Classification over the Ponce texture dataset using Convolutional neuronal Networks

Stephannie Jimenez Gacha
Universidad de los Andes
Cra 1 Este No 19A - 40 Bogotá, Colombia
s.jimenez16@uniandes.edu.co

Sergio Galindo León
Universidad de los Andes
Cra 1 Este No 19A - 40 Bogotá, Colombia
sa.galindo10@uniandes.edu.co

Abstract

Convolutional neuronal networks(CNN) correspond to state of art systems capable to represent and classify image candidates based on a their filter responses. This elements have a great performance in all types of computer vision tasks, sometimes, better than human performance. One of the most difficult classification tasks with multiple applications correspond to texture classification, for that, a VGG like CNN with 11 layers was implemented using 10 convolutional layers and one linear layer with a log-softmax classifier. The CNN was trained for 50 epochs using augmented data (Jitter, horizontal flip and random crop) from the Ponce texture dataset, a negative log-likelyhood loss function and tested online in a given test set. The best ACA obtained corresponds to 89.2% with a maximal F measure of 0.893.

1. Introduction

Convolutional neuronal networks (CNN) correspond to one of the state of art methods in all computer vision task, including classification. This elements take a given input and calculate a response based on some predetermined weights that, using error back-propagation, can be tuned to get a specific desired response. Specifically, a CNN calculates the convolution between the inputs and a kernel, learning the most discriminative filters for the task and performing the representation as well as the classification phases simultaneously. Inside a CNN, images are represented by their filter responses (and filter responses' responses) in an n dimensional space and, after several convolutions, the representation become discriminative enough to activate mostly one single neuron in the terminal layer, which acts as a classifier[2].

One of the main advantages of CNN's (and neuronal networks) is their ability to learn from the train examples, which is achieved by the parameter tuning when a specific input with certain labels is given. These systems posses a tremendous amount of parameters that allows an adequate learning of the images characteristics and labels and specially, an extraordinary performance on many computer vision problems such as digit recognition, animal classification and even in hard problems where humans tend to develop poorly.

Texture classification is one these difficult task for humans. Textures are composed of characteristic and repeatable local patterns made up of simple orientations and blobs, which can be very similar between classes and easily confounded. In addition, color information tends to be little discriminative in this case as texture information is usually independent of color or objects. For that, visual pattern information is fundamental in texture classification[1].

Classical image processing methodologies do not perform accurately in texture classification as local pattern information is not captured appropriately, however, CNN's can take advantage of the convolution operation to obtain this information, represent the images with discriminative filter responses and classify accurately different textures.

As texture is an important feature of objects which can be used to classify materials, detect instances, identify particular elements, segment objects and its classification corresponds to hard task for humans, we implemented a convolutional neuronal network with 10 convolutional and one fully connected layer to classify textures and it's performance was evaluated in a given dataset consisting of 2500 augmented data from the ponce dataset.

2. Materials and Methods

To train, validate and test the implemented CNN, a larger texture dataset was constructed from the Ponce group texture dataset by cropping random patches from the original images. Respecting the convolutional neuronal network it was built using pytorch and a VGG architecture.

2.1. Dataset

The dataset used was constructed from the original Ponce texture dataset which contains 40 images per class. For each texture category, twenty 128x128 random crops were taken from each image obtaining a total of 800 images per label and a total of 20000 images. The images were split in 15000 images for train, 2500 for validation and 2500 for test with a respective distribution of 75%, 12.5% and 12.5%. All images have an equal height and width of 128 and are found in .jpg format, also, the test labels were not provided.

2.2. CNN architecture

The CNN was constructed based on a VGG 13 architecture provided in the torchvision repository at GitHub, specifically, the VGG13D architecture. The net contains ten convolutional layers and a final log-softmax classifier consisting of fully connected layer with 25 neurons, one per texture. The convolutional layers are divided into 4, starting from an RGB standard image with 3 channels the first two layers return an output of 64 channels, the next two, 128, the following three 256 and the remaining three an output of 512 channels. After each dimension change a maxpooling operation with a window size of 2 was performed, also, each layer had a self normalizing activation functions (selu) and the kernel size for the convolutional layer remain constant with a value of 3.

