Laboratory 11: CNN for Image Classification

Ramón Emiliani
Universidad de los Andes
201125694
rd.emiliani689@uniandes.edu.co

Alejandro Posada
Universidad de los Andes
201227104
a.posada10@uniandes.edu.co

Abstract

Many approaches have been proposed to classify texture. In particular, texton-based classification has seen important success. Recently, convolutional neural networks have obtained excellent results in different computer vision tasks. In this laboratory, we implemented a convolutional neural network to classify textures from a modified version of the Ponce's group texture database. We performed experiments using different batch sizes and eliminating convolutional layers. Our best result was obtained using batch size $= 12$ *and jitter (ACA* $= 93.63\%$*).*

1. Introduction

Texture classification has a wide variety of applications such as industrial surface analysis, biomedical diagnostics, ground classification, sattelital imagery segmentation, etc. Indeed, texture is an important cue to object identity and material properties [1]. Even though texture classification has seen important advances (for example, texton-based classification [5][6][4]), it remains an elusive task. The traditional texture classification framework is to transform the texture image into a feature vector using a bank of filters, followed by a nonlinearity and smoothing steps before classification [7].

Convolutional Networks (convnets) have demonstrated excellent performance at different tasks such as image classification and object detection since their introduction in the 1990's by LeCun *et al.* [3]. The key to their success is their ability to leverage large labelled datasets to learn increasingly complex representations of the input.

One of the principal problems of texture classification lies in how to extract representative texture features. Additionally, convnets have proven to be excellent at extracting features. Thus, performing texture classification using convnets seems like an interesting method. In this laboratory, we implemented a CNN architecture to perform texture classification from a random weight initialization (note that we do not present our best results, which were obtained with another architecture).

2. Materials and methods

2.1. Dataset

The dataset on which we trained our convnets is based on the Ponce's group texture database [2]. This database features 25 texture classes with 40 samples each. It is divided into 30 training images per class and 10 test images per class. The modified database we used consists of randomly sampled 128×128 patches from the train set image. It is thus composed by 25000 images divided into train (15000 images), validation (5000 images) and test (5000 unlabeled images). An example of the images present in the dataset is shown in figure 1.

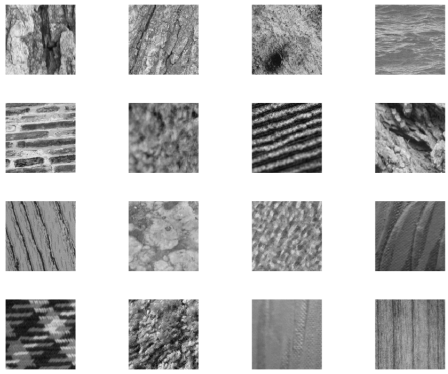


Figure 1: Examples of textures from the dataset.

3. Approach

Initially, we tested an architecture similar to LeNet-5. We decided to add more convolutional layers and modify the kernel sizes given that our dataset's images are almost 5 times bigger than the digit images used originally by LeCun [3] (128×128 vs. 28×28). Our architecture is composed of four convolutional layers (C1, C2, C3 and C4), each one of them followed by a ReLu activation layer. C2 and C3 are also followed by a maxpooling layer with kernel sizes 2×2

and 3×3 respectively. C4 is followed by a fully connected layer and a softmax layer is used to classify the input into one of the 25 texture categories. The kernel sizes of C1, C2, C3 and C4 are 5×5, 9×9, 3×3 and 9×9, and their strides are 1, 2, 1 and 1, respectively. A diagram of the architecture is shown in figure 2.

