UNIVERSITÀ DEGLI STUDI DI TRIESTE
Dipartimento di Ingegneria e Architettura

Robotics & AI

Image classification by means of a Convolutional Neural Network

Computer Vision project

Davide Vidmar

Gabriele Pittia

Academic year 2021 - 2022

Problem statement

The task we dealt with and that we brought to completion concerned a classification problem. There was a given set of classified images and our goal was to train an ANN capable of predicting the category the input image belonged to.

The provided dataset (from [Lazebnik et al., 2006]), contains the following 15 categories:

Office, Kitchen, Living room, Bedroom, Store, Industrial, Tall building, Inside city, Street, Highway, Coast, Open country, Mountain, Forest, Suburb.

Approach

Our approach consists of the training of a CNN as a classifier.
The first 85% of the training dataset has been isolated, the other 25% has been used to validate the model while the training itself was going on.
We started from a 'basic' shallow network (topology given by our assignment) and then we assessed the quality fluctuation brought on by some improvements, applied one after the other.
Such improvements are:

- Data augmentation:
  by introducing left-to-right reflection on the horizontal axis
- Batch normalization
- Playing with the optimization parameters (learning rate, weights regularization, minibatch size . . .)
- Dropout
- Employing an ensemble of five networks , trained independently.

We'll discuss only the ones that significantly improved the performance of our classifier.
The Optimization algorithm we used is Stochastic Gradient Descent Momentum first, then we moved also on 'Adam'.
We decided to "standardize" the number of epochs for each training to 10, although in some cases the loss function curve reaches a relatively steady value earlier.
Finally we implemented a transfer learning approach, at first by training last layers of the 'ResNet18" and then by extracting the features accessing the activation functions of one of the ResNet's layers and collecting new data for training a multi-class SVM.
We choose ResNet-18, by looking at the graph below ResNet-18 could be a good trade-off between training speed and accuracy.

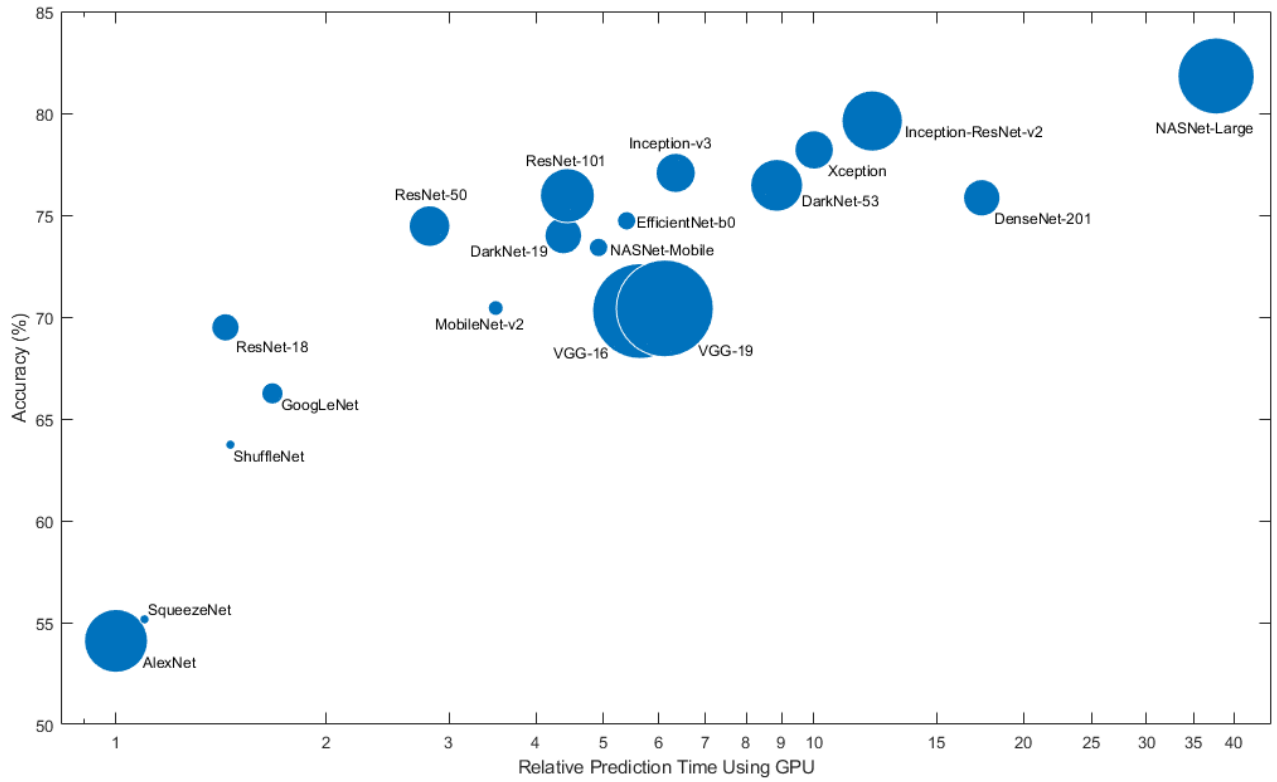


Image from [Mathworks]

Implementation choices

Our project has been implemented in Matlab.
We used the ‘Deep Learning Toolbox’, a set of libraries and UIs developed for such purposes.
We also worked with the “experiment Manager” Matlab Tool which permits us to run multiple simulations in order to assess different hyperparameters tunings.
[Code] is available on GitHub.

Results

Shallow Network

The ‘first try’ consists of a shallow network, described by the table:

