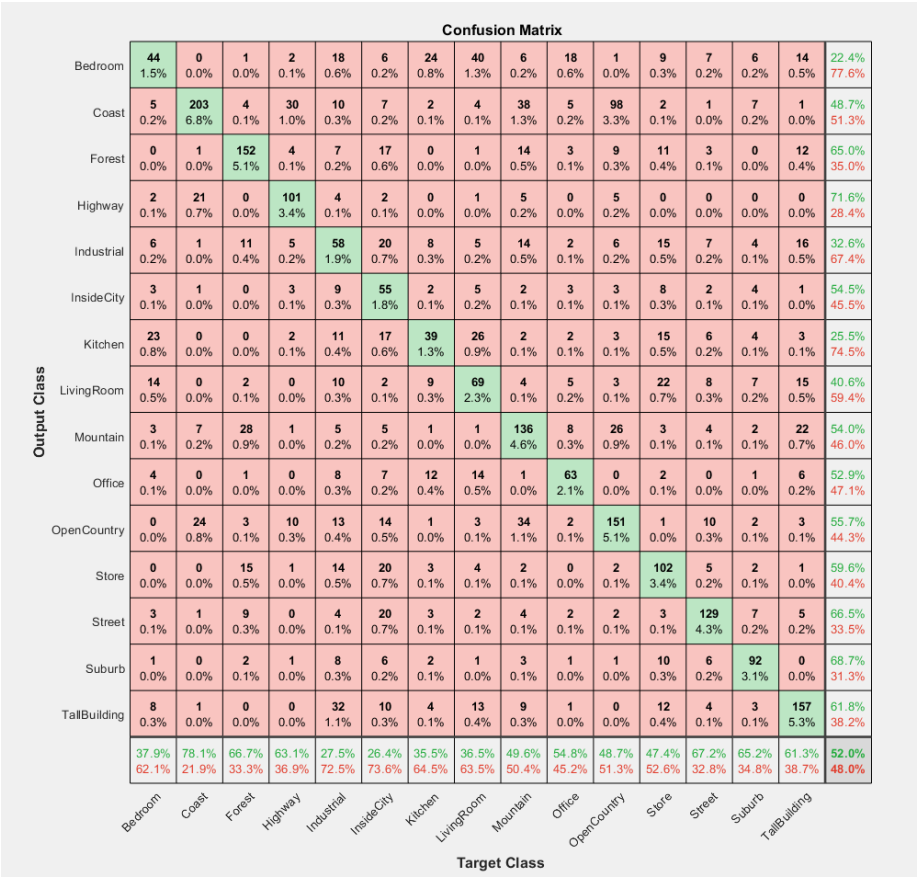


d. Incrementing size of convolutional layers

Incrementing the support of convolutional layers moving from input to output (3x3, 5x5, 7x7), resulted in a further improvement on the test accuracy, reaching 52%.

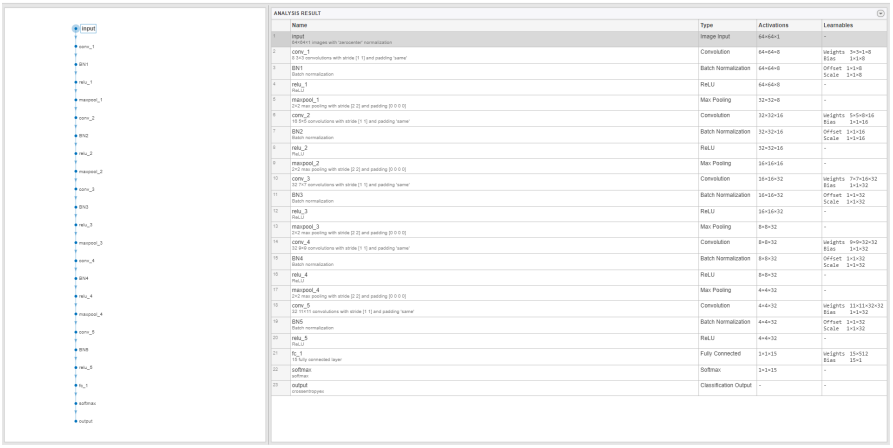
Here training had been stopped imposing the *ValidationPatience* to '4', but lower values could probably lead to the same results.