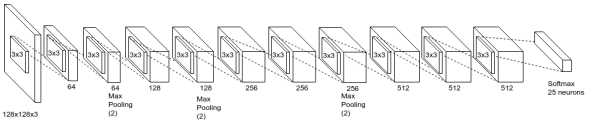

Figure 1. Arquitecture of convolutional neural network

To evaluate the effect of deepness in the performance two CNN were implemented, one with the complete 10+1 layers and another with the first 4 convolutional layers of the first and a fully connected layer. Both nets were trained in similar conditions. The architecture is shown

2.3. Training with augmented data

The datasets.ImageFolder function was used to load the dataset while changing the folder names to *label_xx* (were xx corresponds the class given by a two digit number) in order to load the labels as expected for evaluation purposes. Then, to train the CNN's with data augmentation the torchvision.transforms function was used. A color jitter, random horizontal flip and random crop was performed with with a probability of 0.5 for each transformation. The torch.utils.data.Dataloader function was used to generate and load the train batches with a batch size of 50 images(to respect the 4Gb GPU limit). Also, en experiment without data augmentation was performed.

The 11 layer net was trained for 50 epochs while the 5 layer net was trained for 20 epochs. For both nets the negative log-likelihood loss function was used due to the log-softmax classifier and a the stochastic gradient descent was used as optimizer with a learning rate of $2 \times 10^{-}3$ and a momentum of 0.9. For each epoch the model and validation accuracy and loss was obtained. A patience of 2 epochs was used preventing the model accuracy to decay after several epochs.

2.4. Evaluation metrics

The evaluation of the Net was performed on the course server by uploading a .txt file containing the image number and prediction. The predictions were ordered by average classification accuracy, and the precision, recall and maximal F-measure was returned too. As the task posses balanced classes the ACA provides an appropriate evaluation metric for the performance of the net.

3. Results

A great diversity of results where obtained by the training, validation and test of the final architecture. During the following section the results for each phase for the 11 layer network will be exposed [1].

3.1. Training and validation

In the training phase of the neural network it was necessary to monitor five different measures, loss per iteration, loss per epoch for the training set, accuracy per epoch for the training set, loss per epoch for the validation set, and the accuracy per epoch for the validation set. In this way, five different graphs could be obtained from each experiment. Figures 2, 3, 4, 5 and 6 show respectively the results for the 11 layer neural network with 50 epochs.

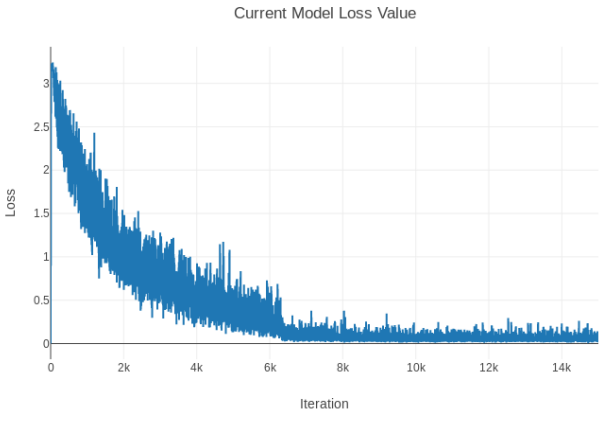

Figure 2. Loss value per iteration for the model

[1]The results for the neural network of 5 layers with 20 epochs of the training and validation phase are shown in the appendix A.

