COMPUTER VISION AND PATTERN RECOGNITION PROJECT
CNN CLASSIFIER

Alessandro Cesa

January 2024

Contents

Introduction

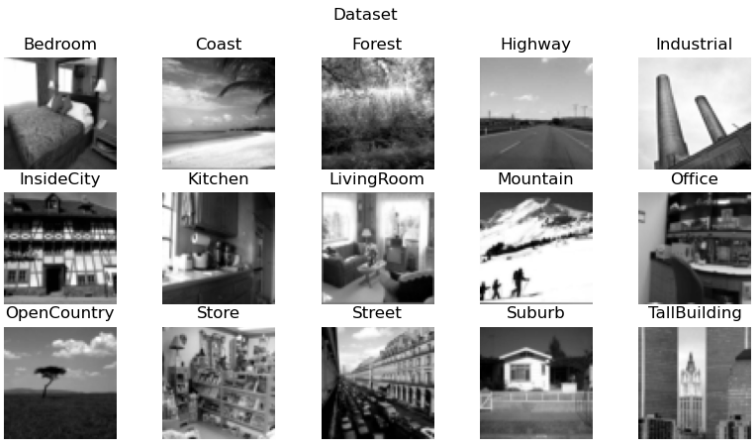
The final project for the course in Computer Vison and Pattern Recognition consists in a classification task on a provided dataset containing 15 categories of images [4]. The classification has to be carried out by employng a Convolutional Neural Network, built following given specifications and steps described in the project assignement.

Problem statement

The project requires the implementation of an image classifier based on Convolutional Neural Networks, following three major tasks.

Dataset

The Dataset is a collection of gray-scale images of 15 different categories: 'Bedroom', 'Coast', 'Forest', 'Highway', 'Industrial', 'InsideCity', 'Kitchen', 'LivingRoom', 'Mountain', 'Office', 'OpenCountry', 'Store', 'Street', 'Suburb', 'TallBuilding'. It consists of 1500 images for the training set and 2985 for the testing set, making up a total of 4485 images around 300 images for each of the 15 categories. In the training set each category is represented by exactly 100 images, while in the testing set this number varies. The training set was eventually divided into 85% for actual training and 15% for validation, so the actual training was performed on 1275 images.



Task 1: Basic CNN

Build from scratch a Convolutional Neural Network following this architecture:

Convolutional Neural Network Architecture

Layer	**Size**
Image Input	$64 \times 64 \times 1$
Convolution	$8 \times 3 \times 3$
ReLU	-
Max Pooling	2×2
Convolution	$16 \times 3 \times 3$
ReLU	-
Max Pooling	2×2
Convolution	$32 \times 3 \times 3$
ReLU	-
Fully Connected	15
Softmax	-
Classification Output	-

Following these specifications:

- Rescale the images to 64x64
- Split the provided training set in 85% for actual training set and 15% to be used as validation set;
- Employ the stochastic gradient descent with momentum optimization algorithm, using the default parameters of the library you use, except for those specified in the following;
- Use minibatches of size 32 and initial weights drawn from a Gaussian distribution with a mean of 0 and a standard deviation of 0.01; set the initial bias values to 0;

It's required to reach an accuracy of 30%

Task 2: Improved CNN

Improve the previous result, according to the following suggestions :

- **Data Augmentation:** Using left-to-right reflections, reach an accuracy of about 40%.
- **Batch Normalization [2]:** Add batch normalization layers before the ReLU layers.
- **Change Convolutional Filters:** Change the size and/or the number of the convolutional filters.
- **Optimization Parameters:** Play with the optimization parameters (learning rate, weights initialization, weights regularization, minibatch size, etc.), and switch to the Adam optimizer.
- **Dropout:** Add dropout layers.
- **Ensemble of Networks:** Employ an ensemble of networks (five to ten), trained independently. Use the arithmetic average of the outputs to assign the class, as in [5].

It's required to reach an accuracy of around 60%

Task 3: Transfer Learning

Use transfer learning in two manners:

- Freezing the weights of all the layers but the last fully connected layer and fine-tuning the weights of the last layer based on the same train and validation sets employed before;
- Employing the pre-trained network as a feature extractor, accessing the activation of an intermediate layer and training a multiclass linear SVM.