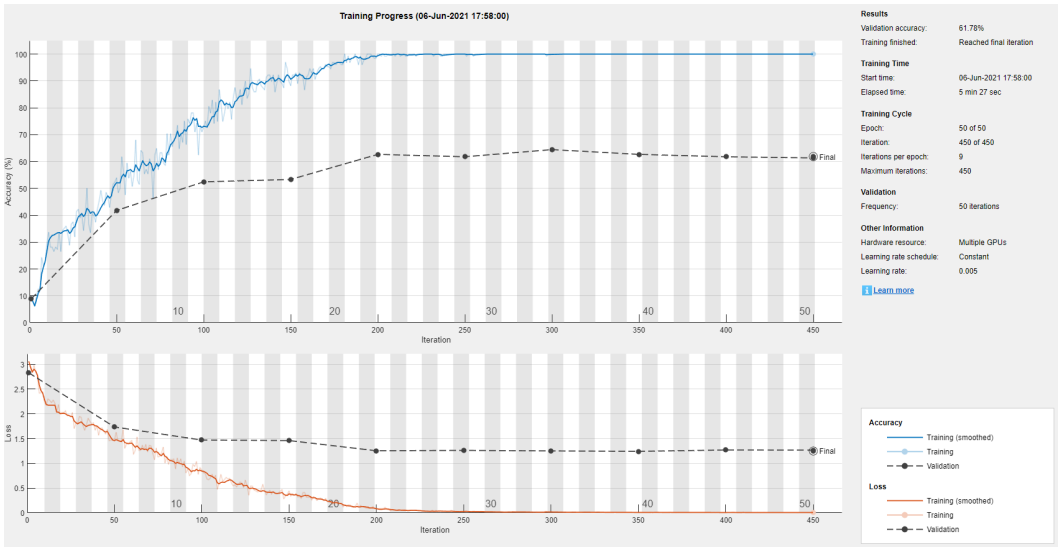


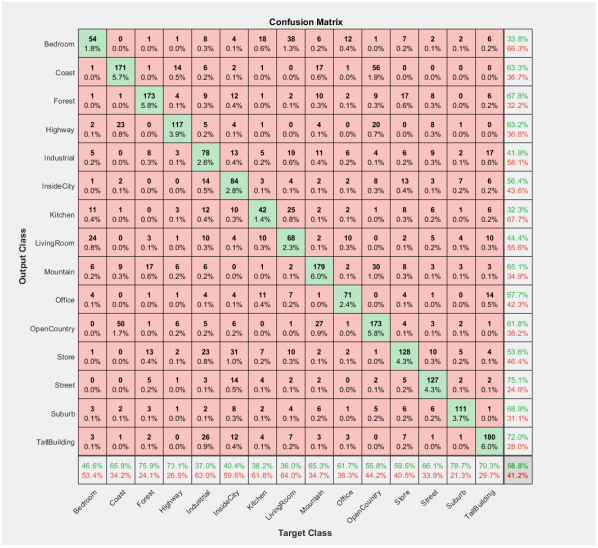


e. Increasing Number of Layers[Optional] and Minibatches

Increasing the number of layers in the network and mini-batches size to 128 led to an accuracy of 57.4%.