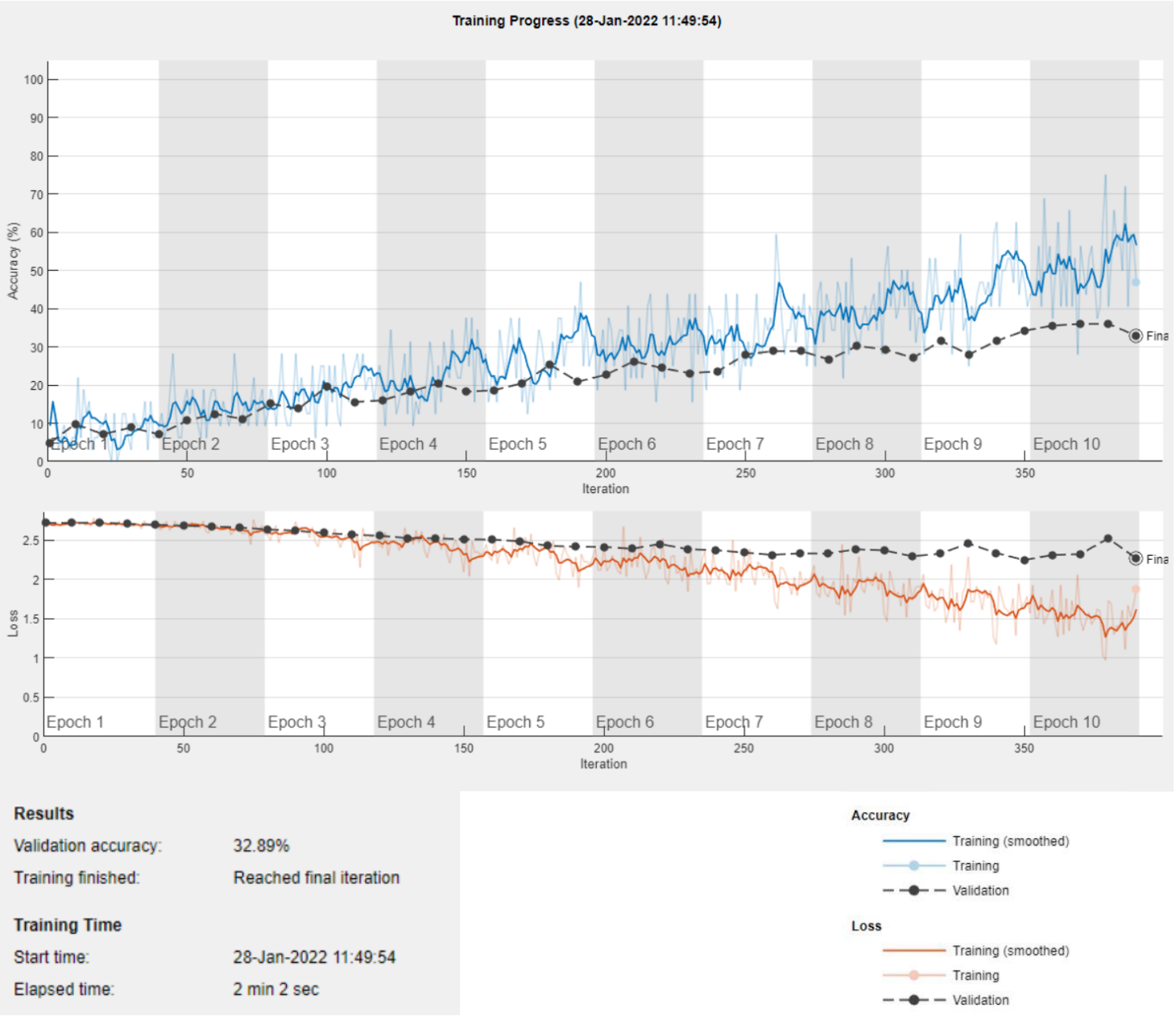
ANALYSIS RESULT

	Name	Type	Activations	Learnables
1	input 64×64×1 images with 'zerocenter' normalization	Image Input	64×64×1	-
2	conv_1 8 3×3 convolutions with stride [1 1] and padding 'sa…	Convolution	64×64×8	Weights 3×3×1×8 Bias 1×1×8
3	relu_1 ReLU	ReLU	64×64×8	-
4	maxpool_1 2×2 max pooling with stride [2 2] and padding [0 0 0…	Max Pooling	32×32×8	-
5	conv_2 16 3×3 convolutions with stride [1 1] and padding 's…	Convolution	32×32×16	Weights 3×3×8×16 Bias 1×1×16
6	relu_2 ReLU	ReLU	32×32×16	-
7	maxpool_2 2×2 max pooling with stride [2 2] and padding [0 0 0…	Max Pooling	16×16×16	-
8	conv_3 32 3×3 convolutions with stride [1 1] and padding 's…	Convolution	16×16×32	Weights 3×3×16×32 Bias 1×1×32
9	relu_3 ReLU	ReLU	16×16×32	-
10	fc_1 15 fully connected layer	Fully Connected	1×1×15	Weights 15×8192 Bias 15×1
11	softmax softmax	Softmax	1×1×15	-
12	output crossentropyex with 'Bedroom' and 14 other classes	Classification Output	-	-

We first made a training of the shallow network with parameters:

Key	**Value**
Algorithm	*sgdm*
InitialLearnRate	*0.01*
MaxEpochs	*7*
Shuffle	*every-epoch*
ValidationFrequency	*10*
MiniBatchSize	*32*

And results are described by the plot reported below,



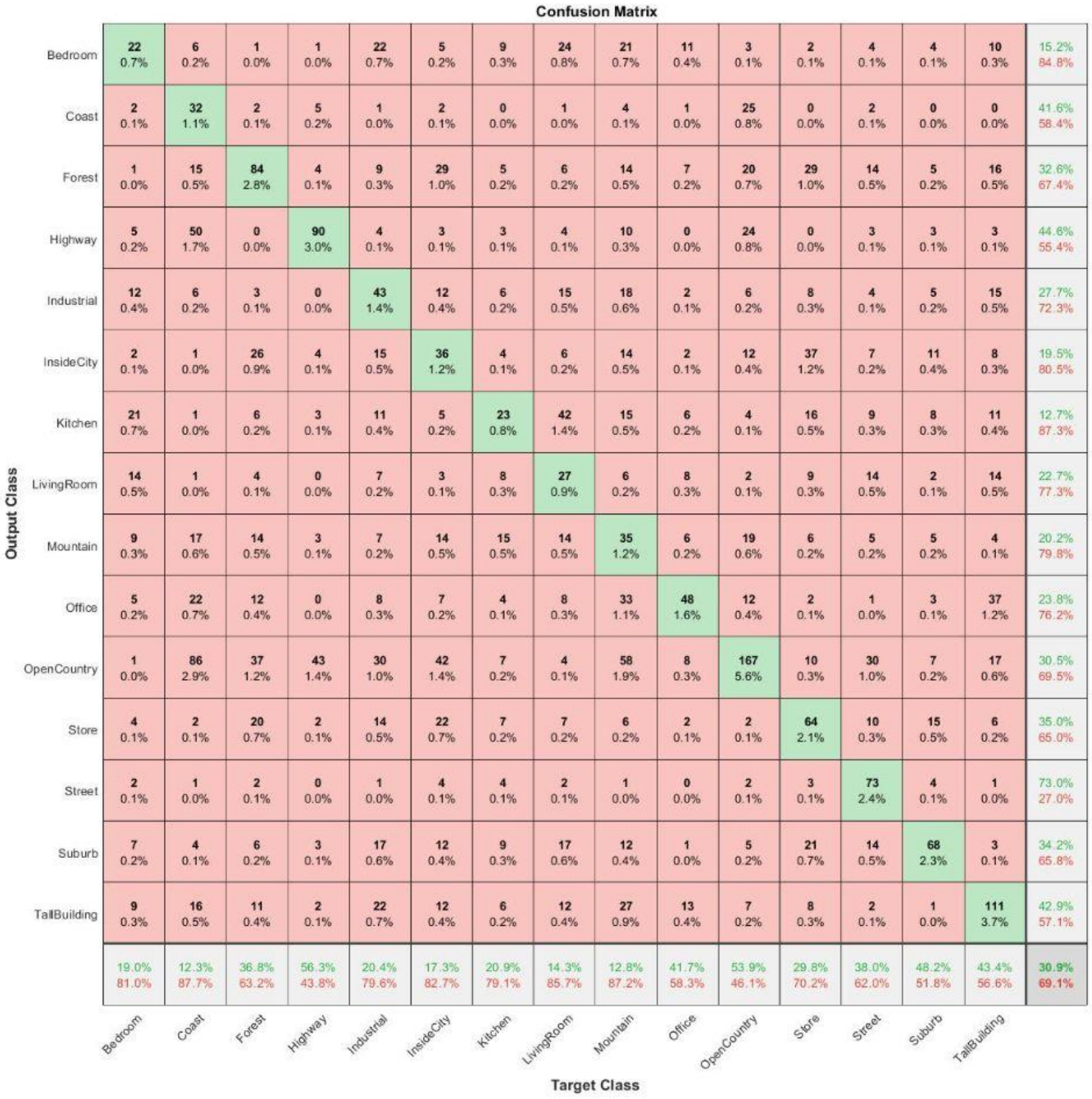
Comments

We can assess our training process by looking at the Accuracy (up), and Loss (below) plots; here are some considerations:

- By the end of the training, the model tends to overfit with training data; in fact the ‘blue line’ continues to increase while the dashed one reaches a sort of plateau. We can see overfitting phenomena either by checking the loss function plot.
- After the epoch #8 the Loss function is not monotonical. This happens to be a clear indicator for stopping the training, since no more knowledge can be extracted from data.

