DSSC - CVPR

Grimaldi Marco Alberto

February 2020

Introduction

This project requires the implementation of an image classifier based on convolutional neural networks. It's dived in three parts which I'll cover one at time. The data-set is given and has been already split into training and test sets. I used MATLAB.

Common Setup

For all steps and scripts there are some common features like:

- Input resizing: to match different CNN 1^{st} layer proprieties (in all three dimensions), I used "imresize", "repmat" and "augmentedImageDatastore" option.
- Input normalization: is almost always automatically done by 1^{st} layer.
- Data augmentation: to avoid overfitting and improve performance; I used translations, rotations, shear and reflection along vertical axe.
- Early stopping criteria: also to avoid overfitting, I used "MaxEpoch" and/or "Patience" on Validation Set Loss.
- Train-Validation split: 85%-15%.

Part 1

Just followed given instruction, only interesting part is recognize that (without using data augmentation) after a couple of epoch Validation Loss start to increase while Training Loss is still decreasing (same but reversed on accuracy score), meaning that the CNN is no more learning features but just overfitting on the Training Set.

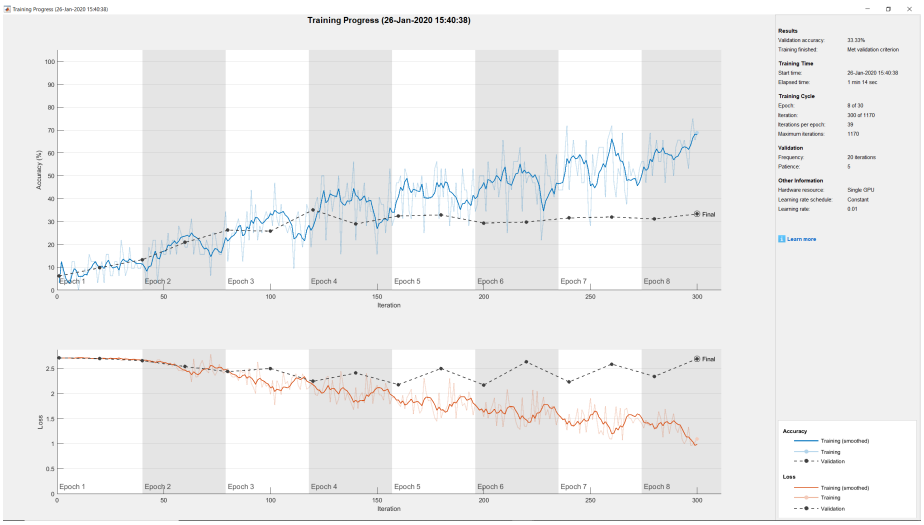

Figure 1: Shallow NN

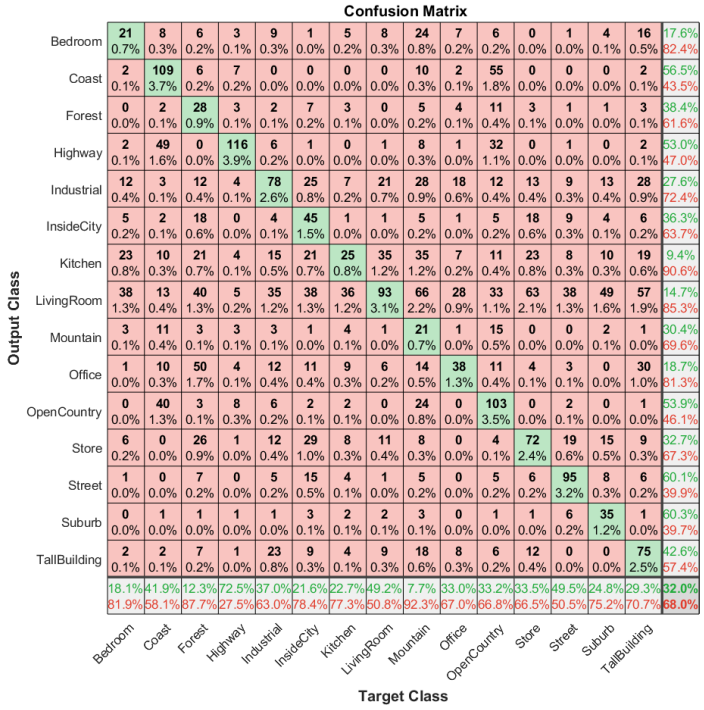

Figure 2: Shallow NN

The overall validation accuracy is around 33%. Test accuracy around 32%.

Part 2

For that part the goal was to improve the CNN with some different techniques. In addition to data-augmentation I added a couple more of convulutional layers and also introduced batch-normalization and dropout layers.