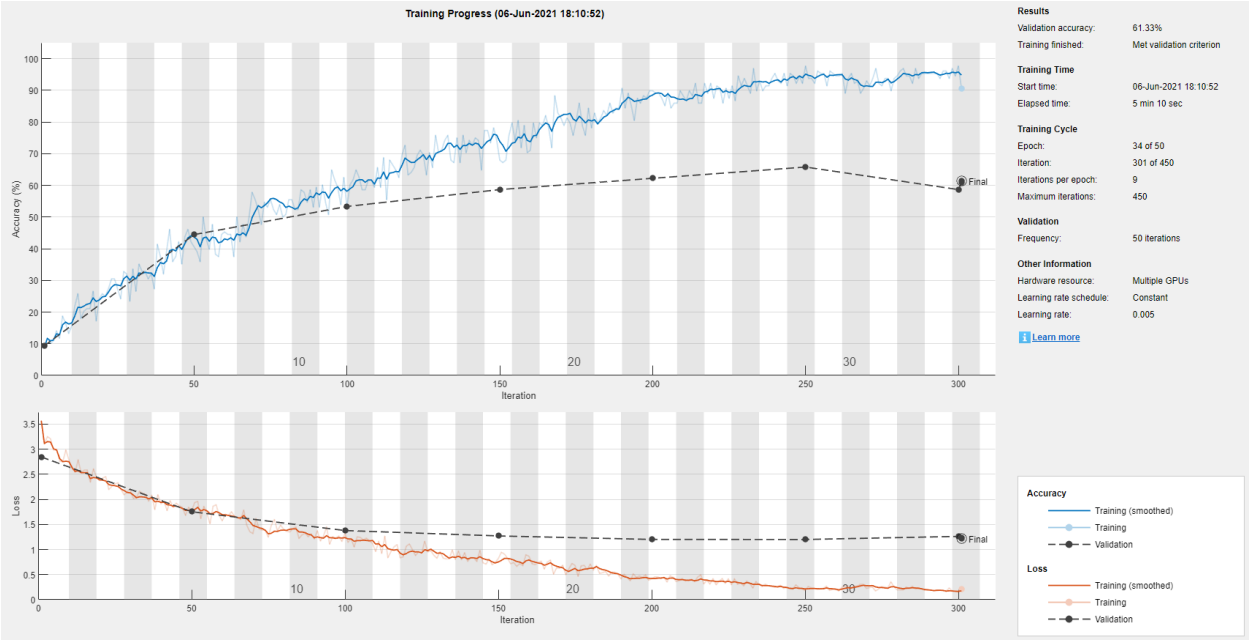


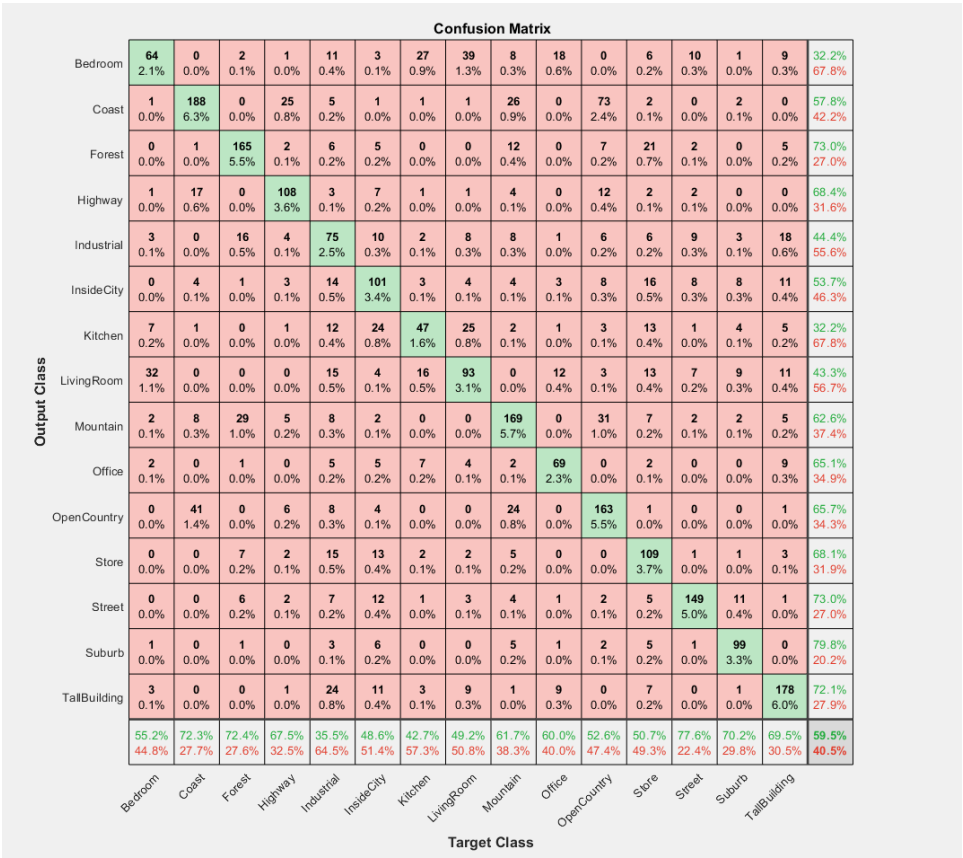




f. Adding Dropout

Here adding a dropout layer increased accuracy of about 2%.
Training has been stopped imposing the *ValidationPatience* to '1', given the lower number of validations.
The resulting accuracy was





