Texture Recognition with AlexNet

Esteban Galvis
Universidad de los Andes
e.galvis10@uniandes.edu.co

1. Introduction

A fundamental problem in Computer Vision is representing what an image is. There have been ways to describe an image according to its color, intensities and more developed features. Yet, an image can be described by the textures it conveys. Traditional methods filter images with a handmade filter bank [1] to extract these texture descriptors of an image - which are only repeated patterns within it. However, with Neural Networks, these type of texture descriptors are learned on the first layers of a Neural Network and in this paper will see how it performs at classifying a dataset of textures.

2. Methods

2.1 Traditional texture feature extractor

In lab 5 we introduced a traditional way of extracting texture descriptors by using a filter bank. In this paper we used Ponce's dataset [2] and achieved 41% ACA in the training set and 22% ACA in the test set using a Nearest Neighbor Classifier. To see what type of filters there where, please see figure 1, 2.

2.2 Neural Network texture feature extractor

Neural Networks can learn representations [3] of things, in this case its no different. Usually a NN learns low level features at the beginning while learning richer and more complex features at the end. It is different from the traditional method in the way that the filter bank is learned in these earlier layers.

In the following paper we are not going to designed by scratch a neural network but instead use the award winning AlexNet [4] architecture. This architecture is special in the way that it has a convolution neural network as an encoder, which extracts all the low-level features and learns their

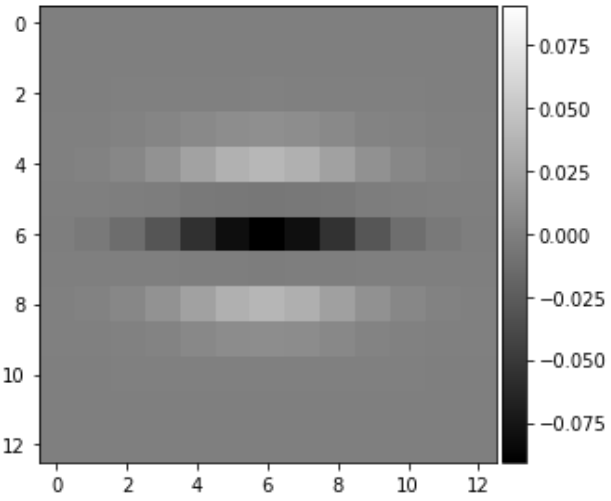

Figure 1: Example of a horizontal filter from the filter bank.

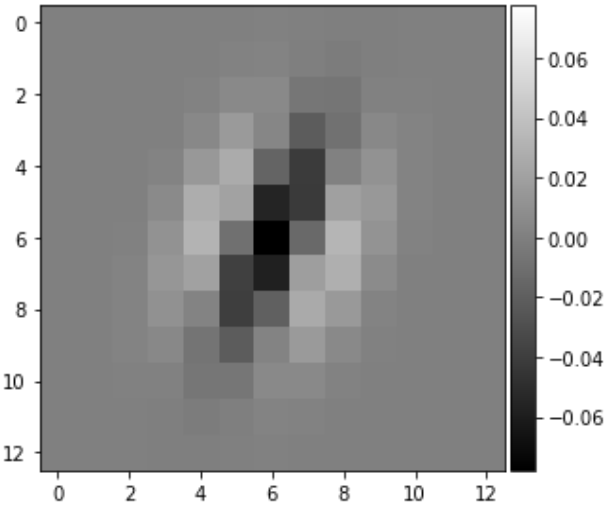

Figure 2: Example of a diagonal filter from the filter bank.

3. Results

Training of AlexNet with our dataset took eternity and also the problems I faced doing it just complicated the

process. Since training took a lot of time, I tried saving torch models but failed to do it every time. For the baseline setup I batched 4 images. The first model I trained for 10 epochs and achieved a mere positive result of 0.05% in accuracy. This made me realize that it needed more training.