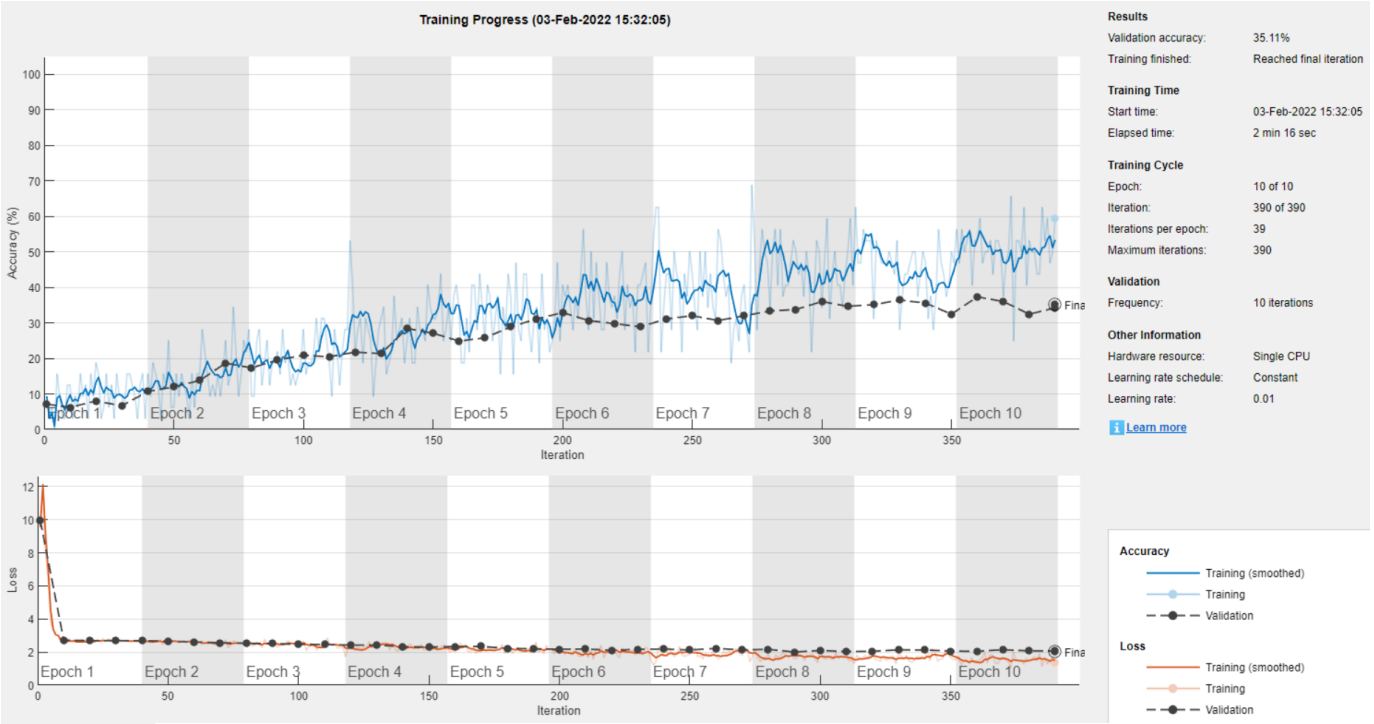
Testing our model on the test set

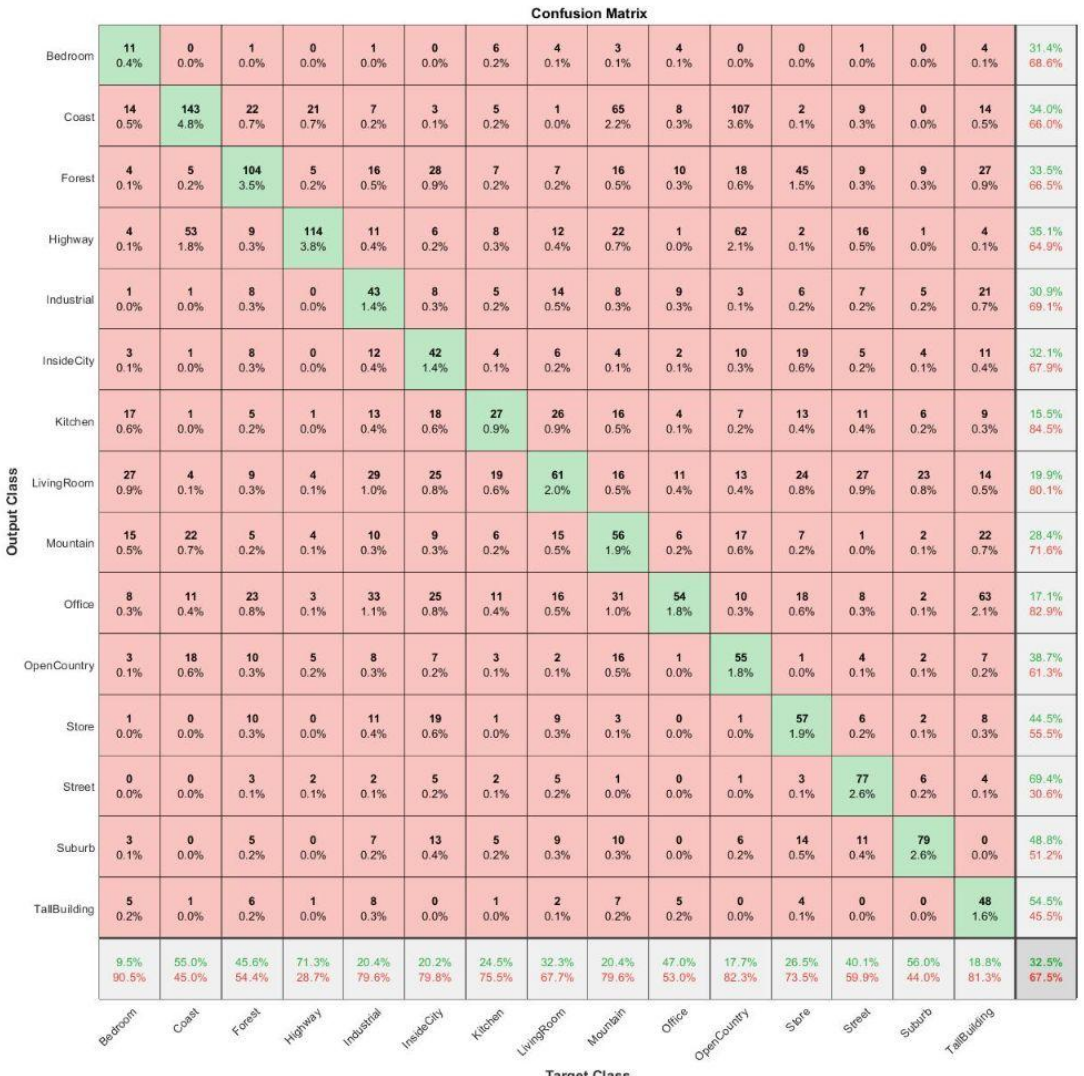
Since now our model did his training on data previously seen it's time to see how it works on new unseen data.
Here it is reported the confusion matrix plot: we can ensure a total accuracy of 30.9%



Adding left-to-right reflections

Next we moved on to some of the proposed improvements. The first one tried is about data preprocessing and it aims to simulate a larger set of data. It aims to trick the model by flipping on the vertical axis some randomly chosen images.
(accuracy validation 35.11%)