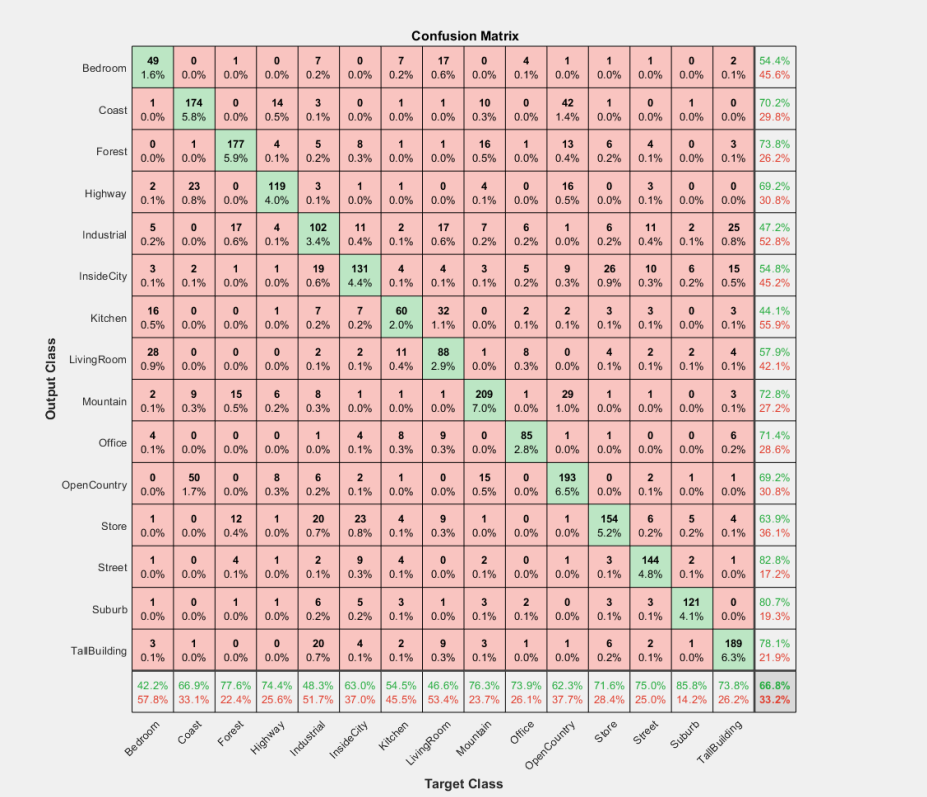
g. Ensemble Of Networks

Applying Ensemble of networks with different training sets did show a general improvement of around 7%.
I display below the single accuracies for every network and the resulting confusion matrices after taking arithmetic mean from the networks' scores

5 Networks

Training 5 Networks resulted in a mean accuracy value of 66.83%.

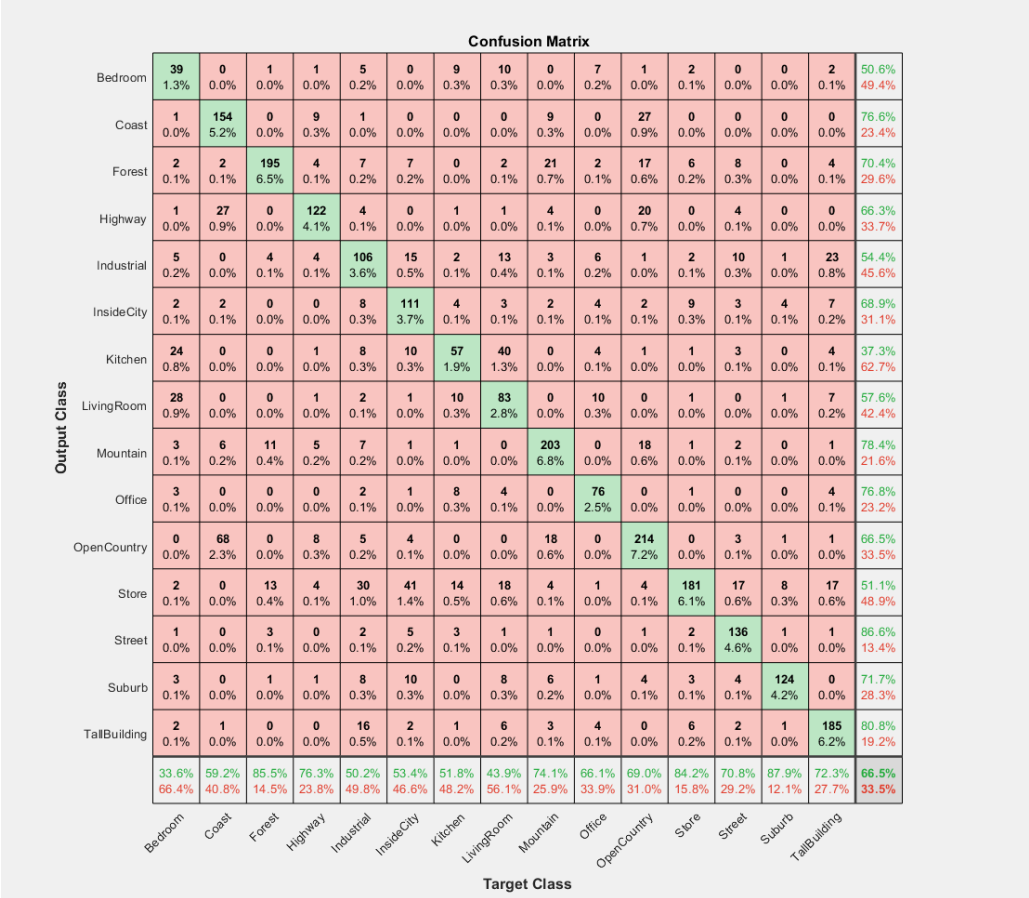
Network	Accuracy
Net1	61.67%
Net2	59.83%
Net3	57.38%
Net4	59.69%
Net5	61.54%