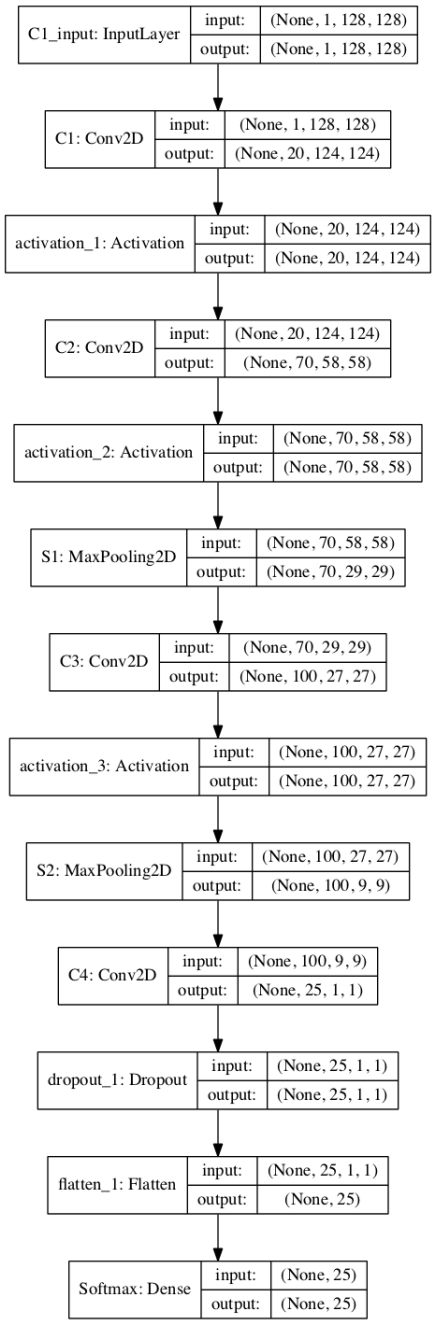


Figure 2: Diagram of our architecture. The size of the inputs and outputs is shown at the right. For example, C3 uses 100 kernels and produces activations maps of size 27×27.

Our network was trained from a random weight initialization with 50 epochs using stochastic gradient descent with learning rate $= 0.01$ and momentum $= 0.9$. We implemented our model in Keras using the Theano backend. We trained it using batch sizes of 8, 12, 40, 400 and 900 in order to determine the effect of the batch size in the classification accuracy. Once we found the optimal batch size, we trained the network using jitter: the train images were randomly shifted, rotated in a range of 90^{o} and horizontally and vertically flipped. Finally, we performed ablation tests by training the network using jitter and removing C3, C4, C3 and C4, C2 and C3, and C1 and C2.

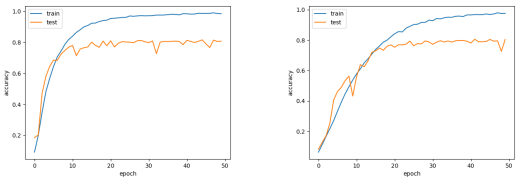
4. Results and discussion

Table 1 shows the relationship between the batch size used to train the network and the average classification accuracy (ACA) of the network evaluated on the validation set. ACA is significantly smaller for big batch sizes than for small batch sizes. The optimal batch size is near 12, so we performed the rest of experiments using this batch size.

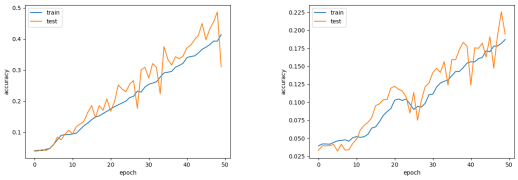
Table 1: Effect of batch size on classification accuracy.

Batch size	ACA (%)
8	81.05
12	81.54
40	80.65
400	48.72
900	22.62

Figures 3 and 4 show, respectively, the train/loss accuracy and loss plots for the different batch sizes. We omit the batch size $= 8$ since the accuracy and loss plots and practically identical to the ones obtained using batch size $= 12$. Figures 3 and 4 explain why ACA is low for big batch sizes. For small batch sizes, the model converges between epoch 10 and 25, while for big batch sizes the model has not converged at epoch 50. Thus, augmenting the number of epochs is not an effective way to improve the classification accuracy for small batch sizes.

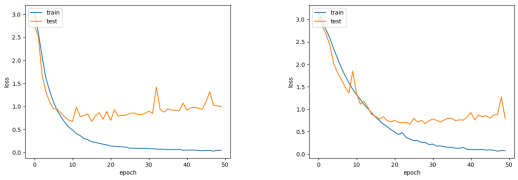


(a) Batch size = 12. (b) Batch size = 40.

