Computer Vision and Pattern Recognition Project: CNN-based Image Classifier

Lorenzo Diaz Avalos Martin and Matteo Alessandro Fumis

Fall 2024 Course

Contents

1 Introduction

This report presents the realization of an image classifier based on Convolutional Neural Networks (CNNs). The realization started from a simple baseline implementation, then improved by using different techniques, ranging from data augmentation to transfer learning.

2 Problem Statement

The aim of the project was to train a CNN able to solve an image multi-class classification problem with good performance. The training was made on a given dataset (from [Lazebnik et al., 2006] [5]) containing images of 15 different categories of objects (office, kitchen, living room, bedroom, store, industrial, tall building, inside city, street, highway, coast, open country, mountain, forest, suburb). The given dataset is already split in training set (appr. 1500 examples) and test set (appr. 3000 examples).

3 Assessment

The performance index chosen for assessment was the accuracy achieved on the test set by the best model (i.e. with best parameters) obtained with each approach. To help the evaluation of the models were plotted confusion matrices, in order have a graphical and more intuitive picture of the model performance.

4 Environment and Libraries

The programming language chosen to consduct the project was Python. Google Colab was used as shared working environment between team memebers. The following key Python libraries were used:

- *PyTorch*: for building and training networks
- *TorchVision*: for image transformations and pre-trained models
- *ScikitLearn*: for machine learning tasks such as plotting and SVM implementation

Web resourses and ChatGPT were used in some parts as helpers for writing Python code.

5 Implementations

Next are presented the approaches followed to train each model. The results are discussed in Section 6.

5.1 Baseline

The first, more simple implementation of the network (referred to as CNN_0) is based on the scheme represented in Table 1

#	**Type**	**Size**
1	Image Input	$64 \times 64 \times 1$ images
2	Convolution	$8 \times 3 \times 3$ convolutions with stride 1
3	ReLU	
4	Max Pooling	2×2 max pooling with stride 2
5	Convolution	$16 \times 3 \times 3$ convolutions with stride 1
6	ReLU	
7	Max Pooling	2×2 max pooling with stride 2
8	Convolution	$32 \times 3 \times 3$ convolutions with stride 1
9	ReLU	
10	Fully Connected	15
11	Softmax	softmax
12	Classification Output	crossentropyex

Table 1: The given CNN_0 structure

Both training and test set were properly rescaled to the required input size (64×64), along with other processing such as grayscaling and conversion to tensor. Then the train set was split into actual train set and validation set, respectively 85% and 15% of the original dataset.
Initial bias values were set to 0 and the initial weights drawn from a Gaussian distribution having 0 mean and a standard deviation of 0.01. The optimization *Stochastic gradient descent with momentum* algorithm was employed with minibatches of size 32. The best combination of learning rate and momentum were chosen with hyperparameter tuning, since no baseline values were suggested; the first parameter value was taken from the set of values {0.0001, 0.001, 0.01}, the second from {0.1, 0.5, 0.9}.

The training was set to occur in at most 30 epochs (`num_epochs=30`) and loss and accuracy were tracked for both training and validation; *cross-enthropy* was employed as loss function. The early stopping was triggered if the validation loss failed to improve for 5 consecutive epochs (`patience=5`) respect to the minimum.

5.2 Initial improvements

The first improvement was made respect to the given data: the training set was augmented by performing random horizontal flip (each image flipped with probability `p=0.75`),

random rotation (each image rotated by an angle between -10° and 10°) and random resized crop (crop area in the range [0.8, 1.0] of the original, then back rescaled to 64×64). The processing was made after the train/validation split (kept with the same ratio as in CNN_0), so only on the actual training set. Validation set and test set were not processed.
The following techniques were employed to change the network structure and obtain a better performing model:

- Batch Normalization: added a normalization layer right before each ReLU to stabilize and accelerate training ([Ioffe and Szegedy, 2015] [2])
- Filter number and size increase: more convolutional layers were added, increasing their size from input to output (3×3, then 5×5 and finally 7×7, while in CNN_0 were all of size 3×3)
- Dropout layers: added to improve regularization; the dropout probability was set to 0.3

The structure of the new neural network is shown in Table 2:

#	**Type**	**Size**
1	Image Input	$64 \times 64 \times 1$ images
2	Convolution	$8 \times 3 \times 3$ convolutions with stride 1
3	Batch Normalization	
4	ReLU	
5	Max Pooling	2×2 max pooling with stride 2
6	Convolution	$16 \times 5 \times 5$ convolutions with stride 1
7	Batch Normalization	
8	ReLU	
9	Max Pooling	2×2 max pooling with stride 2
10	Convolution	$32 \times 7 \times 7$ convolutions with stride 1
11	Batch Normalization	
12	ReLU	
13	Max Pooling	2×2 max pooling with stride 2
14	Convolution	$64 \times 7 \times 7$ convolutions with stride 1
15	Batch Normalization	
16	ReLU	
17	Max Pooling	2×2 max pooling with stride 2
18	Fully Connected	15
19	Softmax	softmax
20	Classification Output	crossentropyex

Table 2: CNN_1 structure

A new model was trained (CNN_1), this time keeping the best performing values of learning rate and momentum in CNN_0, while tuning just the weight decay, taken from the set of values {0.0, 0.00001, 0.0001, 0.0005}, and the optimizer, which was *adam* or SDG. Finally, using the better performing parameters in CNN_1, was trained an ensemble of 5 independent networks; the classification was based on the arithmetic average of

the raw scores of each network, as suggested by [Szegedy et al., 2015] [6].

5.3 Transfer Learning

In this phase was employed a pre-trained network, in particular *AlexNet* [Krizhevsky et al., 2012] [4]. Images of the datase were adjusted to meet specifications (3 channels, of size 224×224, normalization with `mean=[0.485, 0.456, 0.406]` and `std=[0.229, 0.224, 0.225]` (found in PyTorch documentation). Augmentation for training set was performed as for CNN_1.

The pre-trained network was used with two different approaches. The first one consisted in freezing the training of the network up to the final fully connected layer and performing the weights adjustments just for the final layer (AlexNet_0).
The second one aimed to use the network as feature extractor for then training an *SVM* classifier on the extracted features. The pre-trained network was kept up to the sixth fully connected layer. Two different versions of the SVM classifier were trained, one using *one-vs-one* (OvO) comparation (SVM_a), the other *Error Correcting Output Code* (ECOC) ([Dietterich and Bakiri, 1994, James and Hastie, 1998] [1] [3]) (SVM_b); for both versions were trained four different models, each one using a different kernel, in particular *linear*, *radial basis function* (rbf), *polynomial* and *sigmoid*. For the three non-linear kernels, hyperparameter tuning was performed on regularization constant, taking from the set {0.01, 0.1, 1, 10, 100}, and on the gamma parameter, taking from {0.0001, 0.001, 0.01, 0.1, 1, 0}, while on the linear kernel just on the regularization constant, using the set {0.01, 0.1, 1, 10, 100}. Due to computational constraints, the tuning was made just for SVM_a; the best performing values were employed for the corresponding kernel in SVM_b.

6 Results

The CNN_0 was the worse performing model, with the best accuracy on the test set of 26% achieved with a learning rate of 0.001 and momentum of 0.9. The results of training and testing with the best performing parameters are shown in Figure 1.