7 Networks

Training 7 Networks resulted in a mean accuracy value of 66.53%.

Network	Accuracy
Net1	62.11%
Net2	62.41%
Net3	60.20%
Net4	59.59%
Net5	61.00%
Net6	59.02%
Net7	57.92%



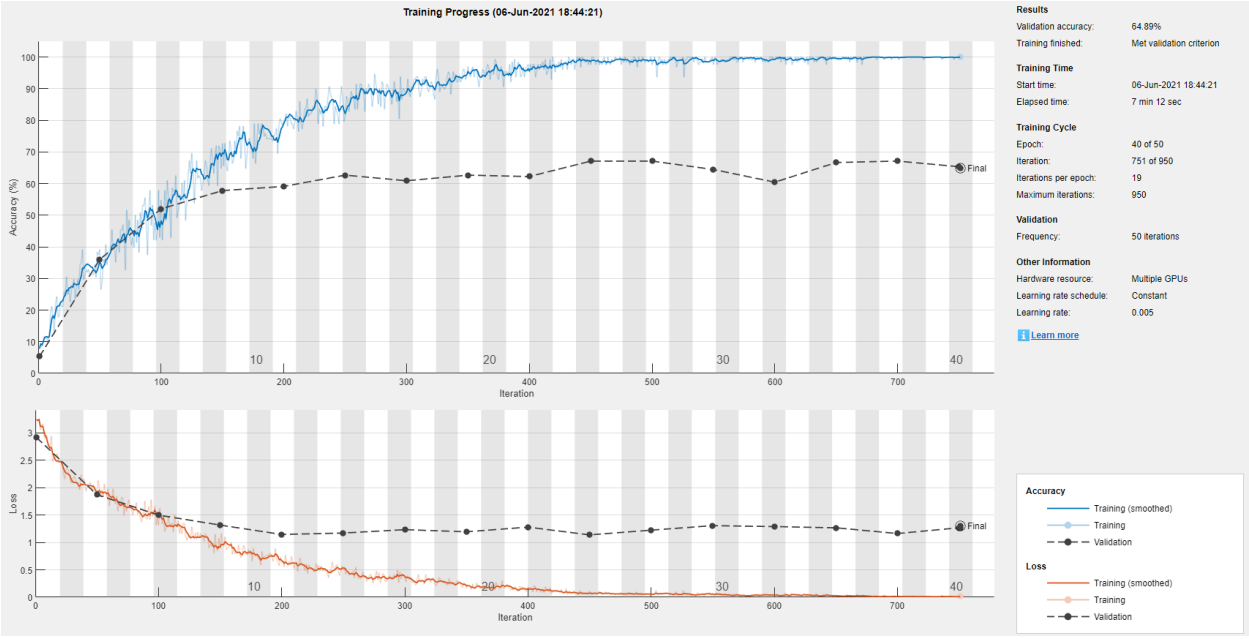
10 Networks

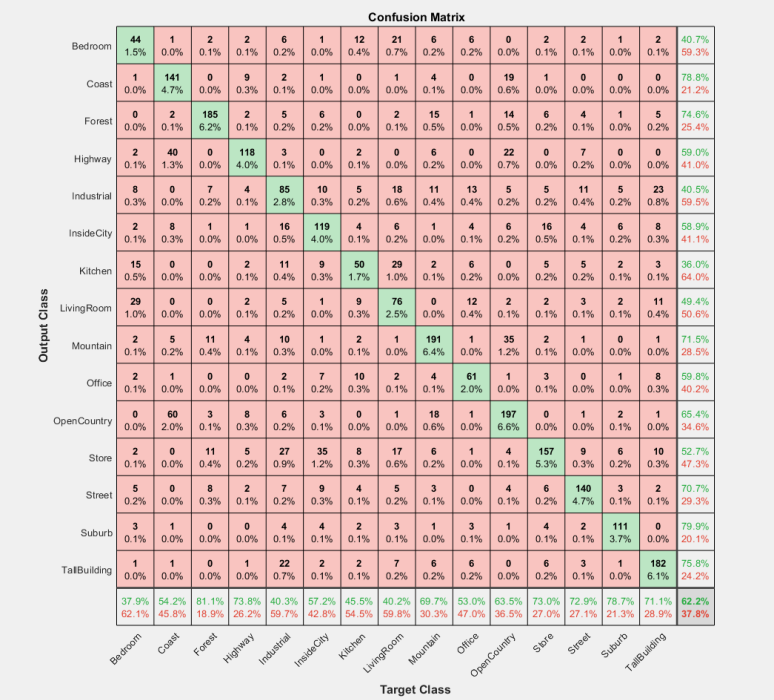
Training 7 Networks resulted in a mean accuracy value of 67.14 %.

Network	Accuracy
Net1	59.89%
Net2	59.73%
Net3	61.40%
Net4	56.51%
Net5	60.20%
Net6	62.21%
Net7	60.50%
Net8	62.37%
Net9	63.88%
Net10	59.32%

[Optional] - Data Augmentation

Adding random cropping and rotation to the network used in *step f.* allowed to improve accuracy to 62.2%.