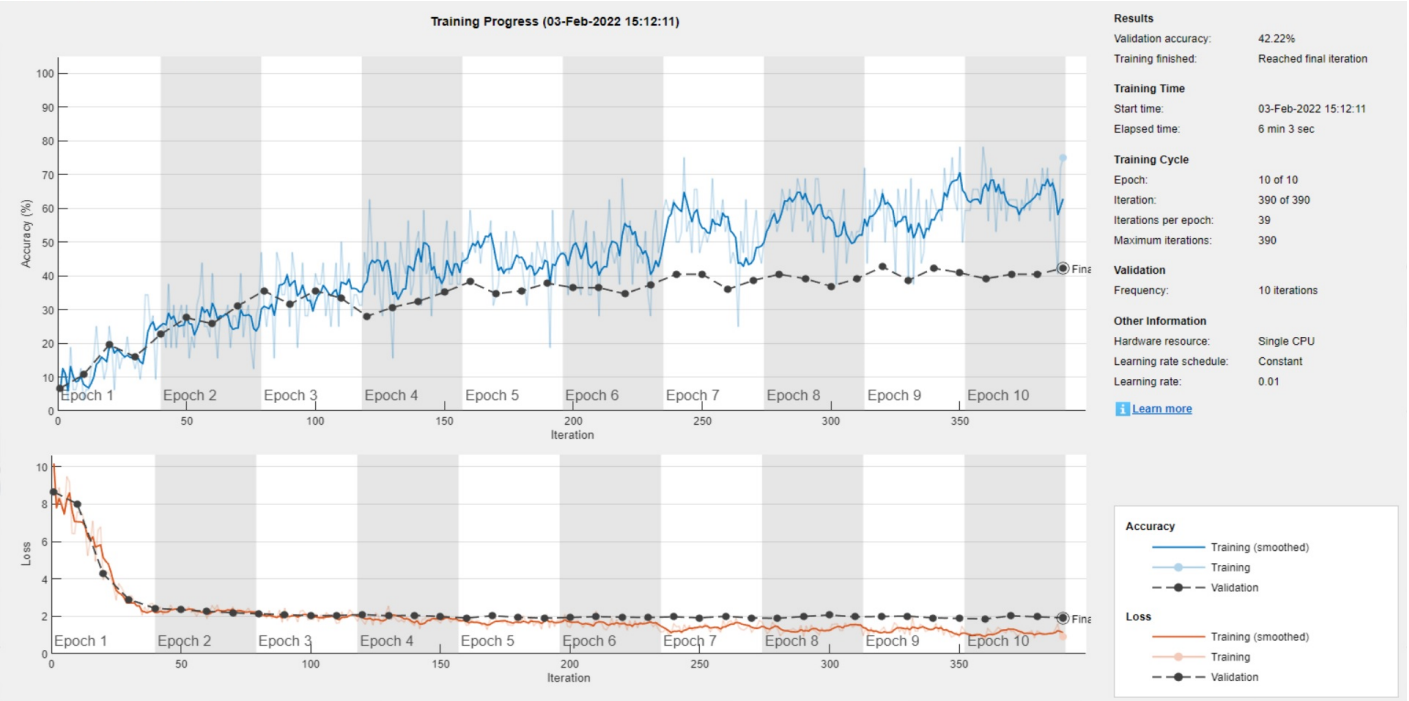


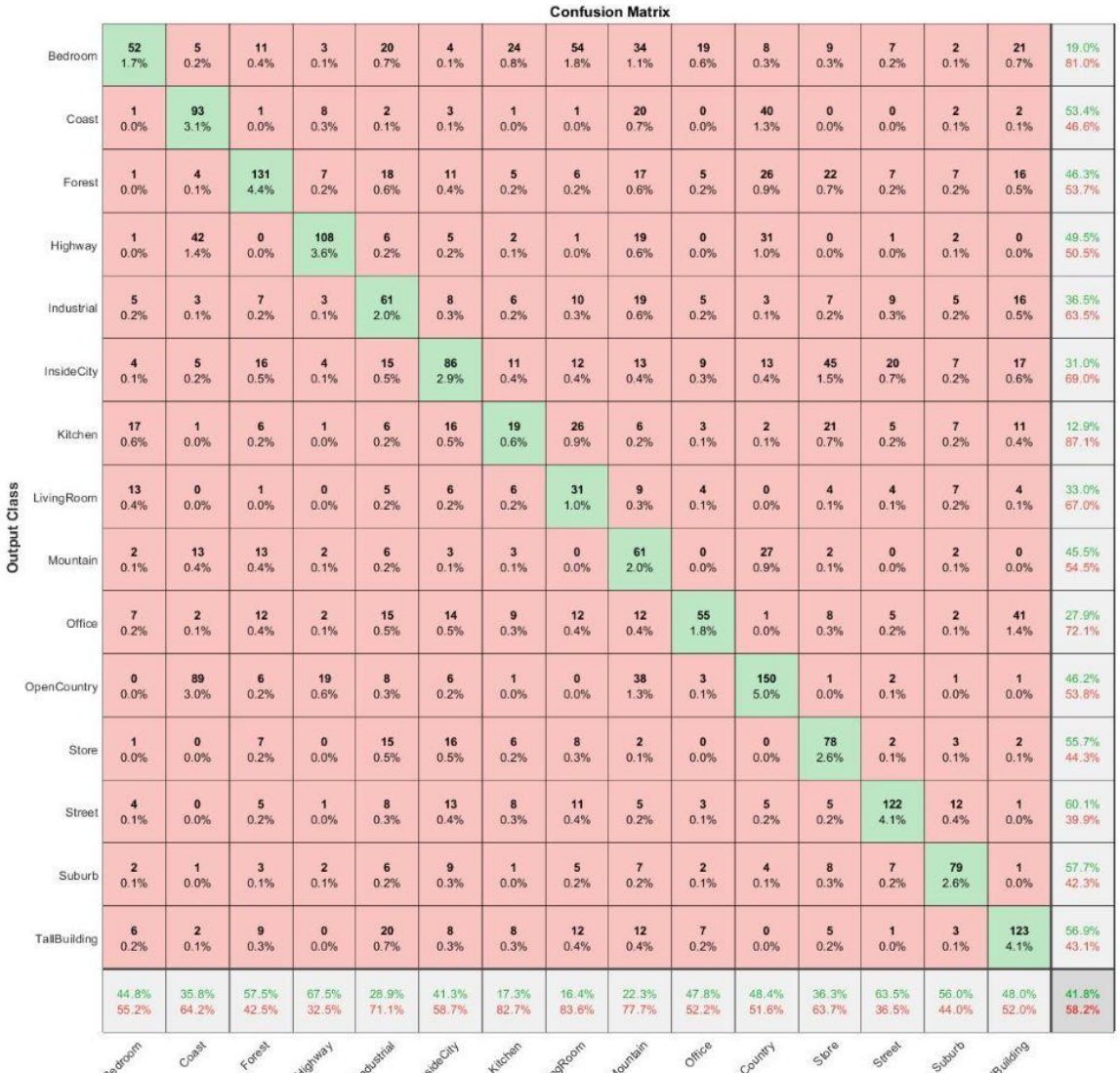
Comments

No particular improvements are noted.

Batch normalization

As suggested, we inserted batch normalization layers before each ReLu layer.



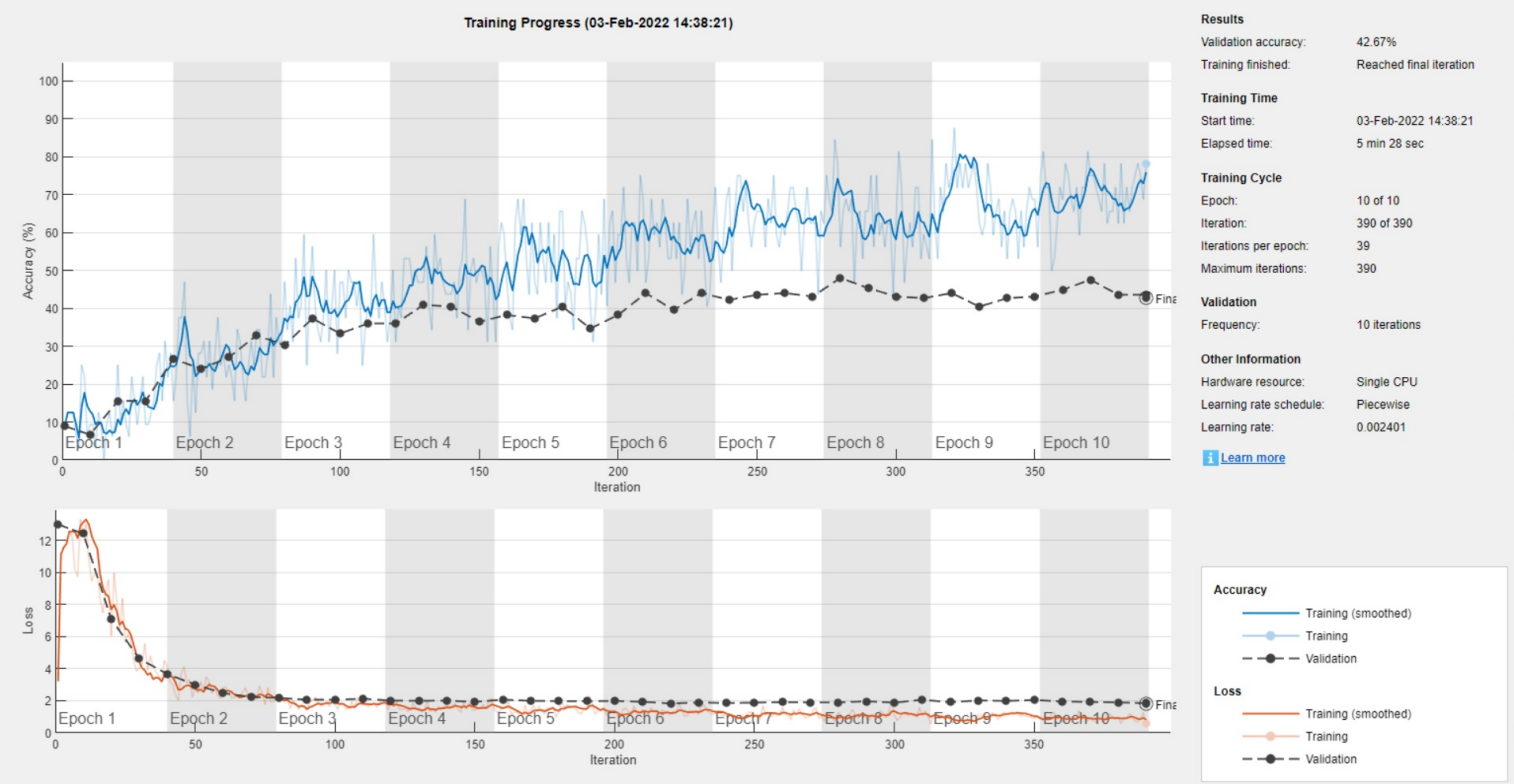


Comments

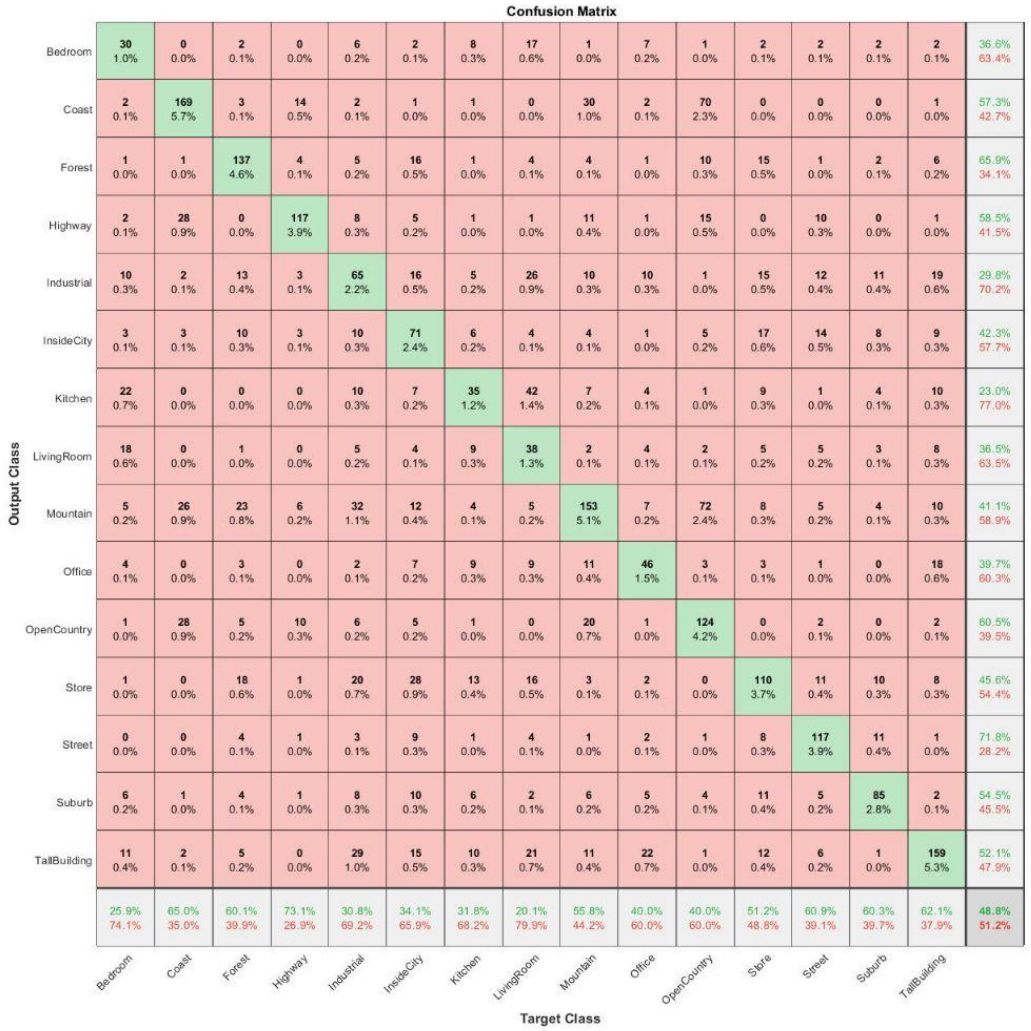
With batch normalization we achieved a significant improvement. Note how the loss function has become smoother.