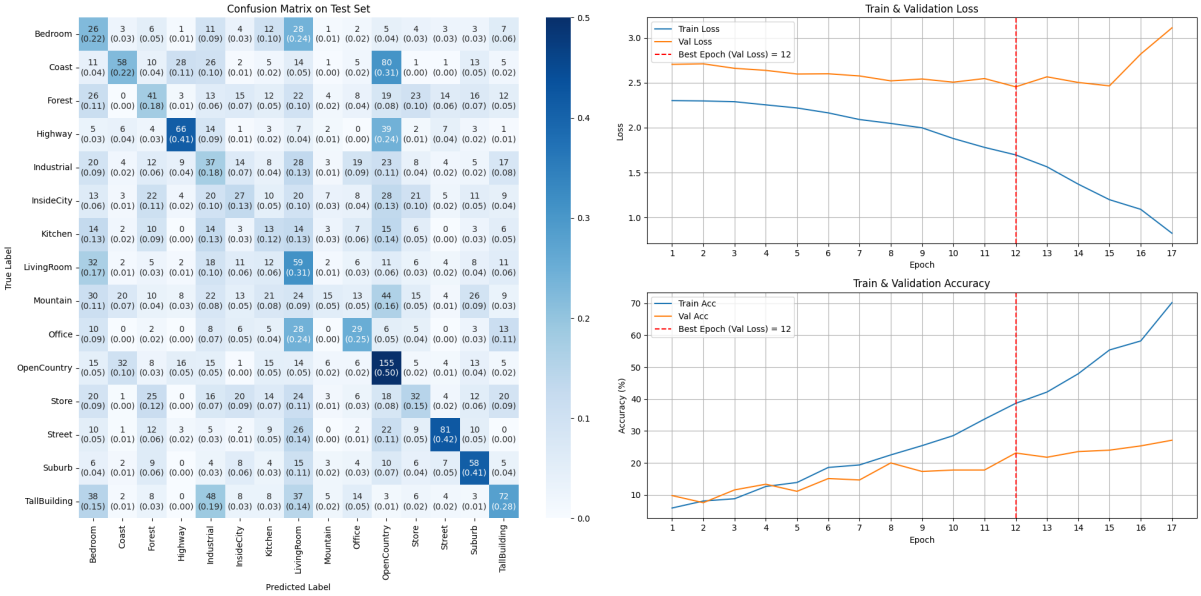


Figure 1: On the left, confusion matrix on the test set. On the upper right, loss values graph on training and validation, while on lower right the accuracy values graph on training and validation.

The graphs on the right show the curves of the loss and accuracy values for both training and validation through the epochs, highlighting the epoch at which was hit the validation loss minimum before the early stopping. From the confusion matrix are visible the misclassifications made by the model; some of them are particularly coarse, like confusing highways or coasts with countries and tall buildings with living rooms.

The improved implementation CNN_1 performed significantly better than the baseline, achieving an accuracy of 61%; for learning rate and momentum values were kept the best performing ones used in CNN_0, while tuning just weigth decay and the type of optimizer; the best combination was 0.00001 for weight decay and *adam* optimizer.

The ensemble of networks (EoN) achieved a slightly better performance than CNN_1, hitting 66% of accuracy. Figure 2 shows the loss function graph and confusion matrix for EoN:

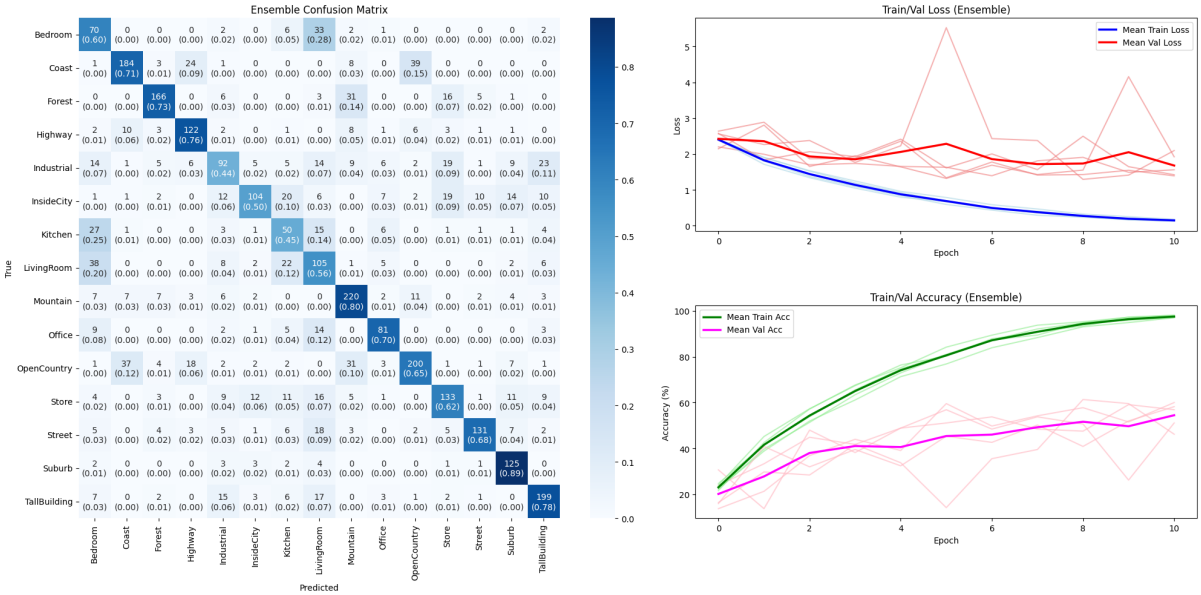


Figure 2: On the left, the confusion matrix of EoN on test set. On the upper right, loss function graph on training and validation of EoN, while on the lower right the accuracy function graph on training and validation of EoN.

