Laboratory 11 - Image classification using Neural Networks

Guillaume Jeanneret
Universidad de los Andes
Cra 1 #18a-12, Bogotá, Colombia
g.jeanneret10@uniandes.edu.co

Diana Herrera
Universidad de los Andes
Cra 1 #18a-12, Bogotá, Colombia
ds.herrera10@uniandes.edu.co

Abstract

Neural Networks have been one of the most powerful tools used recently for most Computer Vision tasks. They are built with different layers that learn filters capable of generating specific responses to each class. According to the output, backpropagation is used to modify the weights of each layer to improve results. The core problem of Neural Networks is designing their architecture: how many layers to use, the nature of each of them (convolutional, max pooling, fully connected, among others) and the activation function of the neurons. Therefore, empiric training to find the best architecture is the most important process when using Neural Networks. After many trials, we decided to use a network inspired on ImageNet architecture formed by 9 layers between four convolutional, two ReLU, two max pooling and a softmax layer at the end, to classify a set of images from ImageNet and also a set of texture images. The results were satisfactory achieving an objective close to 0.6, top 1 error around 0.2 and top 5 error around 0.02 for train and validation subsets.

1. Introduction

Neural Networks have demonstrated to be a powerful tool in most Computer Vision tasks. Since they were rediscovered, in the 21st century, their applications and performance has been improving and has not found limits yet. Based on biological inspiration, Neural Networks are built by layers, each one containing a certain amount of neurons. These neurons learn filters that respond to certain classes, the deeper the layer, the more specific the filter answer. For these structures, there are different kinds of layers that are usually combined throughout the architecture:

- Convolutional layers are the ones in which the output is a linear combination of the inputs, using weights that are initially random values that are recalculated using Backpropagation through each epoch (passing the whole train dataset through the network), to improve the result.
- Max Pooling layers are between some other layers and they subsample the outputs of the previous layer in which filters for the same features are repeated, by keeping only the maximum value among the vicinity.
- Non linear layers are built using a non-linear activation function after a linear filter. The most common function used is the Rectified Linear Unit (ReLU).
- Fully connected layers are usually located at the end of the network and they transform the output of the previous layer into a vector with the same number of data as the number of classes in which classification must be done. Each number corresponds to the probability of a certain class.
- Softmax layer is commonly located at the very end of the network and classifies the image by choosing the neuron with gratest activation of the last fully-connected layer.

The corresponding weights for each layer are initially random values, but to learn better weights, training stage is done in epochs, meaning a single time that the whole dataset is passed through the network, and for each epoch, backpropagation is done to recalculate the weights. The basic principle of backpropagation algorithm is derivating the loss function (error between output given by the network and ideal output) on each weight using the chain rule. The idea is to minimize error function through each epoch.

Nevertheless, one of the main problems of Neural Networks is the overfitting. Training the network requires to determine thousands of parameters, hence, a huge amount of training data is needed and in case it is not enough, overfitting is guaranteed. To minimize this problem, dropout is a common used technique. This method randomly eliminates a certain percentage of the connections among some layers. Furthermore, Ablation is also used and it consists on deleting a whole layer randomly and analizing the change in the performance and number of free parameters that must be determined.

For all these reasons, it is paramount to note that the major challenge of using Neural Networks is finding an useful architecture and training it properly. Although using a network trained for a different task and performing fine tuning has shown to be the best method to use these structures (specially based on ImageNet network), the only way to either train from scratch a new network or perform fine tunning is through empirical trial and error. Thus, this practice was focused on building a neural network on our own to find out what works, what does not and a promising architecture able to clasify images among 25 textures.

2. Materials and Methods

2.1. Dataset

In this scenario, we are using a database of textures, previously used in the laboratory 5 (features). In this case, 25 classes of images were determined to be classified. We used a total of 15.000 gray images of textures for training, 5.000 images for validation, and 5.000 for test, each with a 128x128 resolution. Because of the Anna Boch incident, the labels for the test sub-dataset are not given.

2.2. Neural Networks

We tried several different architectures, initially considering random layer order (1 to 11). Nevertheless, none of them gave a reasonable result. Hence, we designed a network inspired on ImageNet architecture, which showed best performance over all the other tested nets. The architecture used is shown in figure 6. Due to the fact that this architecture had promising results, a greater number of epochs was used (up to 75), different learning rate values ($1 * 10^-6 to 1 * 10^-4$) and smaller batches (from 100 to 50). After choosing the best set of parameters, we made the ablation test on the net.

