Convolutional neural network to classify images based on texture

Johana M. Ramirez Borda
Los Andes University
jm.ramirez11@uniandes.edu.co

Francisco A. Rozo Forero
Los Andes University
fa.rozo1843@uniandes.edu.co

Abstract

Image texture can be described as the pattern of variations in the gray levels of an image that provides information about smoothness, coarseness, bumpiness and also regularity. In literature there can be found several approaches used to analyze, classify or segmentate images based on this type of information (Texture). Some methods started using Minimum Distance Classifiers, others using Bag of Words, Fisher vectors and Deep learning. In this project, two convolutional neural network (CNN) were implemented (VGG-7 and VGG-16) and trained with 15.000 images [9], [4] in order to classify 2.500 validation images belonging to 25 categories. There were used convolution, max-pooling, rectified linear unit and softmax operators, as well as jitter transform and dropout technique. The results showed that VGG-7 had the highest ACA index corresponding to 76.96%. Additionally, it was found that when using the jitter transform and the dropout technique, the ACA index did not reach or exceed the previously mentioned.

1. Introduction

Texture analysis is a common task in computer vision and refers to the characterization of regions in an image by their texture content. In this sense, the texture is related to roughness, bumpiness, smoothness... etc, which quantitatively refers to variations in the gray levels of an image. This analysis is used in numerous applications such as medical image processing, automated inspection, remote sensing and texture segmentation. Even more, texture analysis is used when trying to identify objects in an image that are more characterized by their texture than by intensity [5].

One recent approach to texture analysis is based on the use of Deep Neural Networks, which have shown the ability to learn data representations in both supervised and unsupervised settings [1]. However, texture classification is not an easy task due to large intraclass variation and low interclass distinction. Studies have shown that although methods such as bag of words (BoW) had good results, Fisher vectors (FV), which is based on Gaussian mixture models is able to provide more discriminative power for images with low inter-class distinction [8]. Moreover, in literature there can be found many methodologies based on neural networks, for example, Leung and Peterson [6] used feed-forward neural networks with 1 to 2 hidden layers for the classification and segmentation of textred images. Also, Kulkarni and Byars [6] suggested an artificial NN model to extract frequency-domain features that can recognize textures.

Considering this, the objective of the present project is to design, train and validate two CNN (VGG-7 and VGG-16) from a random weight initialization. Moreover, parameters such as batch-size, number of epochs and size of output of convolutional layers were varied in order to find which NN gave the best results (ACA index). Finally,

2. Materials and methods

2.1. Database

The dataset used in this project was provided by [9] and contains randomly sampled 128x128 patches from each image found in the database of the Ponce group [4]. The provided .zip contains one folder for each test, train and validation sets. Within the test one, there are 2.500 images, all labeled as ”0”. The train folder contains 15.000 images divided into 25 categories, 600 images each, and all the images belonging to one category are within a sub-folder named with the corresponding category label. The validation set is similar to the train but it contains 2.500 images. Finally, there is also a .text file that contains the name corresponding to each label, for example, ”bark2 T03, wood1 T05”, etc.

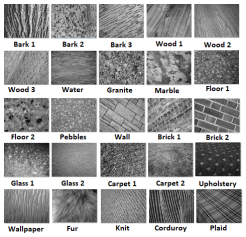

Figure 1: Example of one random image within each category of the database.

As it can be seen, there are categories such as bark, wood, water, floor, carpets,glass,wallpaper, etc .Also, each category within the database has images taken from different angles and distances.

2.2. CNN creation and training

When creating the neural network, and based on the practical retrieved from [3], we implemented four main building blocks:

- Convolution: implementation of a bank of filters to the input images preserving the resolution of the feature map:

$$y_{i'j'k'} = \sum_{ijk} w_{ijkk'} x_{i+i',j+j',k} \quad (1)$$

  Which operate on a tensor *x* with *K* channels and *K'* filters.