The transparent lines in the graphs on the right corresponds to the curves associated to the different networks, which fluctuate through the epochs. The confusion matrix, compared to the one corresponding to CNN_0, highlights the improvement of the model, but some very different classes are still confused, such as industrial or urban buildings that are labeled as stores.

The use of the pre-trained network led to the best results. The model obtained from the fine-tuning of the last layer, AlexNet_0, achieved 87% accuracy using default parameters and *adam* optimizer. In the confusion matrix in Figure 3 are visible the few misclassifications made, which prevalently confused living rooms with bedrooms and coast with countryside.

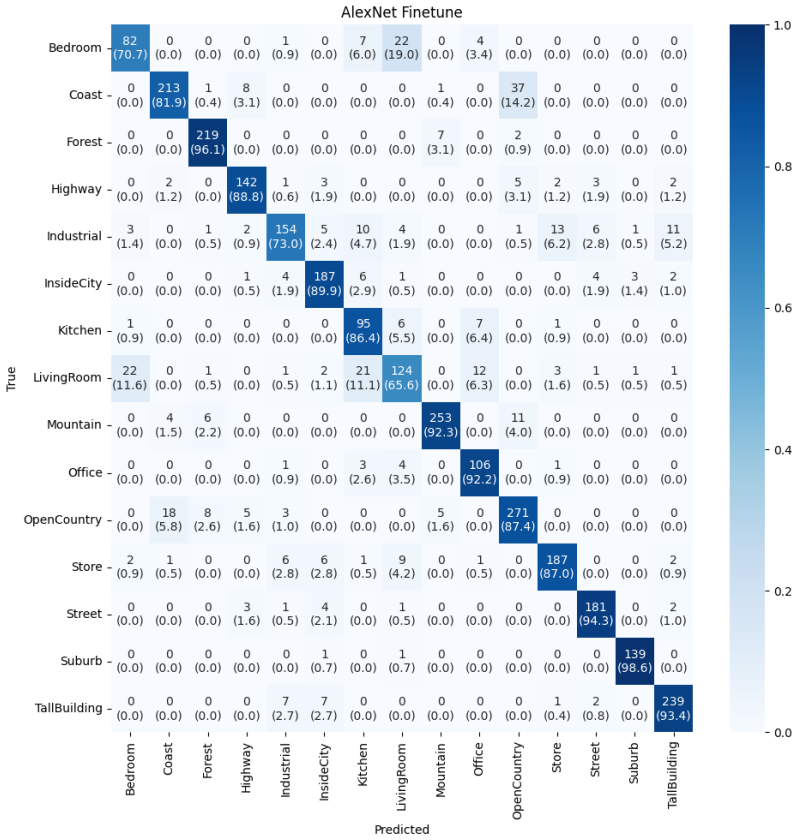


Figure 3: Confusion matrix of AlexNet_0 on test set

A summary of performances obtained with different SVM classifier implementations trained on features is shown in Table 3:

Version	**Kernel**	**Best C**	**Best γ**	**Accuracy (%)**
SVM_a (OvO)	Linear	0.01	-	86
	RBF	10	0.0001	84
	Polynomial	0.01	0.01	67
	Sigmoid	10	0.0001	87
SVM_b (ECOC)	Linear	0.01	-	84
	RBF	10	0.0001	83
	Polynomial	0.01	0.01	81
	Sigmoid	10	0.0001	85

Table 3: Accuracy of different SVM models with best hyperparameters

In both implementations, the best performing model was the one using the sigmoid kernel. The confusion matrix is shown Figure 4, which is similar to the one associated to AlexNet_0