The architecture of the first random neural networks are shown in figures 1, 2, 3, 7 and 11.

layer	0	1	2	3	4	5	6	7	8
type	input	conv	mpool	conv	mpool	conv	relu	conv	softmx
name	n/a	layer1	layer2	layer3	layer4	layer5	layer6	layer7	layer8
support	n/a	7	2	5	2	4	1	2	1
filt dim	n/a	1	n/a	20	n/a	50	n/a	500	n/a
filt dilat	n/a	1	n/a	1	n/a	1	n/a	1	n/a
num filts	n/a	20	n/a	50	n/a	500	n/a	25	n/a
stride	n/a	1	2	1	2	1	1	1	1
pad	n/a	0	0	0	0	0	0	0	0
rf size	n/a	5	6	14	16	28	28	32	32
rf offset	n/a	3	3.5	7.5	8.5	14.5	14.5	16.5	16.5
rf stride	n/a	1	2	2	4	4	4	4	4
data size	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN
data depth	NaN	20	20	50	50	500	500	25	1
data num	1	1	1	1	1	1	1	1	1
data mem	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
param mem	n/a	2KB	0B	98KB	0B	2MB	0B	195KB	0B

Figure 1. Structure of the first net, taken from the tutorial.

layer	0	1	2	3	4	5	6	7	8	9
type	input	conv	relu	mpool	conv	mpool	conv	relu	conv	softmx
name	n/a	layer1	layer2	layer3	layer4	layer5	layer6	layer7	layer8	layer9
support	n/a	7	1	2	5	2	4	1	2	1
filt dim	n/a	1	n/a	n/a	25	n/a	50	n/a	500	n/a
filt dilat	n/a	1	n/a	n/a	1	n/a	1	n/a	1	n/a
num filts	n/a	25	n/a	n/a	50	n/a	500	n/a	25	n/a
stride	n/a	1	1	2	1	2	1	1	1	1
pad	n/a	0	0	0	0	0	0	0	0	0
rf size	n/a	7	7	8	16	18	30	30	34	34
rf offset	n/a	4	4	4.5	8.5	9.5	15.5	15.5	17.5	17.5
rf stride	n/a	1	1	2	2	4	4	4	4	4
data size	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN
data depth	NaN	25	25	25	50	50	500	500	25	1
data num	1	1	1	1	1	1	1	1	1	1
data mem	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
param mem	n/a	5KB	0B	0B	122KB	0B	2MB	0B	195KB	0B

parameter memory|2MB (4.8e+05 parameters)|
data memory| NaN (for batch size 1)|

Figure 2. Structure of the second net.

layer	0	1	2	3	4	5	6	7	8	9
type	input	conv	relu	conv	mpool	conv	relu	mpool	conv	softmx
name	n/a	layer1	layer2	layer3	layer4	layer5	layer6	layer7	layer8	layer9
support	n/a	7	1	5	2	4	1	2	2	1
filt dim	n/a	1	n/a	25	n/a	50	n/a	n/a	500	n/a
filt dilat	n/a	1	n/a	1	n/a	1	n/a	n/a	1	n/a
num filts	n/a	25	n/a	50	n/a	500	n/a	n/a	25	n/a
stride	n/a	1	1	1	2	1	1	2	1	1
pad	n/a	0	0	0	0	0	0	0	0	0
rf size	n/a	7	7	11	12	18	18	20	24	24
rf offset	n/a	4	4	6	6.5	9.5	9.5	10.5	12.5	12.5
rf stride	n/a	1	1	1	2	2	2	4	4	4
data size	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN
data depth	NaN	25	25	50	50	500	500	500	25	1
data num	1	1	1	1	1	1	1	1	1	1
data mem	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
param mem	n/a	5KB	0B	122KB	0B	2MB	0B	0B	195KB	0B

parameter memory|2MB (4.8e+05 parameters)|
data memory| NaN (for batch size 1)|

Figure 3. Structure of the third net.

layer	0	1	2	3	4	5	6	7	8	9
type	input	conv	relu	conv	mpool	conv	relu	mpool	conv	softmx
name	n/a	layer1	layer2	layer3	layer4	layer5	layer6	layer7	layer8	layer9
support	n/a	9	1	7	2	6	1	2	2	1
filt dim	n/a	1	n/a	25	n/a	50	n/a	n/a	500	n/a
filt dilat	n/a	1	n/a	1	n/a	1	n/a	n/a	1	n/a
num filts	n/a	25	n/a	50	n/a	500	n/a	n/a	25	n/a
stride	n/a	1	1	1	2	1	1	2	1	1
pad	n/a	0	0	0	0	0	0	0	0	0
rf size	n/a	9	9	15	16	26	26	28	32	32
rf offset	n/a	5	5	8	8.5	13.5	13.5	14.5	16.5	16.5
rf stride	n/a	1	1	1	2	2	2	4	4	4
data size	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN
data depth	NaN	25	25	50	50	500	500	500	25	1
data num	1	1	1	1	1	1	1	1	1	1
data mem	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
param mem	n/a	8KB	0B	239KB	0B	3MB	0B	0B	195KB	0B

parameter memory|4MB (1e+06 parameters)|
data memory|NaN (for batch size 1)|

Figure 4. Structure of the fourth net.