Trying to ‘play’ with some parameters

The first modification we attempted to do was to switch to an *Adam* optimizer, then we also introduced a dynamic Learning Rate, which reduces every two epochs during the training by a factor of 0.7. Eventually we tried to initialize weights with the default option, which is called ‘Glorot’. We’re talking about an uniform distribution, dependent on layer parameters and dimensions.



During test, model reached 48.8% accuracy



Comments

Switching to Adam optimizer improved considerably the model performance, as also changing the weight initialization.

Adding some dropout

We decided to put two dropout layers before the layers that happen to have the greatest number of learnable parameters:

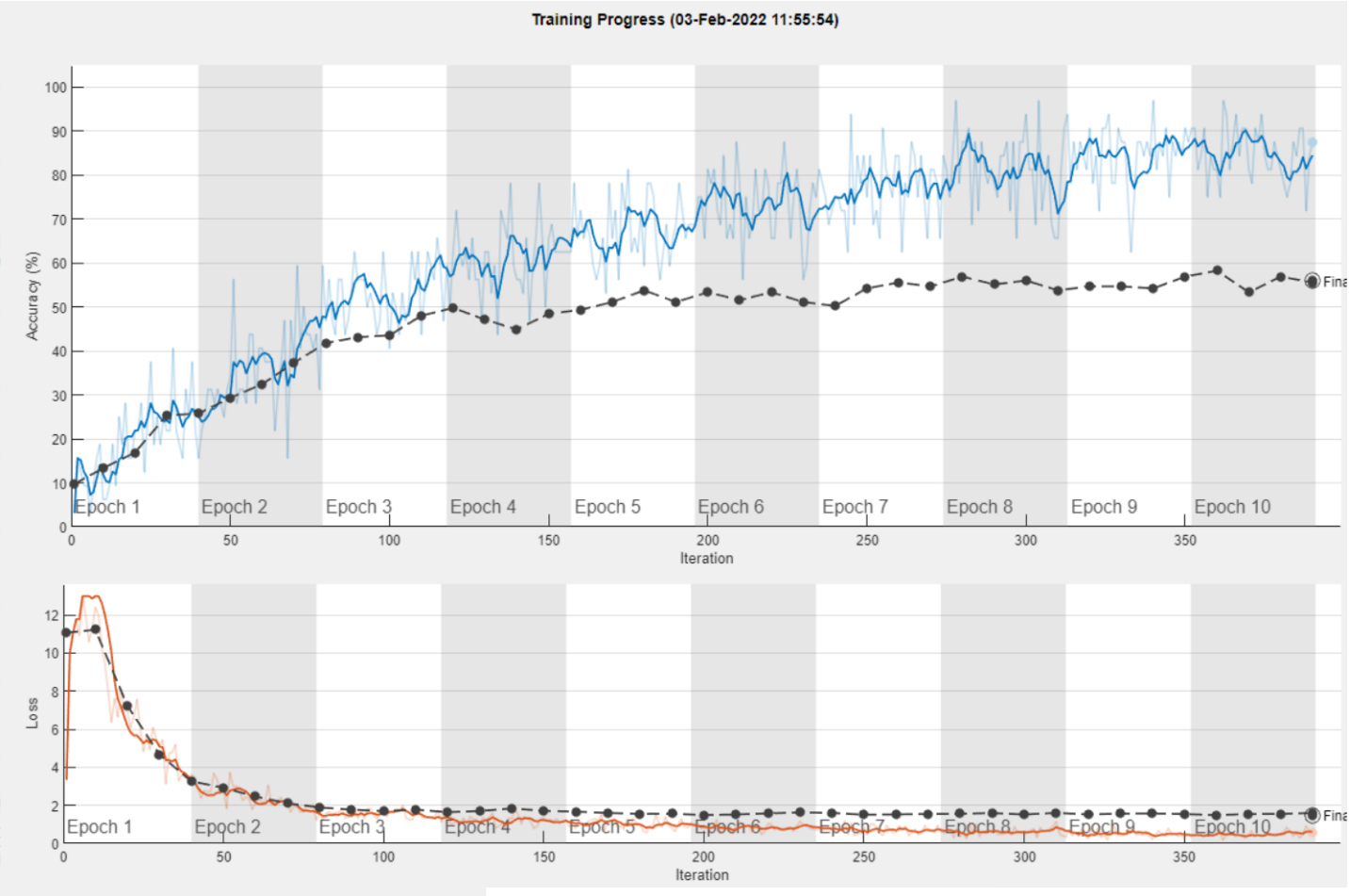
- the third convolutional layer
- the fully connected one

CNN structure is described by the table:

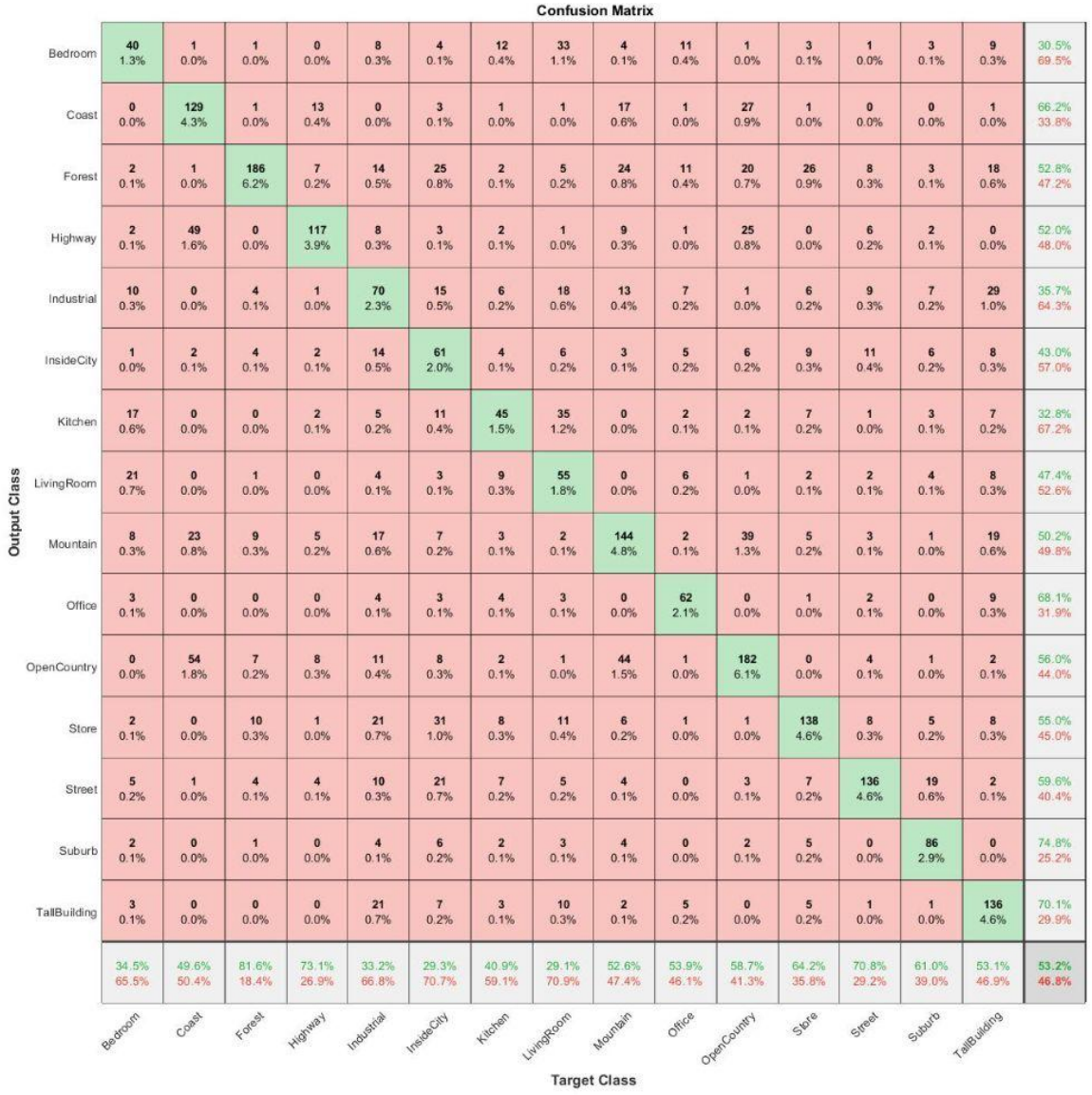
ANALYSIS RESULT

	Name	Type	Activations	Learnables
1	input 64×64×1 images with 'zerocenter' normalization	Image Input	64×64×1	-
2	conv_1 8 3×3 convolutions with stride [1 1] and padding 'same'	Convolution	64×64×8	Weights 3×3×1×8 Bias 1×1×8
3	norm1 Batch normalization	Batch Normalization	64×64×8	Offset 1×1×8 Scale 1×1×8
4	relu_1 ReLU	ReLU	64×64×8	-
5	maxpool_1 2×2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	32×32×8	-
6	conv_2 16 3×3 convolutions with stride [1 1] and padding 'same'	Convolution	32×32×16	Weights 3×3×8×16 Bias 1×1×16
7	norm2 Batch normalization	Batch Normalization	32×32×16	Offset 1×1×16 Scale 1×1×16
8	relu_2 ReLU	ReLU	32×32×16	-
9	maxpool_2 2×2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	16×16×16	-
10	dropout_1 25% dropout	Dropout	16×16×16	-
11	conv_3 32 3×3 convolutions with stride [1 1] and padding 'same'	Convolution	16×16×32	Weights 3×3×16×32 Bias 1×1×32
12	norm3 Batch normalization	Batch Normalization	16×16×32	Offset 1×1×32 Scale 1×1×32
13	relu_3 ReLU	ReLU	16×16×32	-
14	dropout_2 25% dropout	Dropout	16×16×32	-
15	fc_1 15 fully connected layer	Fully Connected	1×1×15	Weights 15×8192 Bias 15×1
16	softmax softmax	Softmax	1×1×15	-
17	output crossentropyex with 'Bedroom' and 14 other classes	Classification Output	-	-

And training process is described by the usual plot:



Right down there are results under testing phase (accuracy 53.2%)

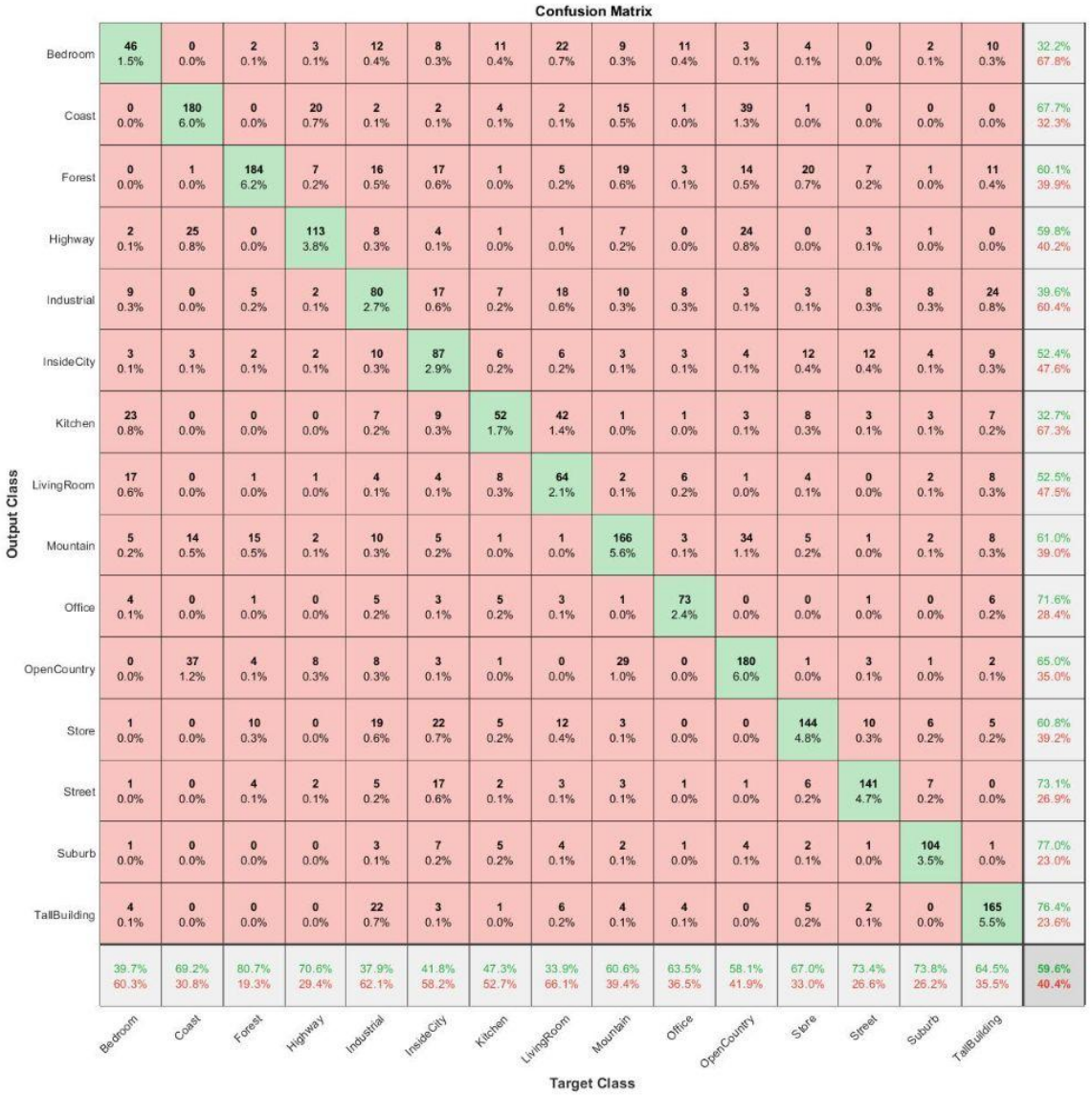


Comments

We gained some accuracy by using dropout, notice that we also tried to introduce L2 regularization but the default value (1e-4) has been revealed to be the best one.

Adding ensemble of networks

The final trick for raising our accuracy consists in an *ensemble of networks*, 10 networks have been trained independently and then, the 5 most accurate networks were extracted.
In the end, during the test phase, classification was accomplished by a “majority voting” poll by the 5 networks. We obtained 59.6% accuracy. (see majority vote function in Matlab script)



Comments

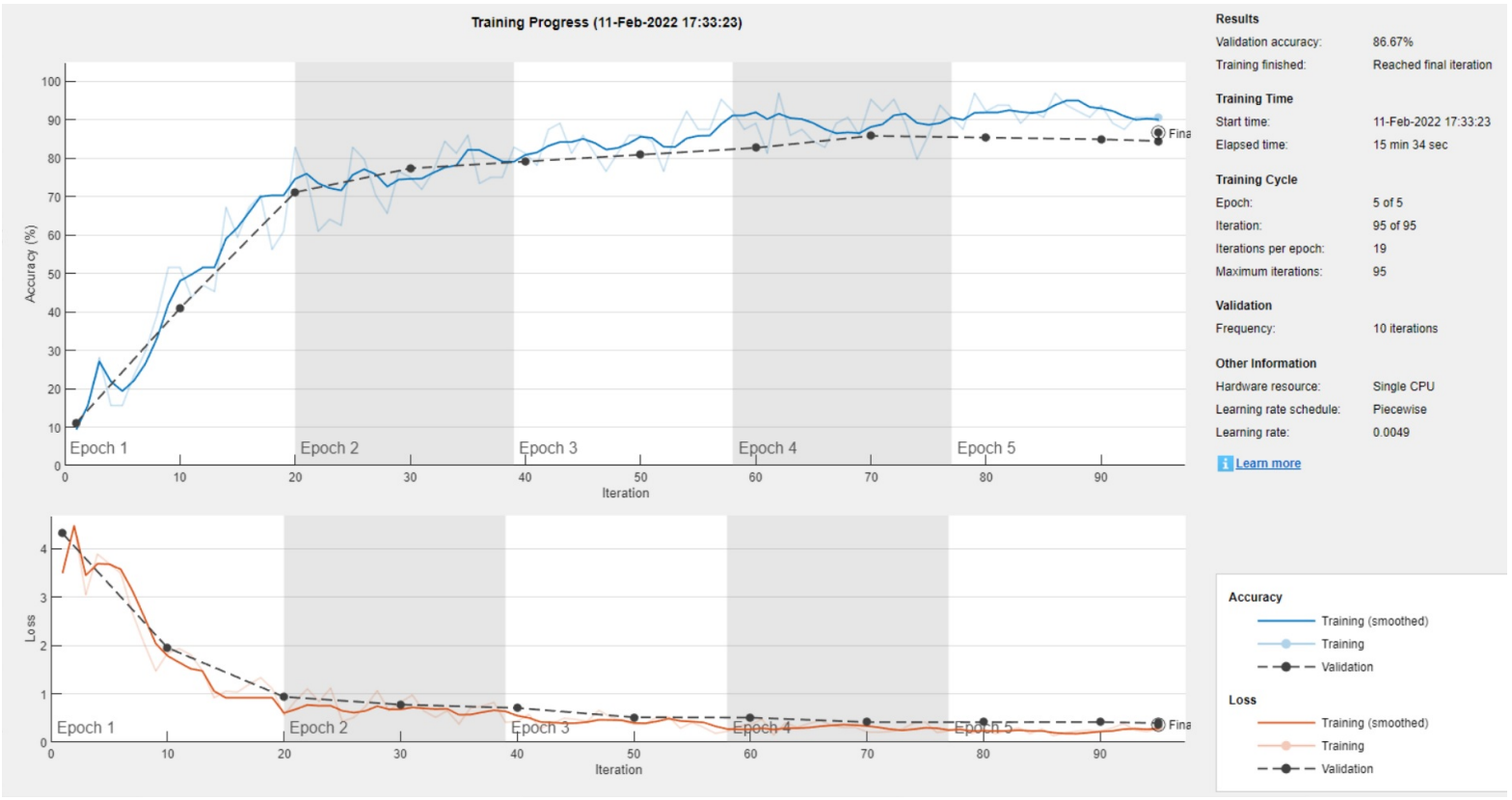
Wisdom of the crowd is the principle that tells how collective knowledge is better than knowledge of the few. In simple terms, it means that asking many people who individually have less knowledge is better than asking a few who have a lot of knowledge.
So we asked 5 networks to vote their choice and we can see that the accuracy rises.
We're not telling that five individuals form a crowd, but still more than one and results are consistent.