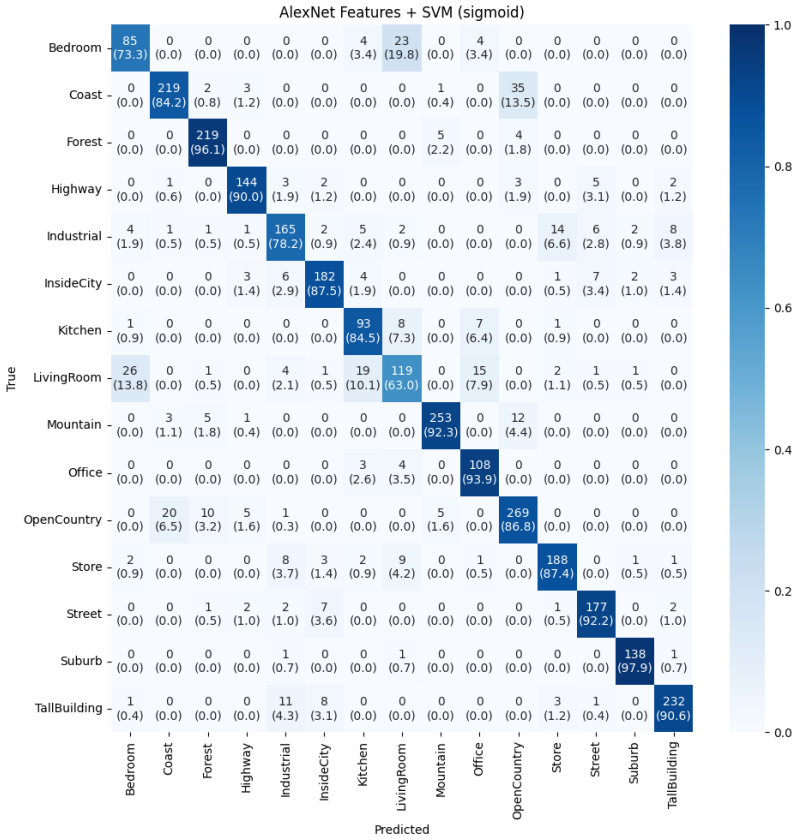
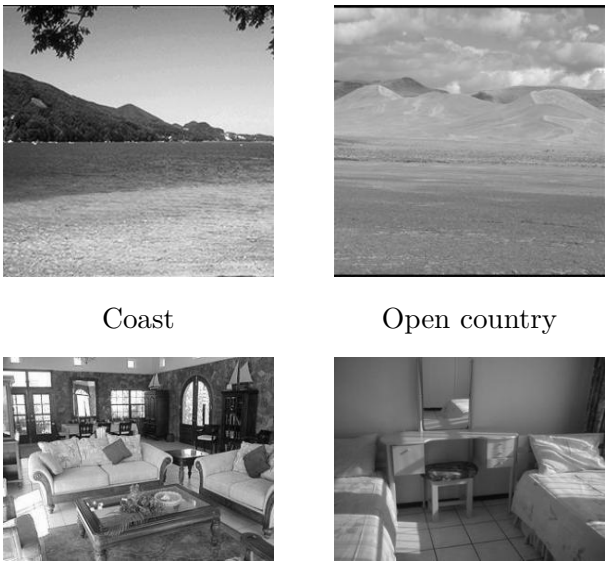


Figure 4: Confusion matrix of SVM_a using sigmoid kernel on test set

Figure 5 shows objects belonging to the four classes that the most led to errors:



Coast | Open country

Living room | Bedroom

Figure 5: Instances the most misclassified objects

From this small sample of images, we could say that errors made by the best classifiers are quite reasonable; at a first glance, also a human could confuse the wide surface of the

sea with a field in open country and, due to the similar furniture, also a bedroom with a living room.

7 Conclusions

The realization of this project showed the effectiveness of CNNs for solving image classification problems. Even small networks can led to decent results, but the application of transfer learning techniques produced models with high accuracy scores, able to correctly classify almost all categories from the dataset and failing where also a human could struggle for a moment.

References

[1] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research (JAIR)*, 2(1):263–286, January 1995.

[2] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 448–456. PMLR, 2015.

[3] G. James and T. Hastie. The error coding method and picts. *Journal of Computational and Graphical Statistics*, pages 377–387, 1998.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1097–1105, 2012.

[5] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2169–2178. IEEE, 2006.

[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, E. Dumitru, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9. IEEE, 2015.