- Pooling: coalescence of nearby feature values into one. In the implemented neural network it was used max-pooling trough the max operator:

$$y_{ijk} = max(y_{i'j'k} : i \leq i' < i + p, j \leq j' < j + p) \quad (2)$$

- Rectified Linear Unit (ReLU): activation function that takes the positive part of it input argument. Is defined by:

$$y_{ijk} = max(0, x_{ijk}) \quad (3)$$

- Softmax: this function squashes each one of the outputs of the last layer to be between 0 and 1. Moreover, it also divides each output such that the total sum of the outputs is equal to 1. This is necessary since the output of this function is equivalent to a probability distribution and it gives information about the probability of an image to correspond to certain class. It is defined by:

$$y_{ijk'} = \frac{e^{x_{ijk'}}}{\sum_k e^{x_{ijk}}} \quad (4)$$

These blocks were used when implementing versions of both VGG-7 and VGG-16 neural networks. First, the batch-size was varied between 20 and 200 when using 20 epochs. Then, the number of epochs was increased to 100 in order to analyze the performance of the classifier (while measuring the ACA index) through the epochs.
As layers inside neural networks are represented as matrices in $\mathbb{R}^n$, the way they connect is according to their dimensions. This means, that the outputs of each layer should coincide with the dimensions of the inputs from the next layer. For this reason, using the network that had the best performance, which was VGG7, the size of the output of the convolutional layer 4 and input of the convolutional layer 5 was varied between 218, 256, 384 and 512. Each layer is separated by *weights*, which are the ones that operate with the inputs to obtain the outputs. These operations can be linear and non-linear such as dot product and sigmoid function respectively.

After varying the parameters mentioned above, the jitter transform and the dropout technique were implemented in order to determine whether the use of these techniques improved the ACA.

3. Results and discussion

As mentioned above, the batch-size was first varied. The ACA index obtained when varying it though 20 epochs in VGG-7 network is shown in Figure 2 and 3.

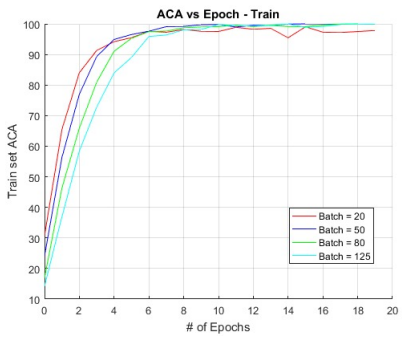

Figure 2: ACA values for train set while varying batch size.

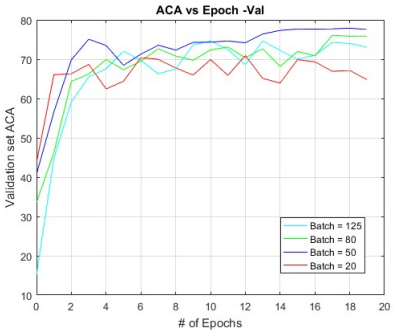

Figure 3: ACA values for validation set while varying batch size.

As it can be seen, the best results were obtained for a batch size of 50. The ACA for the train set rose to 98% after 7 epochs and reached 100% since epoch 15. Likewise happen to the validation set, as the ACA exceeded 73% after 7 epochs. In this sense, after the variation of the batch size in VGG-7 the maximum value was 76.88%.

Then, using the batch size that gave the highest ACA index, the VGG-16 network was trained and validated varying the number of epochs until 20. The resulting ACA indexes are shown and compared with the corresponding values obtained before in VGG-7 (see Figures 4 and 5).

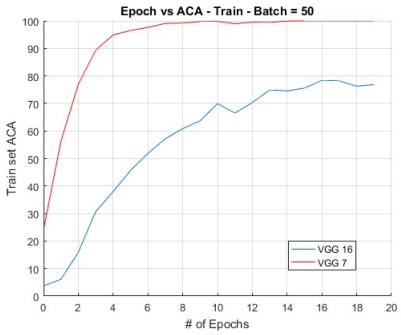

Figure 4: ACA value obtained in train set of VGG 7 and VGG 16 with batch-size of 50.

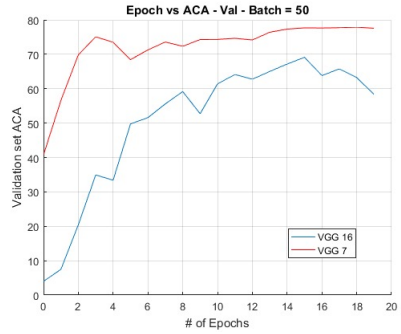

Figure 5: ACA value obtained in validation set of VGG 7 and VGG 16 with batch-size of 50.