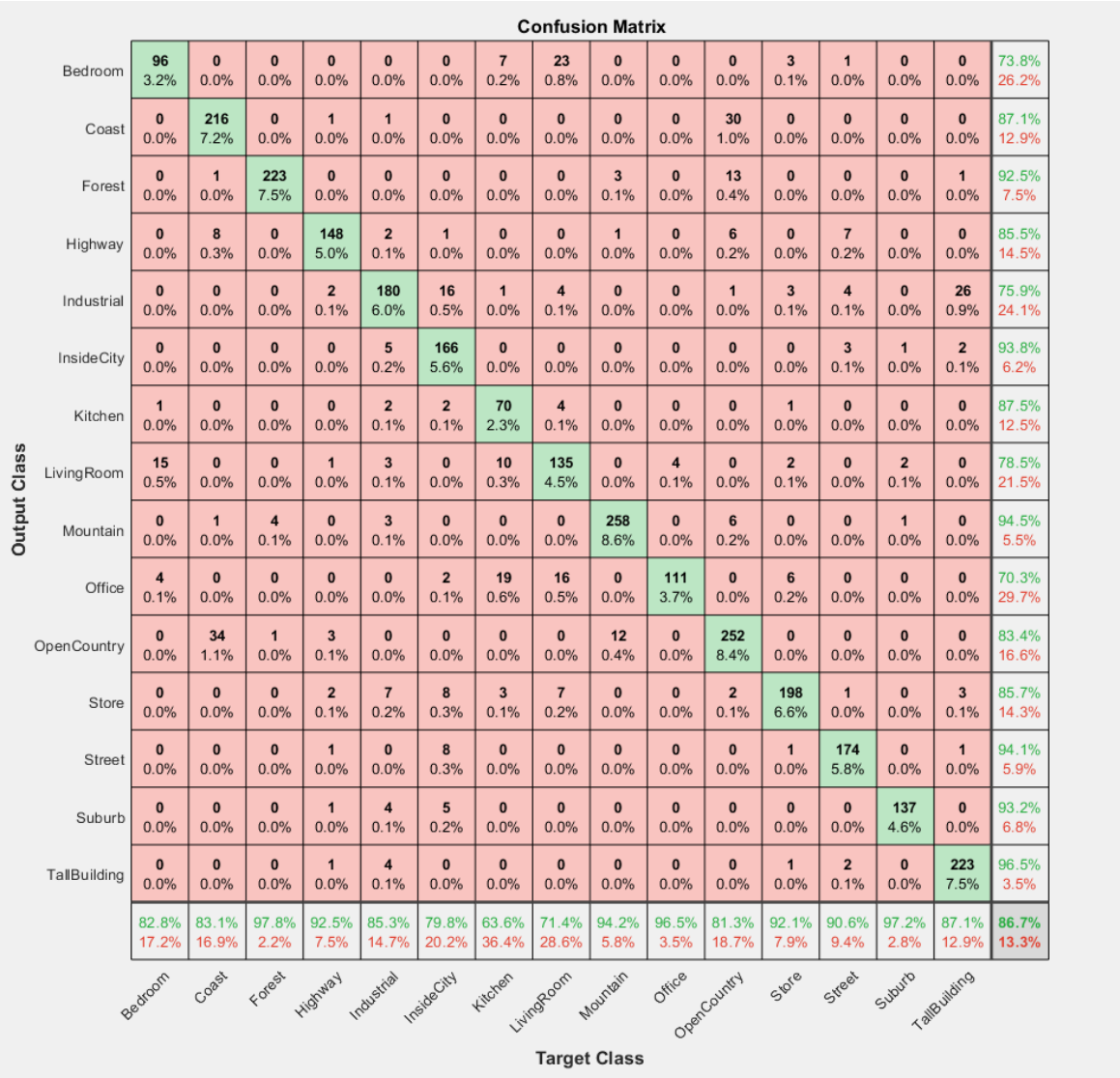




Task 3

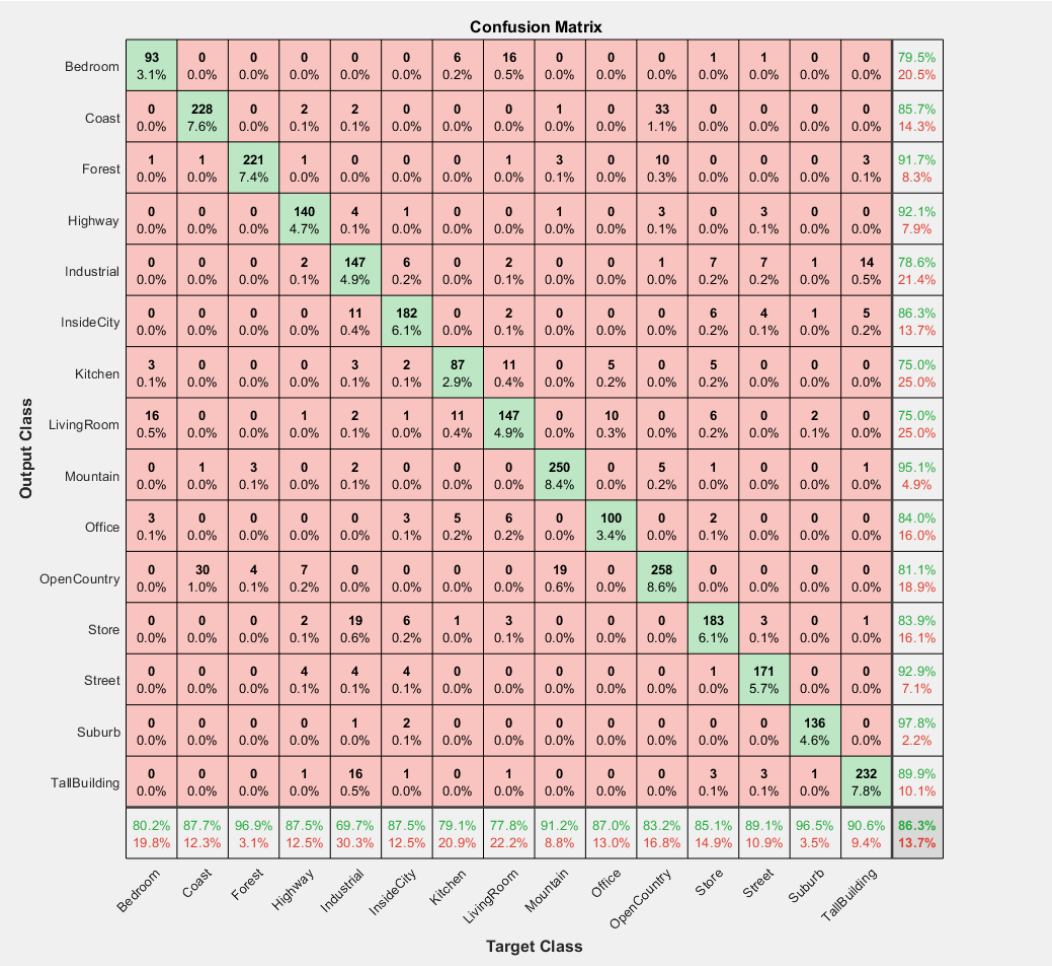
a. Weights Fine Tuning

Employing transfer learning resulted in a further improvement of test accuracy, with a value of 86.6% as expected.



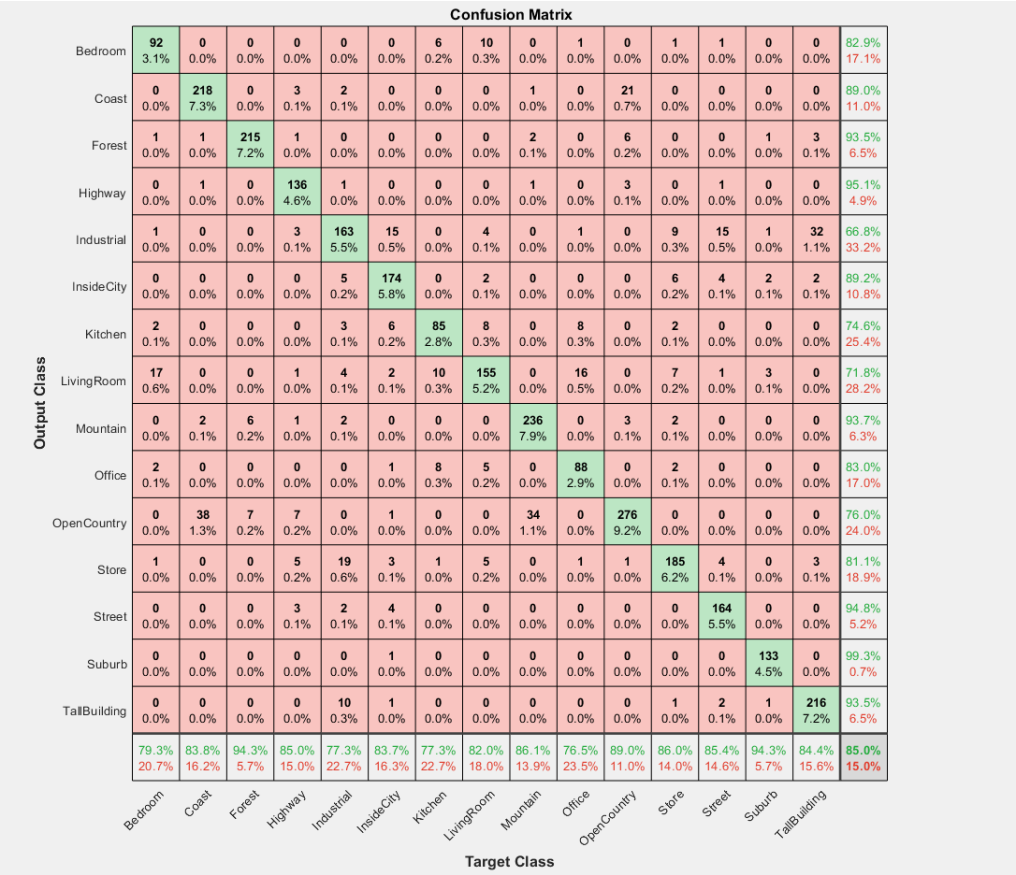
b. Multiclass SVM

Also with SVM the results obtained are aligned to the expectations delivering a test accuracy of 86.3%. The only drawback here is prediction time,which amounted to around 2 minutes, probably due to the poorly optimized DAG implementation as stated before.



[Optional] - ECOC implementation

Matlab also provides an ECOC approach implementation, with the function *fitcecoc*. The result is in line with the latter with an accuracy of 85%.



References

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.