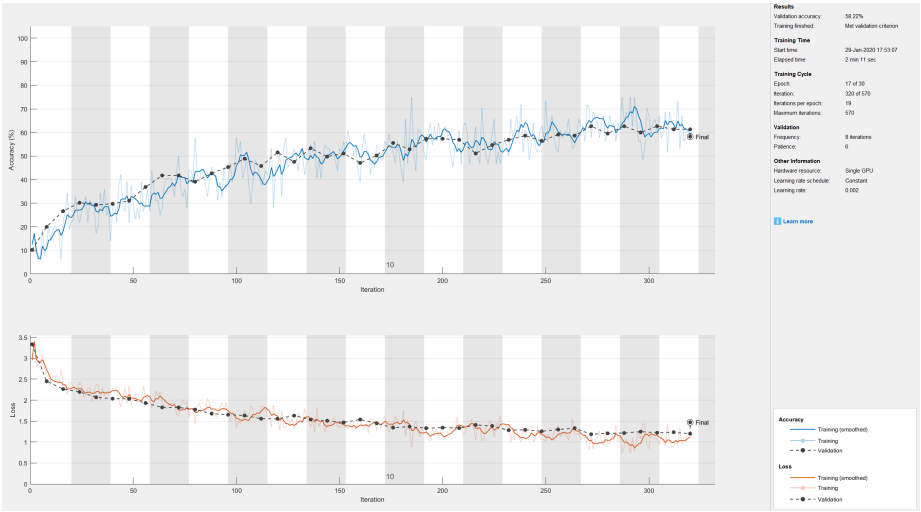

Figure 3: Improved NN

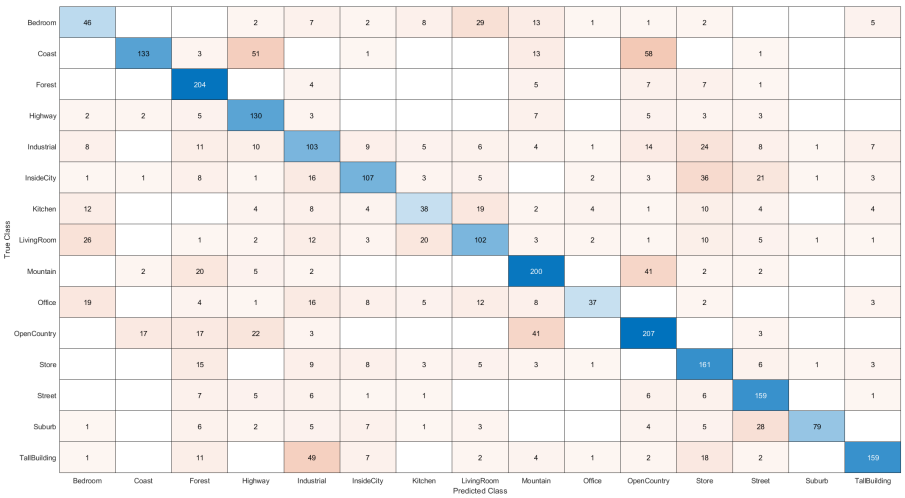

Figure 4: Improved NN

The overall validation accuracy is around 58% and test accuracy is around 57%. Notice how all the new features prevent the CNN to overfit the training set, reaching a kind of plateau with train and validation losses lines really close to each over.

I've also tried to use an ensemble of 10 of the previous NN.

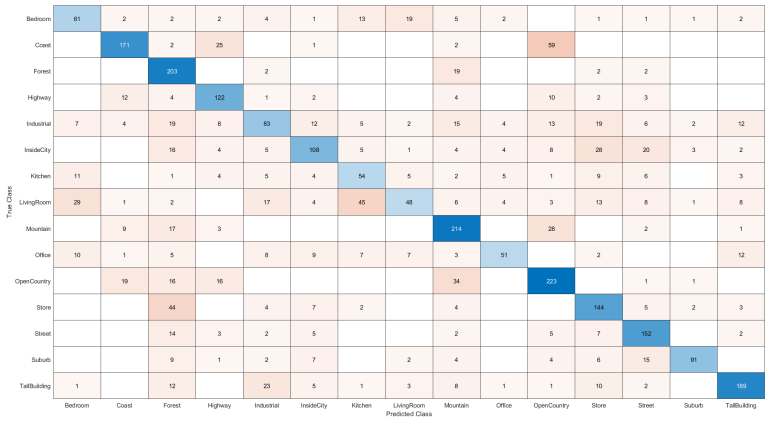

Figure 5: Ensemble of improved NN

I've reached a test accuracy a bit over 64% using the arithmetic means of the output classification score from each CNN.