layer	0	1	2	3	4	5	6	7	8
type	input	conv	mpool	conv	mpool	conv	relu	conv	softmx
name	n/a	layer1	layer2	layer3	layer4	layer5	layer6	layer7	layer8
support	n/a	5	2	5	2	4	2	2	1
filt dim	n/a	1	n/a	55	n/a	100	n/a	200	n/a
filt dilat	n/a	1	n/a	1	n/a	1	n/a	1	n/a
num filts	n/a	55	n/a	100	n/a	200	n/a	25	n/a
stride	n/a	1	2	1	2	1	2	1	1
pad	n/a	0	0	0	0	0	0	0	0
rf size	n/a	5	6	14	16	28	32	40	40
rf offset	n/a	3	3.5	7.5	8.5	14.5	16.5	20.5	20.5
rf stride	n/a	1	2	2	4	4	8	8	8
data size	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN
data depth	NaN	55	55	100	100	200	200	25	1
data num	1	1	1	1	1	1	1	1	1
data mem	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
param mem	n/a	5KB	0B	537KB	0B	1MB	0B	78KB	0B

parameter memory|2MB (4.8e+05 parameters)|
data memory| NaN (for batch size 1)|

Figure 5. Structure of the fifth net.

Lastly, the network inspired on ImageNet architecture is shown in the following figure, which was the one that gave best results.

layer	0	1	2	3	4	5	6	7	8	9
type	input	conv	mpool	conv	relu	mpool	conv	conv	relu	softmxl
name	n/a	layer1	layer2	layer3	layer4	layer5	layer6	layer7	layer8	layer9
support	n/a	11	4	5	1	4	2	1	1	1
filt dim	n/a	1	n/a	64	n/a	n/a	128	25	n/a	n/a
filt dilat	n/a	1	n/a	1	n/a	n/a	1	1	n/a	n/a
num filts	n/a	64	n/a	128	n/a	n/a	25	50	n/a	n/a
stride	n/a	1	4	1	1	4	1	1	1	1
pad	n/a	0	0	0	0	0	0	0	0	0
rf size	n/a	11	14	30	30	42	58	58	58	58
rf offset	n/a	6	7.5	15.5	15.5	21.5	29.5	29.5	29.5	29.5
rf stride	n/a	1	4	4	4	16	16	16	16	16
data size	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN	NaNxNaN
data depth	NaN	64	64	128	128	128	25	50	50	1
data num	50	50	50	50	50	50	50	50	50	1
data mem	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
param mem	n/a	31KB	0B	801KB	0B	0B	50KB	5KB	0B	0B

Figure 6. Best net built.

3. Results

The architecture chosen was the one corresponding to 6 with learning rate of 0.00001 and batch size 50. Results for the best net obtained are shown in the following figures in terms of objective, ErrorTop1 and ErrorTop5.

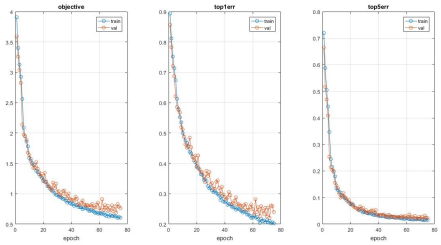

Figure 7. Train results obtained with the best net.

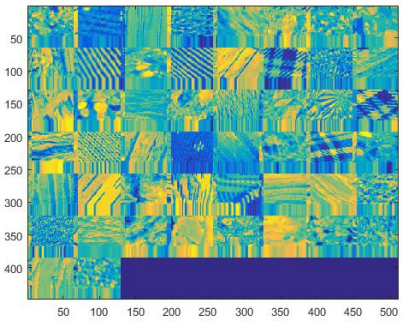

Figure 8. Filters of the last epoch with the best net.

Afterwards, ablation test was performed by removing some layers and comparing performance. Three tests were done: first layer to be removed was the last ReLU layer (8th), second one was the first ReLU layer (4th), third one was the first Max Pool (2nd layer). Results are shown in the following figures.