Implementation and Results

The models were implemented in the Python programming language, mainly using the PyTorch library and for the last part of the third task the scikit-learn library.

The functions seen during the

They were run on Trieste Area Science Park's cluster ORFEO, using the GPU nodes that mount 2 Intel Xeon Gold 6226 12-Core CPUs (256GB of RAM) and 2 NVIDIA V100 GPUs; the main computations were performed by the GPU.

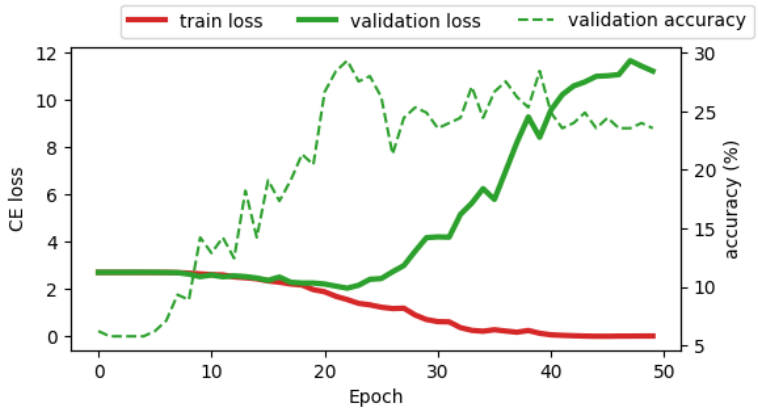
Task 1

The first model was implemented exactly as requested, but it was still necessary to tune the learning rate and the momentum. A manual search was employed, by trying different learning rates in the range $[0.00001, 0.9]$, and once the best learning rate was found the momentum was found in the same way in the range $[0.05, 0.9]$. The best learning rate was found to be 0.0004 and the best momentum 0.9

The optimum was found at epoch 23, leading to an overall test accuracy of 29.3%; the training took 173.5 seconds. This accuracy, though not high, is still significantly higher than the one of a random/dummy classifier (which would score around 6.7%), so this simple network is already learning how to classify the images.

We can see that after epoch 15 the model overfits on the training set.

From the confusion matrix we can see that the model performs better on the images of highways and open country.



Task 2

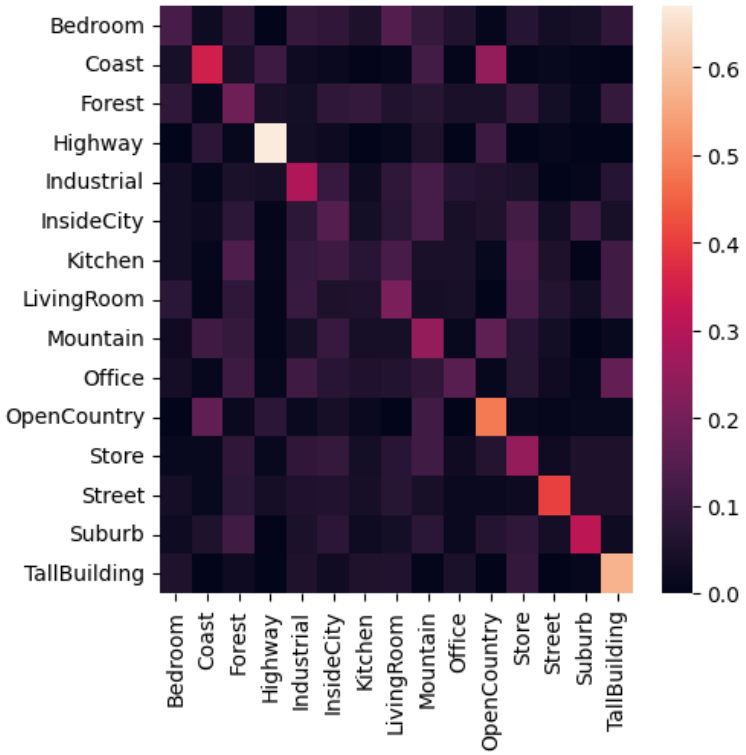
Data Augmentation

Given the fact the categories of the dataset are invariant to horizontal reflection (eg. a bedroom seen through a mirror is still a bedroom), I transformed all the images of the training part of the training set with an horizontal reflection, and added them to the original training set. The test set and validation set were not touched. In this way, I was able to double the number of images on which the model is trained on, having a training part of the training set of 2550 images. By using the same learning rate and momentum used for the first part, I was able to reach an overall test accuracy of 34.1% in 176 seconds