Part 3

Lastly was required to adopt a Transfer Learning approach importing using trained CNNs. There are two approach for that:

- Replace last Full-Connected layer with another one (with the right dimension for our classification problem) and then fine-tune the derived CNN.
- Get activators from last Convolutional layer and then train a SVM to get classification.

I tried first approach with AlexNet and VGG, for AlexNet I tried both freezing all layers except the new one and no freeze but with 0.1x of learning rate and 20x LearnFactor on last layer. For VGG the only possible option due HW limitation was the first one.

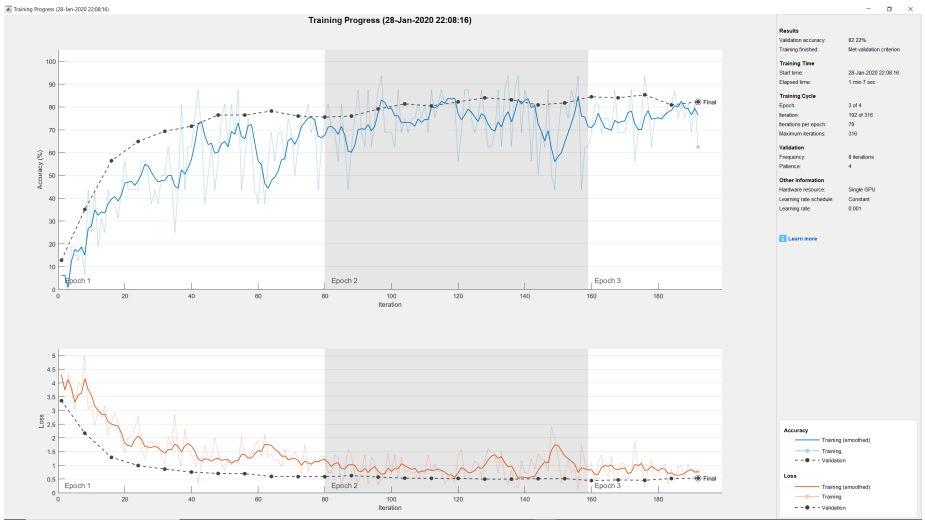

Figure 6: AlexNet Freeze

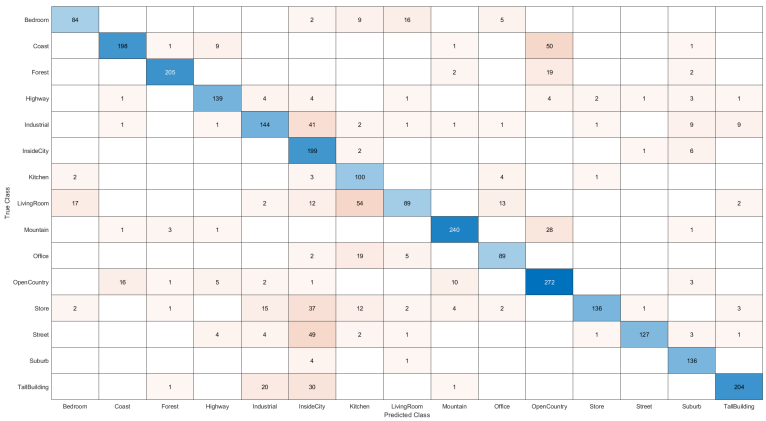

Figure 7: AlexNet Freeze