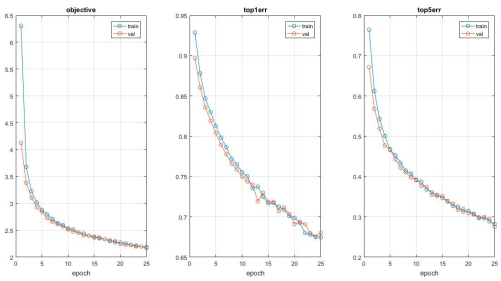

Figure 9. First ablation test removing last ReLU (8th layer).

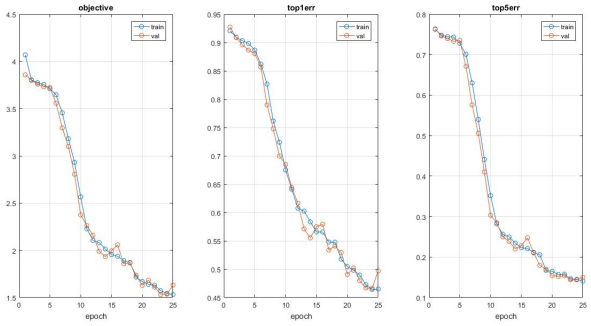

Figure 10. Second ablation test removing first ReLU (4th layer).

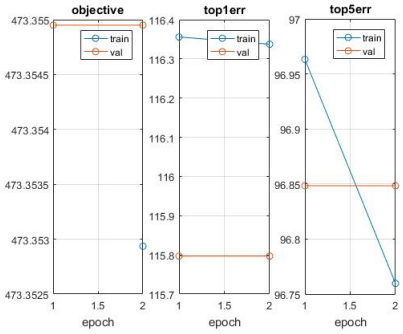

Figure 11. Third ablation test removing first Max Pool (2nd layer).

4. Discussion

Designing an useful architecture for a neural network is time consuming and a task that requires patience and, even more than creativity, some architectures to use as a model which have already been proven to work. Deciding the amount of layers to be used and their order seemed to be an almost random task, but after many failures and studying some examples of successful architectures, some patterns were recognized. Convolutional layers are generally used on the first part whereas fully connected at the end. Max pooling layers should be used after many convo-

lutional layers recognizing similar features, and these layers should not be as abundant as the previous ones. Although there are different non-linear layers, we used only ReLU and we found out that their location is crucial to improve or worsen the overall performance; when located between the very first convolutional layers, performance was badly affected whereas when used on the second halve of the architecture and in a small amount, results improved significantly (using many of these layers worsened the result).

After many trials (more than 8 architectures) and varying the learning rate, batch size and number of epochs, we finally obtained an architecture with promising results. These parameters had great impact on the performance of the architecture because, when using a very small learning rate ($1 * 10^-6$), many epochs were needed to get to a Top 1 Error close to 0.2 and there was a risk of converging at a local minimum far away from an optimal result. Results were neither good using a very large learning rate value, they were even worse, staying in high error values since the first epochs without decreasing. This is due to the fact that a higher learning rate might lead to overfitting [1]. The batch size had great impact on the overall performance, improving significantly with smaller batch size, this happens because training happens times faster than with bigger batches, as weights are updated more times per epoch, although the gradients estimated are noisier and results could be badly affected [2]. Lastly, the number of epochs used naturally increased the time-cost but with promising architectures, it was worth to confirm whether a minimum had already been found or if the algorithm could go further. The steepest slope for objective and errors was during the first epochs, whereas the latest epochs showed small changes in these values. Gradient descent can be easily performed on the first epochs and has smaller steps when approaching to a local minimum.

Now, going deeper in the architecture chosen, we inspired from ImageNet, which is a well known network. We built our network based on their architecture and then made a series of small changes (adding layers and varying the parameters explained previously) until we found a great result that was not overperformed with posterior trials. The comparison among ImageNet architecture and ours is shown in the following table.

Table 1. Comparison between our architecture and ImageNet one.

Layer #	ImageNet	Our Network
1	Conv	Conv (11*11*1*64)
2	Max P	Max P
3	Conv mm	Conv (5*5*64*128)
4	Max P	Relu
5	Conv	Max P
6	Conv	Conv (2*2*128*25)
7	Conv	Conv (1*1*25*50)
8	Max P	Relu
9	FC	SoftMax
10	FC	-
11	FC	-
12	SoftMax	-

When preparing the architecture, special attention was needed when specifying the dimensions of each convolutional layer, as the first two parameters could not be greater than width and height size of the images taken as input, the third parameter had to match the third dimension of the input, which corresponds to the number of filters of the previous convolutional layer and the number of channels for the first layer, and the fourth dimension determines the number of filters used which must be the length of the third dimension for the input of the following convolutional layer.