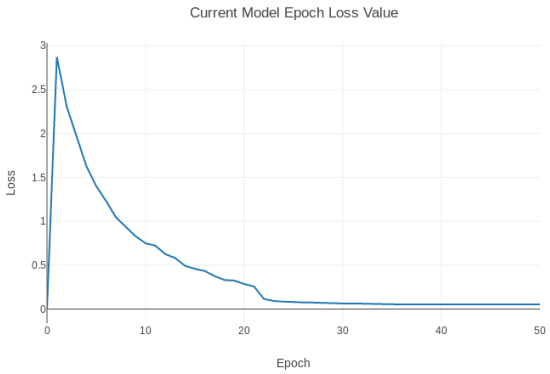

Figure 3. Loss value per epoch for the model in the training dataset

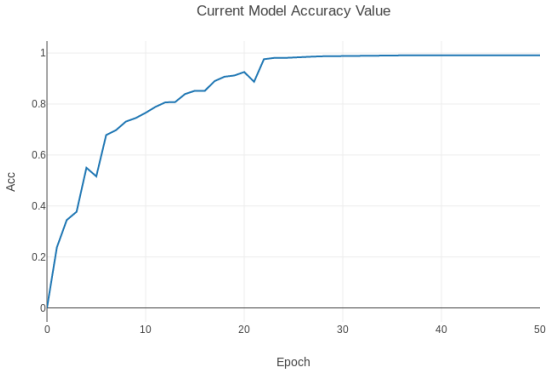

Figure 4. Accuracy value per epoch for the model in the training dataset

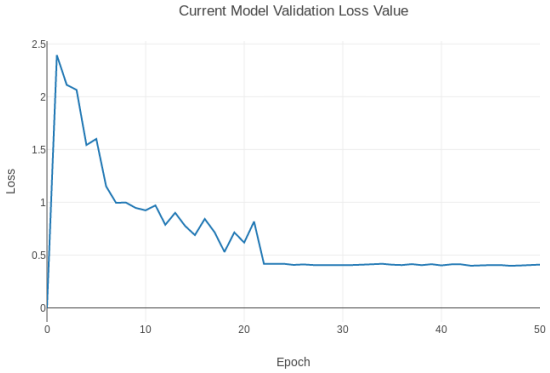

Figure 5. Loss value per epoch for the model in the validation dataset

The curves show the tendency of the neural network, where the learning rate didn't change throughout the whole training and validation phase. The results that we were obtaining were improving with every epoch for the train and validation dataset. It is important to mention that the results were optimal because the the overall loss of the model was decreasing. This means that the loss was decreasing for both datasets. More importantly, the accuracy of the validation and the training dataset was reaching to 1. This is important

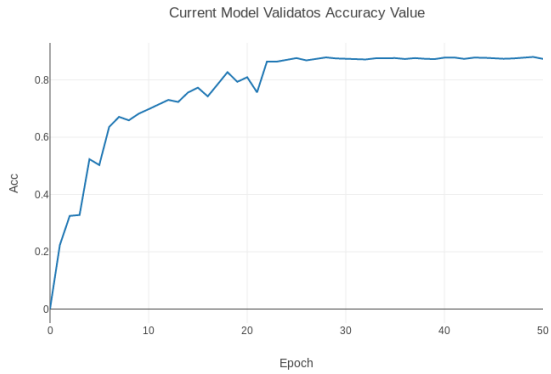

Figure 6. Accuracy value per epoch for the model in the validation dataset

because the predictions for the model were being correct as possible for all the 25 categories. In this way, when the model is submitted to the test dataset the expected value is higher for the correct classification.

It is important to mention that additional to the graphs, the best weights for the model were stored for the whole training session. And these weights were the ones used for the prediction of the neural network in the test, the following phase.

3.2. Test

As mentioned before, the test was made using the model to predict the labels for the test division dataset. In order to measure the accuracy, precision, recall and F-measure the course server was used. The results for these architecture of neural network of 11 layers are shown in the table .

Table 1. Results for the predicted labels for the test dataset

Accuracy	Precision	Recall	Fmeasure
89.2%	0.89	0.89	0.893

4. Discussion

The implemented 11 and 5 layer network follows the dimensions and operations of the VGG architecture. Previous to this implementation, a 5 layer convolutional network with an output of 16, 32, 64 and 128 channels followed by a softmax classifier was trained with poor results. At first, this net was train with a crossentropy loss function which was not adequate for the classifier and only 5 epochs, obtaining an average classification accuracy in validation of 32%. Then the loss function was changed to negative log likelihood and the epochs were increased to 20, improving the performance up to 34%. However, the main improvement in the performance was obtained after changing the architecture to the first 4 VGG13 layers plus a fully connected.