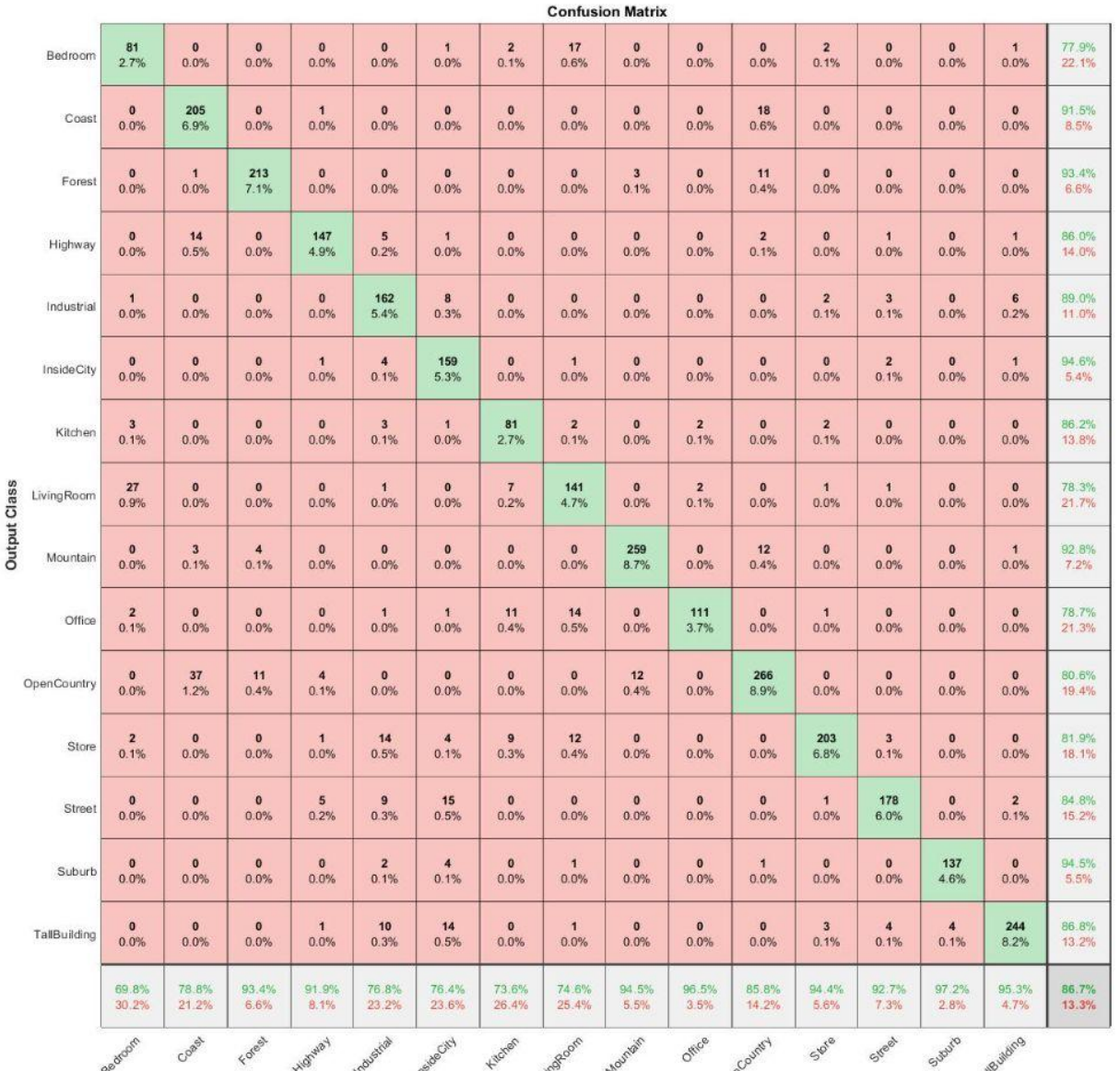
Transfer learning

Pre-trained network

For each layer of the ResNet, except the last ones, we freezed all the weights by setting the property ‘WightLearnRate” to 0.
We clearly also modified the number of classes according to our dataset.
Finally we trained the last fully connected layer with training options specified below:
Note: since training time was significantly longer, we trained our model for 5 epochs only.

Key	**Value**
Algorithm	*adam*
InitialLearnRate	*0.01*
MaxEpochs	*7*
Shuffle	*every-epoch*
ValidationFrequency	*10*
MiniBatchSize	*64*
LearnRateSchedule	*piecewise*
LearnRateDropPeriod	*2*
LearnRateDropFactor	*0.7*


Training Progress (11-Feb-2022 17:33:23)
Accuracy (%)
Iteration
Epoch 1
Epoch 2
Epoch 3
Epoch 4
Epoch 5
Loss
Results
Validation accuracy: 86.67%
Training finished: Reached final iteration
Training Time
Start time: 11-Feb-2022 17:33:23
Elapsed time: 15 min 34 sec
Training Cycle
Epoch: 5 of 5
Iteration: 95 of 95
Iterations per epoch: 19
Maximum iterations: 95
Validation
Frequency: 10 iterations
Other Information
Hardware resource: Single CPU
Learning rate schedule: Piecewise
Learning rate: 0.0049
Learn more
Accuracy
Training (smoothed)
Training
Validation
Loss
Training (smoothed)
Training
Validation