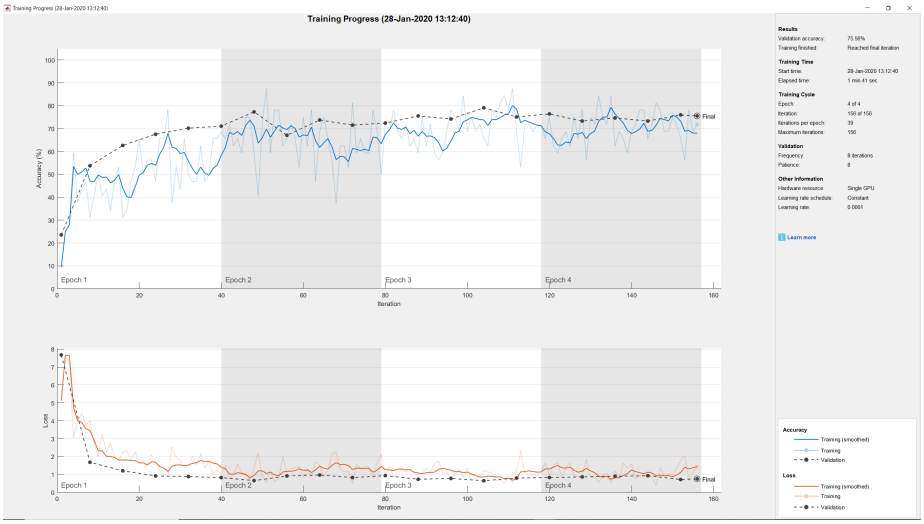

Figure 8: AlexNet No Freeze

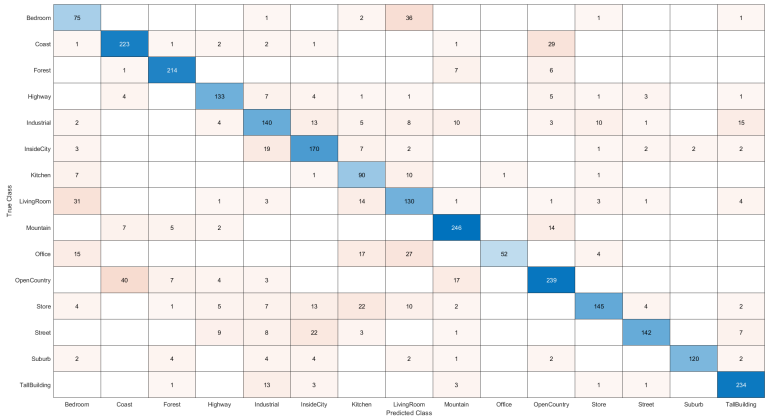

Figure 9: AlexNet No Freeze

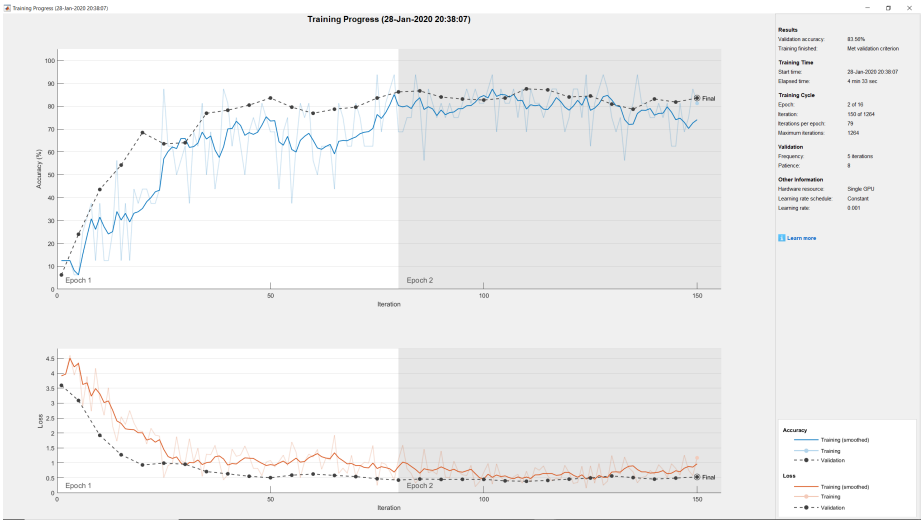

Figure 10: VGG

For second approach I used Resnet18 and Error-Correcting Output Codes Model for training the SVM classifier.

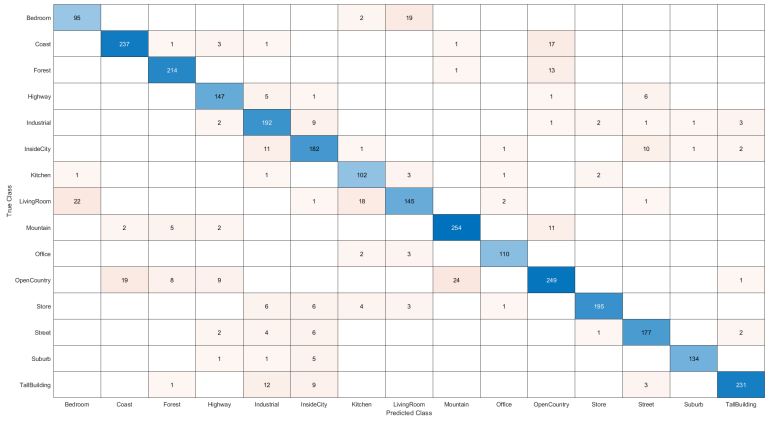

Figure 11: SMV

All approaches went beyond 75% accuracy, with the SVM almost crossing 90% accuracy test score. Notice also that no overfitting is present (which can be a common problem when dealing with transfer learning due to the imbalance between the huge prior training set and the new one) this is mostly thanks to data augmentation applied on training set and early-stopping criterion on training routine.