I configured the next model to be of 60% and achieved 88% in training. I then started to follow the hyperparameter set up of AlexNet and added weight decay of 0.0005 and achieved 90.4% accuracy. I was really happy but didn't save the model and lost all the training at night, however if you want to see the loss see Figure 3.

Then tried to run several models but failed to do it since I executed several screens and ran a copied script of my training model. The training loss for these models never got better and it took ages to train, without considering all the GPU process that I took.

For the final models, I ran the same model of the beginning vs a model with Xavier initialization in a Jupyter Notebook. At the last epochs the model without Xavier initialization had a loss <1. Yet, the loss of the model Xavier looked to have a linear decrease which might be explained by initializing the weights uniformly which gives a more stable statistical model. This surely lets the training go deeper and longer without problem. Their respective val accuracy is 82% without Xavier and 68%. I crashed the kernel of the first model and couldn't get test scores with it to submitted to the competition and lost all the work I waited for. Fortunatley I still had the Xavier model and submitted that to the competion and the results were better than the Val set which might be explained by the Xavier uniformly initialization that means it can generalize better.

```
[1,  2000] loss: 3.858
[2,  2000] loss: 3.291
[3,  2000] loss: 3.222
[4,  2000] loss: 2.742
[5,  2000] loss: 2.453
[6,  2000] loss: 2.071
[7,  2000] loss: 1.706
[8,  2000] loss: 1.458
[9,  2000] loss: 1.268
[10,  2000] loss: 1.145
[11,  2000] loss: 1.003
[12,  2000] loss: 0.927
[13,  2000] loss: 0.860
[14,  2000] loss: 0.826
[15,  2000] loss: 0.745
[16,  2000] loss: 0.722
[17,  2000] loss: 0.671
[18,  2000] loss: 0.667
[19,  2000] loss: 0.608
[20,  2000] loss: 0.611
[21,  2000] loss: 0.581
[22,  2000] loss: 0.556
[23,  2000] loss: 0.546
[24,  2000] loss: 0.516
[25,  2000] loss: 0.524
[26,  2000] loss: 0.499
[27,  2000] loss: 0.486
[28,  2000] loss: 0.493
[29,  2000] loss: 0.458
[30,  2000] loss: 0.476
[31,  2000] loss: 0.437
[32,  2000] loss: 0.446
[33,  2000] loss: 0.420
[34,  2000] loss: 0.457
[35,  2000] loss: 0.424
[36,  2000] loss: 0.401
[37,  2000] loss: 0.410
[38,  2000] loss: 0.417
[39,  2000] loss: 0.387
[40,  2000] loss: 0.378
[41,  2000] loss: 0.366
[42,  2000] loss: 0.385
[43,  2000] loss: 0.378
[44,  2000] loss: 0.369
[45,  2000] loss: 0.358
[46,  2000] loss: 0.357
[47,  2000] loss: 0.371
[48,  2000] loss: 0.375
[49,  2000] loss: 0.365
[50,  2000] loss: 0.354
[51,  2000] loss: 0.345
[52,  2000] loss: 0.346
[53,  2000] loss: 0.348
[54,  2000] loss: 0.337
[55,  2000] loss: 0.336
[56,  2000] loss: 0.330
[57,  2000] loss: 0.347
[58,  2000] loss: 0.341
[59,  2000] loss: 0.309
[60,  2000] loss: 0.325
Finished Training
```

Figure 3: Loss

Paper References

[1] Th. Leung and J. Malik, Representing and Recognizing the Visual Appearance of Materials using Three-dimensional Textons, ICCV 1999.

[2] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. A Sparse Texture Representation Using Local Affine Regions. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 8, pp. 1265-1278, August 2005.

[3] Deep LearningIanGoodfellow, Yoshua Bengio, Aaron Courville(2016)http://www.deeplearningbook.org/
[4] Imagenet classification with deep convolutional neural networksA Krizhevsky, I Sutskever, GE Hinton - Advances in neural information processing systems, 2012