Confusion Matrix
Output Class
Bedroom 81 2.7% 0 0.0% 0 0.0% 0 0.0% 0 0.0% 1 0.0% 2 0.1% 17 0.6% 0 0.0% 0 0.0% 0 0.0% 2 0.1% 0 0.0% 0 0.0% 1 0.0% 77.9% 22.1%
Coast 0 0.0% 205 6.9% 0 0.0% 1 0.0% 0 0.0% 0 0.0% 0 0.0% 0 0.0% 0 0.0% 0 0.0% 18 0.6% 0 0.0% 0 0.0% 0 0.0% 0 0.0% 91.5% 8.5%
Forest 0 0.0% 1 0.0% 213 7.1% 0 0.0% 0 0.0% 0 0.0% 0 0.0% 0 0.0% 3 0.1% 0 0.0% 11 0.4% 0 0.0% 0 0.0% 0 0.0% 0 0.0% 93.4% 6.6%
Highway 0 0.0% 14 0.5% 0 0.0% 147 4.9% 5 0.2% 1 0.0% 0 0.0% 0 0.0% 0 0.0% 0 0.0% 2 0.1% 0 0.0% 1 0.0% 0 0.0% 1 0.0% 86.0% 14.0%
Industrial 1 0.0% 0 0.0% 0 0.0% 0 0.0% 162 5.4% 8 0.3% 0 0.0% 0 0.0% 0 0.0% 0 0.0% 0 0.0% 2 0.1% 3 0.1% 0 0.0% 6 0.2% 89.0% 11.0%
InsideCity 0 0.0% 0 0.0% 0 0.0% 1 0.0% 4 0.1% 159 5.3% 0 0.0% 1 0.0% 0 0.0% 0 0.0% 0 0.0% 0 0.0% 2 0.1% 0 0.0% 1 0.0% 94.6% 5.4%
Kitchen 3 0.1% 0 0.0% 0 0.0% 0 0.0% 3 0.1% 1 0.0% 81 2.7% 2 0.1% 0 0.0% 2 0.1% 0 0.0% 2 0.1% 0 0.0% 0 0.0% 0 0.0% 86.2% 13.8%
LivingRoom 27 0.9% 0 0.0% 0 0.0% 0 0.0% 1 0.0% 0 0.0% 7 0.2% 141 4.7% 0 0.0% 2 0.1% 0 0.0% 1 0.0% 1 0.0% 0 0.0% 0 0.0% 78.3% 21.7%
Mountain 0 0.0% 3 0.1% 4 0.1% 0 0.0% 0 0.0% 0 0.0% 0 0.0% 0 0.0% 259 8.7% 0 0.0% 12 0.4% 0 0.0% 0 0.0% 0 0.0% 1 0.0% 92.8% 7.2%
Office 2 0.1% 0 0.0% 0 0.0% 0 0.0% 1 0.0% 1 0.0% 11 0.4% 14 0.5% 0 0.0% 111 3.7% 0 0.0% 1 0.0% 0 0.0% 0 0.0% 0 0.0% 78.7% 21.3%
OpenCountry 0 0.0% 37 1.2% 11 0.4% 4 0.1% 0 0.0% 0 0.0% 0 0.0% 0 0.0% 12 0.4% 0 0.0% 266 8.9% 0 0.0% 0 0.0% 0 0.0% 0 0.0% 80.6% 19.4%
Store 2 0.1% 0 0.0% 0 0.0% 1 0.0% 14 0.5% 4 0.1% 9 0.3% 12 0.4% 0 0.0% 0 0.0% 0 0.0% 203 6.8% 3 0.1% 0 0.0% 0 0.0% 81.9% 18.1%
Street 0 0.0% 0 0.0% 0 0.0% 5 0.2% 9 0.3% 15 0.5% 0 0.0% 0 0.0% 0 0.0% 0 0.0% 0 0.0% 1 0.0% 178 6.0% 0 0.0% 2 0.1% 84.8% 15.2%
Suburb 0 0.0% 0 0.0% 0 0.0% 0 0.0% 2 0.1% 4 0.1% 0 0.0% 1 0.0% 0 0.0% 0 0.0% 1 0.0% 0 0.0% 0 0.0% 137 4.6% 0 0.0% 94.5% 5.5%
TallBuilding 0 0.0% 0 0.0% 0 0.0% 1 0.0% 10 0.3% 14 0.5% 0 0.0% 1 0.0% 0 0.0% 0 0.0% 0 0.0% 3 0.1% 4 0.1% 4 0.1% 244 8.2% 86.8% 13.2%
69.8% 30.2% 78.8% 21.2% 93.4% 6.6% 91.9% 8.1% 76.8% 23.2% 76.4% 23.6% 73.6% 26.4% 74.6% 25.4% 94.5% 5.5% 96.5% 3.5% 85.8% 14.2% 94.4% 5.6% 92.7% 7.3% 97.2% 2.8% 95.3% 4.7% 86.7% 13.3%
Bedroom
Coast
Forest
Highway
Industrial
InsideCity
Kitchen
LivingRoom
Mountain
Office
OpenCountry
Store
Street
Suburb
TallBuilding

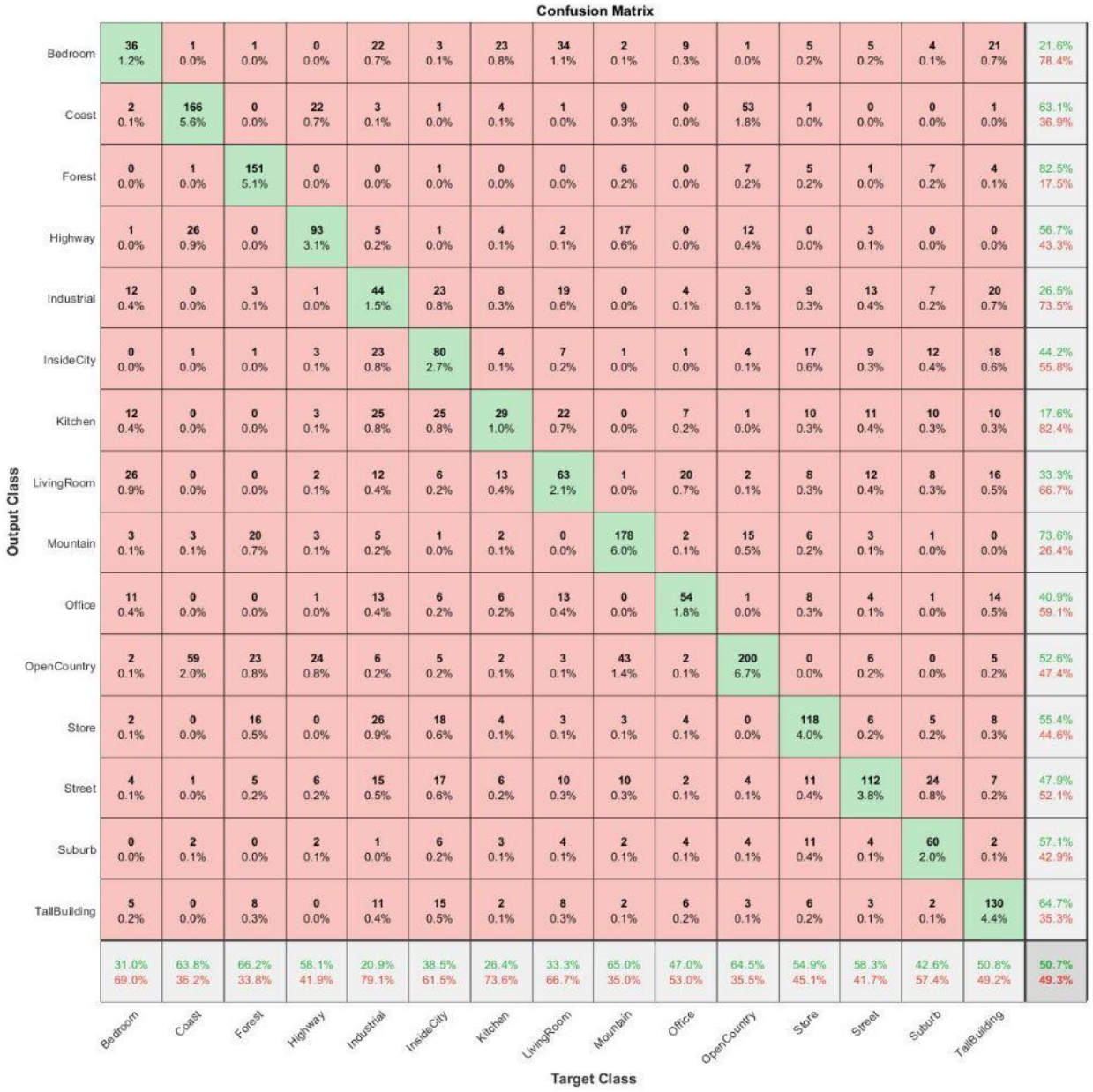
And during the test phase we reached 86.7% of accuracy.

Comments

By means of a pre-trained network our accuracy improved significantly.
Also Loss Function curve appears to be ‘smoother’ which indicates good and robust convergence to the minimum.

Feature extraction

We extracted from one of the last layers (*‘res5a’*) values from the activation functions, in order to train a multi class SVM.



We decided to implement the Error Correcting Output Code approach [Dietterich and Bakiri, 1994, James and Hastie, 1998].

“The **Error-Correcting Output Codes** method is a technique that allows a multi-class classification problem to be reframed as multiple binary classification problems, allowing the use of native binary classification models to be used directly.
Unlike one-vs-rest and one-vs-one methods that offer a similar solution by dividing a multi-class classification problem into a fixed number of binary classification problems, the error-correcting output codes technique allows each class to be encoded as an arbitrary number of binary classification problems. When an overdetermined representation is used, it allows the extra models to act as “error-correction” predictions that can result in better predictive performance.”

From machinelearningmastery.com

We implemented that approach by using the matlab function fitcecoc, which uses ‘*one vs one*’ as default multi-class SVM method.

Comments

By looking at the accuracy index, we can state that this technique doesn’t prove to be effective in that case. Although many runs with different parameters and different layers, results are considerably poorer than the ones achieved by employing the pre-trained network and freezing its weights.

Conclusions

Transfer learning is the undisputed winner of the accuracy contest.
All the other tries did not match his accuracy and scored way below, not overcoming a 60%, something that is just a little better than flipping a coin.
That’s for sure a matter of topology, available processing power, and research dedicated time.
We humbly couldn’t match a NN studied to solve the problem with our “trial and error” parameter modifications of a shallow one. Also probably that wasn’t even expected.
So, in the end, adopting something developed for the purpose is definitely the winning strategy.

References

- [Dietterich and Bakiri, 1994] Dietterich, T. G. and Bakiri, G. (1994). Solving multiclass learning problems via error-correcting output codes. Journal of artificial intelligence research, 2:263–286.
- [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.
- https://www.ccs.neu.edu/home/vip/teach/MLcourse/4_boosting/lecture_notes/ecoc/ecoc.pdf

Code

- https://github.com/dew54/computer-vision.git