Observing Figures 4 and 5, it can be seen that VGG-7 had better results through all 20 epochs. Later, these two networks were tested when increasing the number of epochs in order to see if their performance improved or decreased. The results for some epochs are shown in table 1 and 2, staying VGG-7 as a better classifier for this database.

Table 1: ACAs of VGG-7 while varying the # of epochs

VGG-7			
Batch size	**Epoch**	**ACA Train**	**ACA test**
50	25	100	76,96
	30	100	76,96
	35	100	76,84
	40	100	76,8
	45	100	76,68
	50	100	76,68
	55	100	76,68
	60	100	76,68

As it can be seen in Table 1, although the ACA was very similar since epoch 20 (ACA = 76.88%), it reached a maximum value of 76.96 in epoch 25 and then it stabilized at 76.68 since epoch 45, in VGG-7.

Table 2: ACAs of VGG-16 while varying the # of epochs

VGG-16			
Batch size	**Epoch**	**ACA Train**	**ACA test**
50	25	76,83	58,28
	30	75,3	57,88
	35	70,45	62,28
	40	64,95	48,92
	45	58,39	59,08
	50	68,07	45,92
	55	59,27	49,5
	60	64,15	61,6
	65	4	4

Table 2 shows that the ACA in VGG-16, in contrast to VGG-7, did not increase nor stabilize but decreased drastically from the 60th epoch where it decreased from 61.6 to 4%. From this moment the ACA remained at 4% and the result of the loss between the output and the target remained at NaN. According to the literature there are several reasons why a model can diverge. In our case this could be due to the fact that the Loss function that was used calculates the logarithm of the prediction, which diverges when said prediction approaches zero [7].

Once observed that the VGG-7 network had the best results, the size of the output of the convolutional layer 4 was varied, as well as the input size of the convolutional layer 5. These were varied between 128, 256, 384 and 512 (previous results). The obtained ACA during the first 20 epochs is shown in Figure 6 and 7.

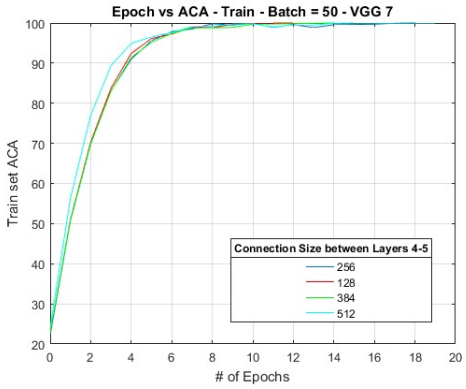

Figure 6: ACA value obtained in train set when varying the output size of convolutional layer 4.

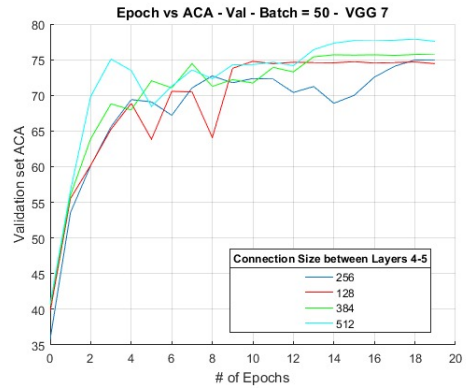

Figure 7: ACA value obtained in validation set when varying the output size of convolutional layer 4

As illustrated in Figure 6, by varying the sizes of the convolutional layers an improvement was obtained in the training and validation ACA. In this sense, until epoch 20, with an output size of layer 4 equal to 128, a maximum ACA of 74.76% was obtained, with a size of 256, 74.96% was obtained, with a size of 384, 75.8% was obtained and finally, with a size of 512 it was obtained 76.88%.

On the other hand, as it may happen that some images in the database might not be ”realistic”, some process should be done to reduce overfitting and make the model wider. In order to solve this, once found that VGG-7 had the best results, and once changed the mentioned parameters, then the jittering transform was randomly applied to some images of the database and then trained with this NN. The results are shown below in Figure 8.