When using the VGG architecture and the same train characteristics an average classification accuracy of 51%

was obtained in the validation set. Then the deepness of the net was changed to the 11 layer (10conv+1FC) and the train epochs were increased to 50, obtaining the best validation and test ACA of 87.9% and 89.2% respectively. This shows that the learning capability increased significantly when using a deeper net and that, possibly, the first experiment performed in which training was done in five epochs is not fairly accurate to describe and develop that CNN best potential.

Respecting overfitting, the average classification accuracy for the train set was calculated per epoch obtaining a difference in approximate 10% above validation ACA which shows little overfitting. The best results were obtained with a model having almost 100% accuracy in training (99.7%) which evidences a high train ACA might not be undesirable if the validation ACA remains close to that value.

The experiment performed without data augmentation was performed in the first non-VGG 5 layer CNN obtaining the same exact results of 32% in validation ACA. This can be explained due to the characteristics of the images as they are grayscale images in which color jitter does not produce similar images to that found in the train set. Also, the random horizontal flip might produce too similar images respecting the train set without increasing the variance of the images significantly. Another option might correspond to the loss function that, being inappropriate for the classifier does not increase the validation ACA despite the data augmentation performed.

finally, in order to understand the importance of the net architecture the 5 and 10 layer nets can be contrasted as the first one could be seen as an ablation test of the second one. We observe that the deepness of the networks corresponds to a parameter with high importance in the performance as the deepest layers increase almost 40% the validation ACA. This can be explained because of the image and texture characteristics. As the images are found in grayscale the most important information of texture relies on local patterns, which can be represented in a discriminative way in the upper layers. This means, the filters that present a high and discriminative response to this local patterns can be constructed in deep layers, so that, filters learned in 4 convolutional layers might not be sufficient enough for texture classification tasks.

5. Conclusions

Convolutional neuronal networks are complex models capable of learning the adequate parameters to develop in a specific task given determined inputs and labels. This systems perform both the representation and classification of the candidates based on the convolution operation in two dimensions, which allows them to construct and learn the most adequate filters for the given job, in this case, for a classification task. Specifically, VGG13 architecture is adequate for texture classification.

There is not a that methodology to predict an appropriate working architecture for a task, for this reason, the number of experiments performed is fundamental to determine the best architecture of the net. From the VGG architecture we observed that deep corresponds to one of the most important characteristics and that performance tends to increased with deepness. Also that the spacial information of the images is reduced while passing through convolutional layers until obtain a very low resolution but highly expressive representation of the image.

In texture classification over the Ponce dataset the local patterns are very important as they become the most discriminative elements of the images. This information can be retrieved using CNN and used with a classifier to classify textures. In addition, data augmentation can be helpful to generate similar train data when the variability or the number of images per class is reduced.

The number of epochs and batch size also corresponds to a very important parameter as the each net posses its own improvement rate. The full learning capacity of the net can be achieved in a determined number of epochs and the train should not be stopped before that point (identified by a stabilization in the loss and ACA). Also a bigger batch size can reduce the number of epochs needed to achieve the best performance with the cost of bigger computational resources needed.

Finally, the loss function and overfitting are fundamental issues respecting the CNNS. The loss function should be adequate for the architecture and the train and validation ACA must be maintained similar to avoid under or overfitting issues.

References

[1] S. Basu, M. Karki, S. Mukhopadhyay, S. Ganguly, R. Nemani, R. DiBiano, and S. Gayaka. A theoretical analysis of Deep Neural Networks for texture classification. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 992–999. IEEE, 7 2016.

[2] 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, 12 2016.

6. Code and Images

The code is available at the team's repository at *https://github.com/steff456/IBIO4680/tree/master/11-TextureCNN*

7. Appendix A: Results 5 layer network with 20 epochs

The following figures show the results for the 5 layer neural network with 20 epochs during the training and vali-

dation phase. The results for the train and validation dataset are less than the final architecture of 11 layers.