Another relevant characteristic of this practice was the usage of Jittering, which is basically a technique used during training to add noise to the images by translating the content slightly, so that the smooth functions do not generate a different output with slight changes from the input image [4]. This helps to improve the performance of the algorithm making it more generalizable, specially taking into account that the more images a NN uses to be trained, the less overfitting and the better performance. The function used was specially edited to work on 64x64 images, which was the case of our dataset.

Additionally, the ablation test gives important information about which layers are crucial and probably detect specific features for each class, and which ones are expendable and increase time cost but do not impact significantly final results. As the comparison was made through 25 epochs, the results are shown for that epoch among the three ablations and the complete network. They show that removing the first ReLU layer (Ablation test 2) does not have such a big impact when compared to removing the last ReLU (Ablation test 1), because results are close to the ones obtained with the whole network in the first case but on the second one they are lot worse. This demonstrates that non-linear

layers such as ReLU lead to better results, specially when used after some convolutional layers, but that even without them the network has an almost acceptable performance. On the other hand, Max Pooling layers demonstrated to be unexpendable, because the third ablation test showed bad results when removing a layer of this kind. Further ablation tests were to be done but unfortunately the machine used stopped compiling.

Table 2. Comparison between our architecture and ImageNet one.

Net	Objective		Top1Error		Top5Error	
-	Train	Valid	Train	Valid	Train	Valid
Original	**1.1**	**1.1**	**0.36**	**0.37**	**0.065**	**0.065**
Abl1(8th)	2.2	2.2	0.67	0.68	0.27	0.27
Abl2(4th)	1.155	1.165	0.46	0.5	0.145	0.15
Abl3(2nd)	473.35	473.355	116.35	115.8	96.75	96.85

5. Conclusions

- Choosing the amount of images per batch will help to achieve more easily the goal of the net. Furthermore, using big batches will be faster to complete an epoch, but the convergence is not in the optimal path. On the other hand, using small batches makes training times faster because weights are updated more times per epoch, leading to better results in less epochs.
- Another characteristic is change rate of the error. Firstly, the error will decrease greatly, giving acceptable results at a early time. Once the training algorithm was advanced, the magnitude of the rate of error decreased to a value very low. This is useful when long term projects need a preliminary result.
- The ablation test will get a understanding of the task of each layer and information about whether the time to compute it is worth the increase in the overall performance. In the ablation tests done, we found that non-linear layers such as ReLu have greater impact when used after many convolutional layers, but that even without them the network has a reasonable performance. Nor is the case of Max Pooling layers, without whom the network does not work.
- Getting various deformations, such as spatial translation, permits to get more data available for training, thus, getting more information for a faster convergence. This can be achieved by using Jittering.
- Finding a useful architecture for a neural network is a procedure that can only be achieved by trial and error, although getting inspired from already built networks such as ImageNet, helped to have a base to start with. Posterior addition or deletion of layers as well as adjustment of proper learning rate and batch size to obtain better results must be done empirically.
- To create the architecture of a neural network careful attention must be paid to the size of input and output of each layer, specially convolutional ones.
- There are some rules that help to build a useful architecture, such as locating each layer according to it's nature.

References

[1] Stack Exchange, "Choosing a learning rate", Datascience.stackexchange.com, 2017. [Online]. Available: https://datascience.stackexchange.com/questions/410/choosing-a-learning-rate. [Accessed: 11- May- 2017].

[2] "Dramatic performance improvements with smaller batch sizes?", reddit, 2017. [Online]. Available: https://www.reddit.com/r/MachineLearning/comments/3k0dp1/dramatic_performance_improvements_with_smaller/. [Accessed: 11- May- 2017].

[3] S. Ilic, "Tracking and detection in Computer Vision", 2009-2009 TUM.

[4] Sarle, W. 2001. "What is training with Jitter?" [online] Available at: ftp://ftp.sas.com/pub/neural/FAQ3.html#A_jitter

[5] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection", 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05).

[6] A. Vedaldi and A. Zisserman, "Object category detection practical", 2014 [online] Available at http://www.robots.ox.ac.uk/%7Evgg/practicals/category-detection/

[7] P. Viola and M. Jones, "Robust Real-Time Face Detection", International Journal of Computer Vision, vol. 57, no. 2, pp. 137-154, 2004.