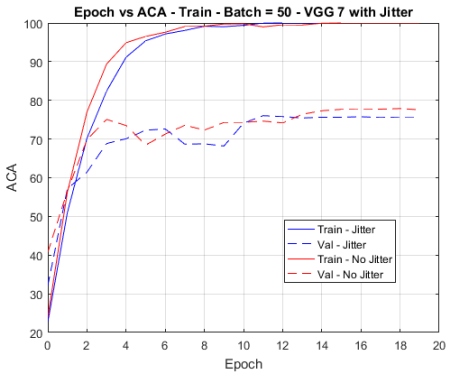

Figure 8: ACA obtained with and without Jitter transform.

Jittering randomly modifies data to random images in the database to simulate noise. Also, it randomly shifts some images to avoid uniformity. In consequence, jittering should improve the results of the classifier than if it is not used. However, looking at the dotted line of Figure 8 (which corresponds to the validation set), it was found that jittering didn't make a big difference in the results, as that classifier stayed below the one trained without jittering. We think that this method could work better when detecting specific object, as it adds profundity, changes brightness and spatial information. Yet, when the database is more realistic, jittering can become redundant and be an unnecessary use of training time and processing, just like in this texture database. Hence, jittering was not considered for the final classifier.

Finally, dropout was implemented in order to regularize the neural network and in order to test if it reduces overfitting. First, just one layer of dropout was applied as layer number 5 and then a second dropout was added as layer number 8. The results are shown in figure 9.

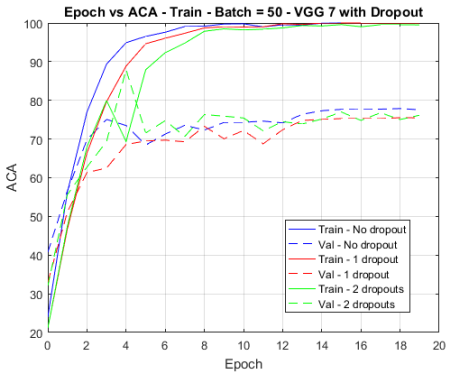

Figure 9: ACA obtained with and without dropout.

As seen in figure 9, not using dropout kept having the best results. Two dropouts were more consistent than just

one and had a slightly better performance: when using two dropouts the highest ACA obtained was 76.8% and when using just one, the highest ACA was 75.5%. Yet, neither of them were enough to reach the uniformly behavior of the VGG-7 without jittering and dropouts.
Searching in related literature, it was found that when the NN is small relative to the dataset, regularization is usually unnecessary [2]. This may be our case since, as the capacity of the model is not high, lowering it further by adding regularization will decrease performance [2]. In consequence, this was the final chosen classifier.

3.1. Conclusions

One version of VGG-7 and VGG-16 NN was implemented in each case. Moreover, parameters like batch-size, number of epochs, size of output and input of convolutional layers were varied and techniques like jittering and dropout were tested. Through this process, it was found that VGG-7 without jittering nor dropout, using a batch size of 50 was the best model since the resulted ACA was 76.96%.
In addition, it was observed that when modeling and training neural networks there is no big limitation. Infinity models can be made by modifying number of neurons, layers, activation functions, loss funcinput dations, ta, output data etc. Then, the wide options to build a classifier make it a limitation, as many models can be trained and there will always be a better combination of parameters to obtain a better performance. In consequence, hardware, time and creativity are the main limits in neural networks.

References

[1] S. Basu, S. Mukhopadhyay, M. Karki, R. DiBiano, S. Ganguly, R. Nemani, and S. Gayaka. Deep neural networks for texture classificationa theoretical analysis. *Neural Networks*, 97:173–182, 2018.

[2] S. Exchange. Dropout makes performance worst, aug 2017.

[3] O. V. G. Group. Vgg convolutional neural networks practical, 2017.

[4] 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.

[5] MathWorks. Texture analysis, 2017.

[6] A. K. Muhamad and F. Deravi. Neural networks for the classification of image texture. *Engineering Applications of Artificial Intelligence*, 7(4):381–393, 1994.

[7] S. Overflow. Deep-learning nan loss reasons, nov 2016.

[8] Y. Song, Q. Li, D. Feng, J. J. Zou, and W. Cai. Texture image classification with discriminative neural networks. *Computational Visual Media*, 2(4):367–377, 2016.

[9] A. F. R. Vergara. Convolutional neural networks for texture image classification, apr 2018.