Improved Network

I tried to improve the new neural network by:

- Adding batch normalization layers [2] before the reLU layers: These layers normalize the input of RELU layers by rescaling them and allowing a higher learning rate and still avoiding the divergence of the optimizer.
- Increasing the support of convolutional layers while moving from input to output: The first Convolutional layer stays 3×3, the second becomes 5×5 and the third becomes 7×7.
- Dropout Layers: in order to reduce overfitting and approximate the use of an ensamble of network, for each minibatch before the fully



connected layer at the end i randomly dropped half of the input.

I was able to get a major improvement, with a model that, employing roughly the same time to train (177 seconds), reached an overall test accuracy of 54.7%

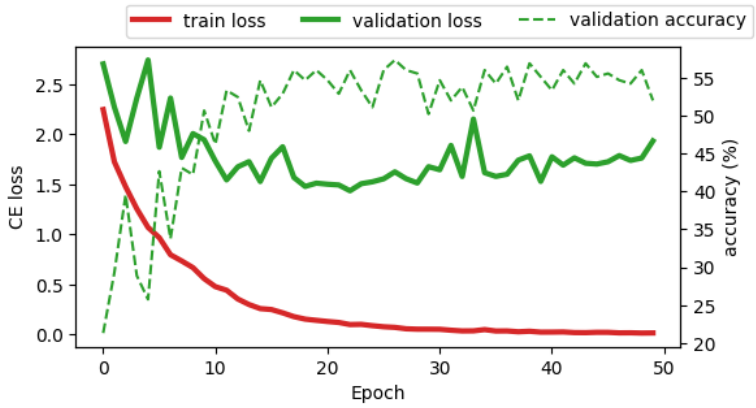
Ensamble of Networks

The i used an ensamble of networks: I trained 5 classifiers of the type described above, which given the random parts of the network gave different results and considered as predicted score for each class the mean of the scores given by the 5 classifiers.

I was able to reach an overall test accuracy of 62.6% in 198 seconds.

The optimum is reached at Epoch 22

We can see that the model performs really well on tall buildings and streets, and in general on pictures taken outdoors, while it has worst performances on indoors pictures, especially it struggles to distinguish between kitchens and living rooms



Task 3

For the last part, I used the pre-trained network AlexNet [3] in order to perform transfer learning. This is a widely known network, which in 2012 achieved a top-5 error of 15.3% on the ImageNet database, 10.8 percentage points lower than that of the runner up [1].

In order to correctly feed this model, the images had to be rescaled to 224x224, and to be represented in the range [0,1]. The initialization of the parameters is still gaussian with zero mean and 0.01 standard deviation for the weights and null biases.

Two approaches were implemented:

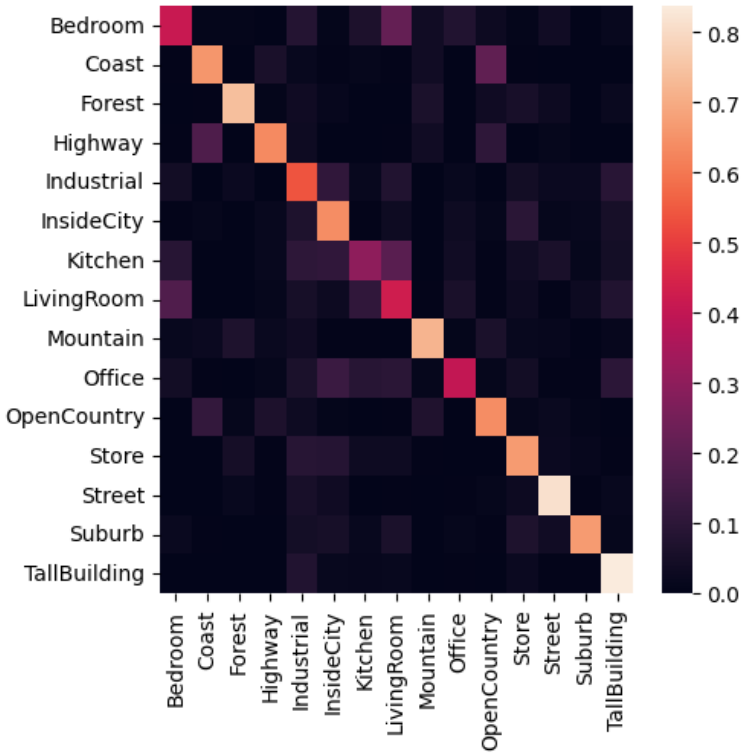
First approach

The first approach is to freeze the weights of all layers except for the last one. In this way the main features were extracted by the network that was already trained on over 15 million images, and then the last fully connected layer was tuned to recognize the images of our dataset.

Second approach

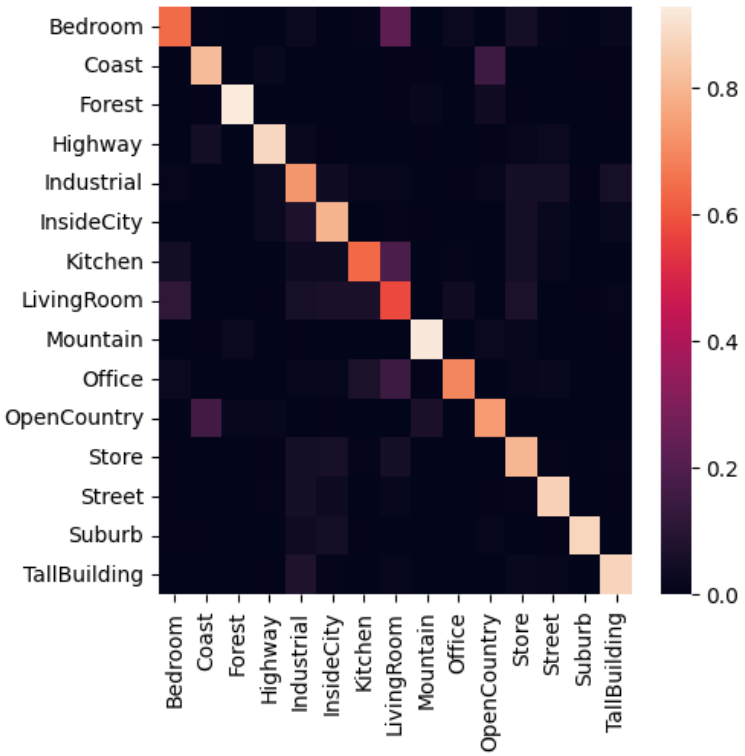
The second approach is to use the network, except for the last layer, to extract from the images the features that will be used to train a multiclass Support Vector Machine.

In order to do this I made AlexNet make an inference on the whole Dataset: in this way I transformed each image in an array of 1000 features extracted by AlexNet. At this point I trained and tested on this data a



multi-class SVM, implementing the one-versus-all approach.

The overall test accuracy was 79.6%, so a slight decrease with respect to the previous model, but that came with a great speed-up: extracting the training features and training the SVM took only 15.6 seconds, less than what was needed to train my first model wich scored only 29.3%. Of course this was expected, since using a pre-trained model means that most of the time needed for training was actually consumed in creating the pre-trained model.


Bedroom
Coast
Forest
Highway
Industrial
InsideCity
Kitchen
LivingRoom
Mountain
Office
OpenCountry
Store
Street
Suburb
TallBuilding
0.8
0.6
0.4
0.2
0.0

Bibliography

[1] Large scale visual recognition challenge 2012. *https://image-net.org/challenges/LSVRC/2012/results.html*, 2012.

[2] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[4] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, volume 2, pages 2169–2178. IEEE, 2006.

[5] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.