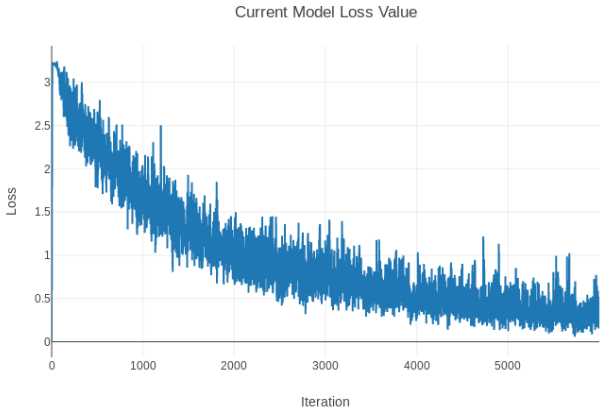

Figure 7. Loss value per iteration for the model

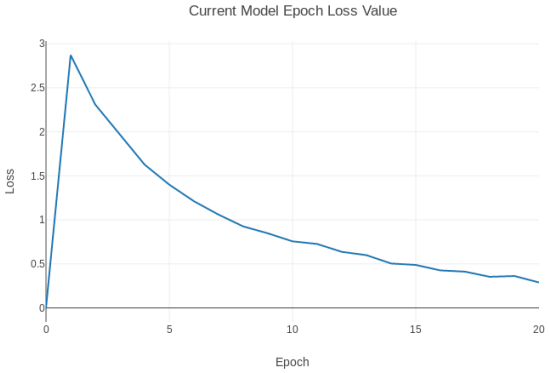

Figure 8. Loss value per epoch for the model in the training dataset

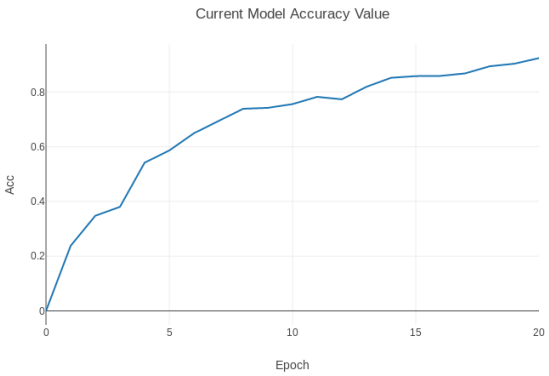

Figure 9. Accuracy value per epoch for the model in the training dataset

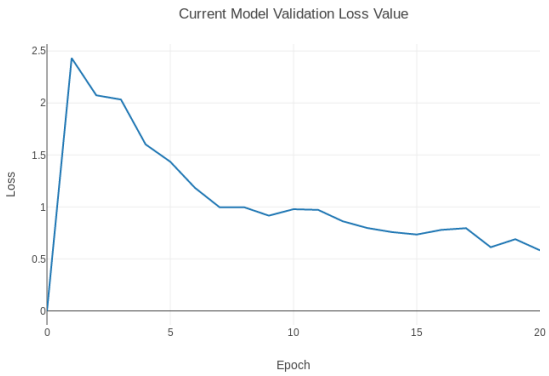

Figure 10. Loss value per epoch for the model in the validation dataset

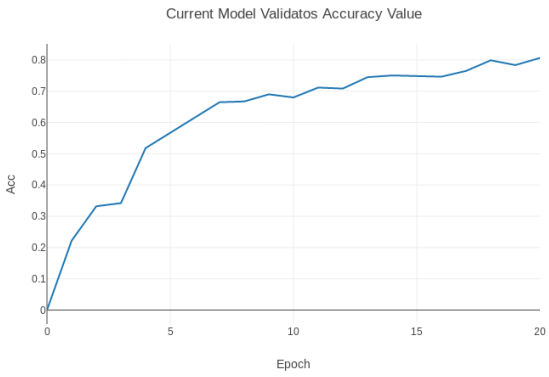

Figure 11. Accuracy value per epoch for the model in the validation dataset