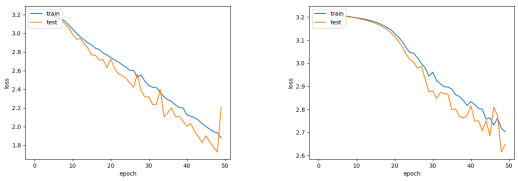


(c) Batch size = 400. (d) Batch size = 900.

Figure 3: Train and validation accuracies for different batch sizes.



(a) Batch size = 12. (b) Batch size = 40.



(c) Batch size = 400. (d) Batch size = 900.

Figure 4: Train and validation losses for different batch sizes.

Figure 5 shows the confusion matrix of the evaluation of the model trained with jitter. This model clearly outperforms the models trained without jitter since it obtained an ACA of 93.63%. Figure 6 shows the train and validation losses for different batch sizes for the case when the network was trained using jitter and batch size = 12. It can be seen that at epoch 50 the model has not converged an thus augmenting the number of epochs might improve the classification accuracy. This addition of synthetic training data allowed us to improve the robustness of the network and to reduce overfitting (note that in figure 6 both train and validation accuracies are still improving at epoch 50).

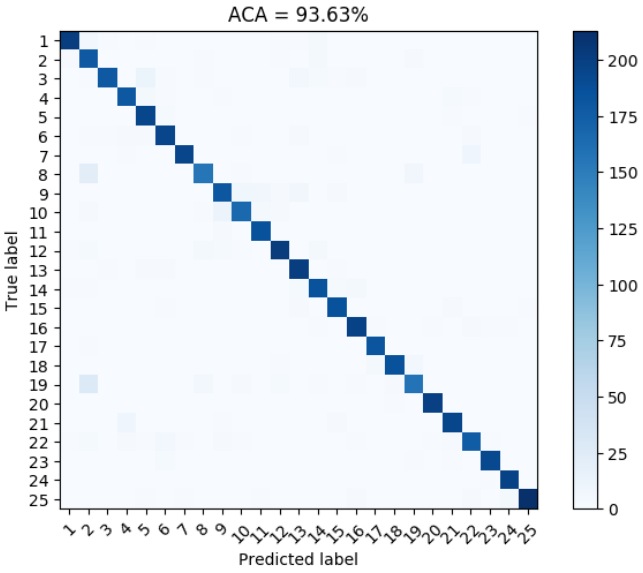
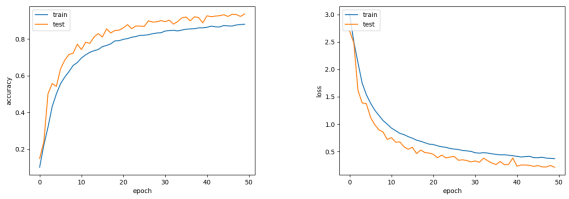


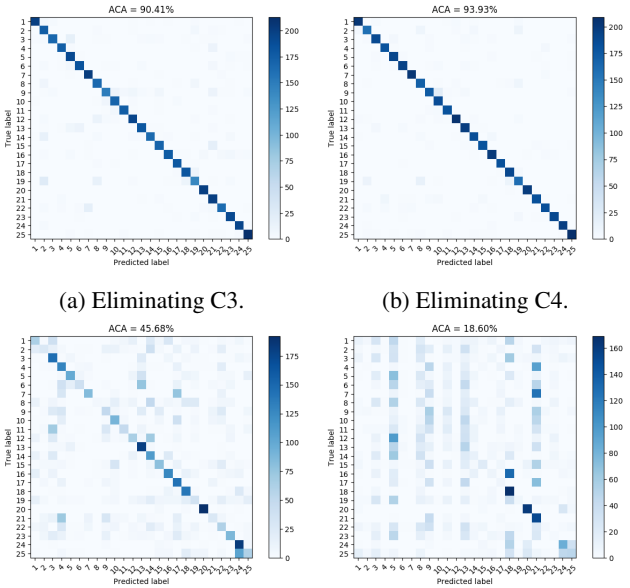
Figure 5: Confussion matrix of the network trained with jitter evaluated on the validation set.



(a) Train and val. accuracy. (b) Train and val. loss.

Figure 6: Train and validation losses for the network trained using jitter.

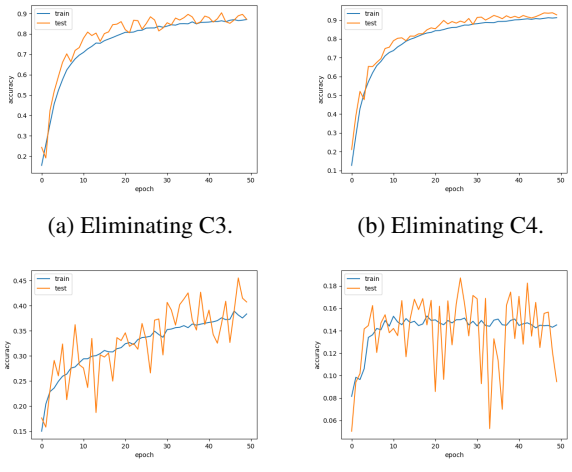
The ablation experiments results are shown in figures 7, 8 and 9. The confusion matrices show that the ACA is mostly affected by layers C1 and C2. However, when both C2 and C3 are removed, the model fails. This implies that C2 and C3 are the core of the model. In the same sense, in figures 8 and 9, it can be seen that eliminating the final convolutional layers (C3 and C4) does not have an important effect. On the other hand, elminating the initial layers (C1 and C2) affects the validation accuracy and loss. Thus, without layers C1 and C2 the model is able to partially learn the train set but cannot generalize to the unseen images of the validation set. In particular, note that eliminating both C2 and C3 greatly affects the validation results, and thus C2 is a critical layer for the network.



(a) Eliminating C3. (b) Eliminating C4.

(c) Eliminating C1 and C2. (d) Eliminating C2 and C3.

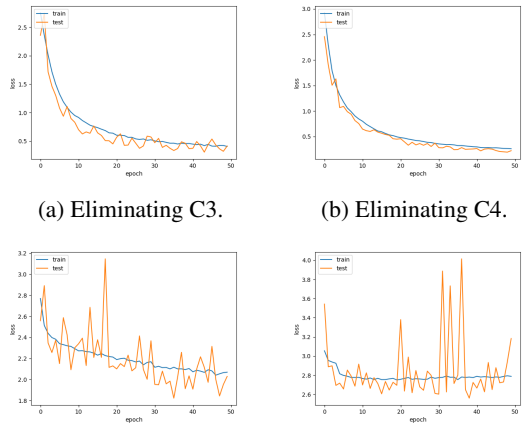
Figure 7: Confusion matrices for different ablation experiments.



(a) Eliminating C3. (b) Eliminating C4.

(c) Eliminating C1 and C2. (d) Eliminating C2 and C3.

Figure 8: Train and validation accuracies for different ablation experiments.



(a) Eliminating C3. (b) Eliminating C4.

(c) Eliminating C1 and C2. (d) Eliminating C2 and C3.

Figure 9: Train and validation losses for different ablation experiments.

5. Conclusions

Many neural network architectures used for image classification tasks are effective but massive. In this laboratory we developped a simple yet effective architecture to classify textures. Even though the use of jitter significantly improved the classification results, these could be improved by increasing the number of layers. Additionally, the architecture trained with jitter did not overfit the train set and thus augmenting the number of epochs might improve the classification results.

References

[1] D. Forsyth and J. Ponce. *Computer vision: a modern approach*. Upper Saddle River, NJ; London: Prentice Hall, 2011.

[2] S. Lazebnik, C. Schmid, and J. Ponce. A sparse texture representation using local affine regions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1265–1278, 2005.

[3] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[4] T. Leung and J. Malik. Recognizing surfaces using three-dimensional textons. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1010–1017. IEEE, 1999.

[5] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International journal of computer vision*, 43(1):29–44, 2001.

[6] J. Malik, S. Belongie, J. Shi, and T. Leung. Textons, contours and regions: Cue integration in image segmentation. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 918–925. IEEE, 1999.

[7] F. H. C. Tivive and A. Bouzerdoum. Texture classification using convolutional neural networks. In *TENCON 2006. 2006 IEEE Region 10 Conference*, pages 1–4. IEEE, 2006.