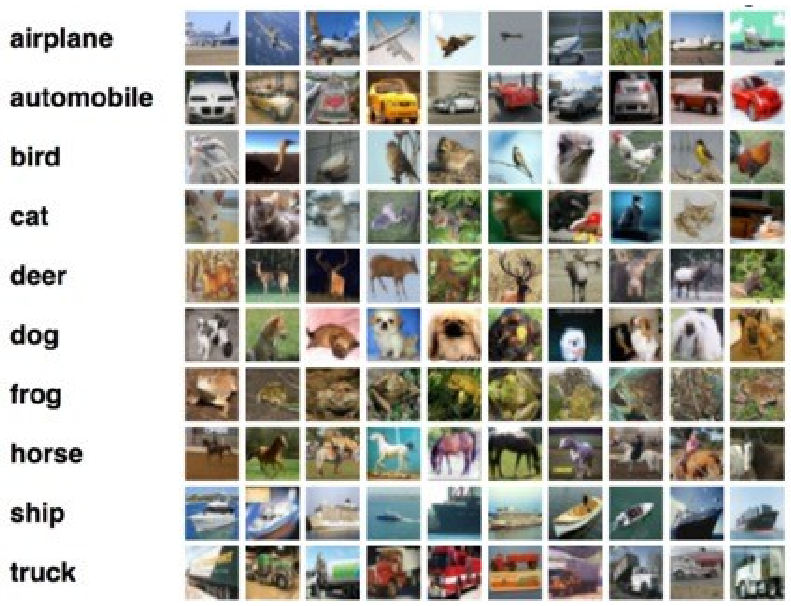
[EQ2425] Analysis and Search of Visual Data - AY 2019/2020
Project #3 Report

IMAGE CLASSIFICATION USING CNNS

Mayank Gulati
(gulati@kth.se)

Federico Favia
(favia@kth.se)



Contents

I. Introduction

The aim of the project is to build a simple convolutional neural network [1] and tune its hyper-parameters to investigate its performances for image recognition [2]. This is a different approach to this problem with respect to the previous projects, where the features are extracted directly on data to perform different matching algorithms. Here instead the features are self-learnt from the network in the training process.
The CIFAR-10 dataset [4] which comprises 50,000 training examples and 10,000 test examples of images from 10 classes is used for training and testing data. The dataset samples are formatted as 32×32 pixel RGB color images.The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

II. Deep Learning Framework

To build the convolutional neural network we are using the deep learning framework Pytorch [5]. PyTorch is an open source machine learning library based on the Torch library used for applications such as computer vision and natural language processing. It is primarily developed by Facebook's artificial intelligence research group. It is free and open-source software released under the Modified BSD license. Although the Python interface is more polished and the primary focus of development, PyTorch also has a C++ front-end.
We are using Google Colab to train the model with Nvidia T4 GPU. Co-laboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud.

III. Data Pre-processing

In order to normalize pixel value data between -0.5 to 0.5, we are using in-built PyTorch function of *transform.Normalize* with mean as 0.5 and standard deviation as 1 respectively.
Normalization is very useful since parameters can have wildly different ranges and the gradient descent will not spend more time training on one of them relatively to the other. Also performing operations on floating point values with large differences in scale could give unexpected errors [3].

IV. Default Architecture

It is a three-layer convolutional neural network based on the following configuration:

1. Convolutional layer 1: Filter size: 5×5, stride: 1, zero-padding: 'valid', number of filters: 24.
   This is the key element of a CNN, performing a convolution with a sliding filter on the input image.
2. ReLu activation function.
   It sets to zero all the negative values.
3. Pooling Layer: Max pooling, filter size: 2×2, stride: 2.
   It divides by two the size of the layer for reducing over-fitting and computational complexity.
4. Convolutional layer 2: Filter size: 3×3, Stride: 1, zero-padding: 'valid', number of filters: 48
5. ReLu.
6. Pooling Layer: Max pooling, filter size: 2×2, stride: 2
7. Convolutional layer 3: Filter size: 3×3, Stride: 1, zero-padding: 'valid', number of filters: 96
8. Pooling Layer: Max pooling, filter size: 2×2, stride: 2.
9. Fully connected layer 1: Output size: 512.
   Before it the output is stretched in a column vector and then feeds it to the FC layers, which is basically an artificial neural network.
10. ReLu.
11. Fully connected layer 2: Output size: 10.
    The number of the classes in the decision problem are 10.

12. SoftMax classifier.
This is useful to change the final output of FC layer into a vector of probabilities to the corresponding class, deciding for the maximum one.

V. Default Training Parameters

A built-in stochastic gradient descent algorithm

$$for\ i\ in\ range(samples\ in\ mini\_batch) : w_i \leftarrow \alpha \cdot (y_i - y_i)x_j^i \tag{1}$$

to train the network is used, based on the following configuration:

- Learning rate: $\alpha = 10^{-3}$.
- Size of mini-batch: 64.
- Training epochs: 300.

Then, some of this parameters will be changed to see their effects on performance.

VI. Changes in the network structure

Before applying any change, the default model accuracy is **39.22 %** (average top-1 recall rate). This is a disappointing result, meaning it makes more mistakes than good predictions. This could be because of not shuffling the data and not doing batch normalization. Therefore, it may perform better on more than 300 epochs to converge. The loss of that model can be seen in (Figure 1).

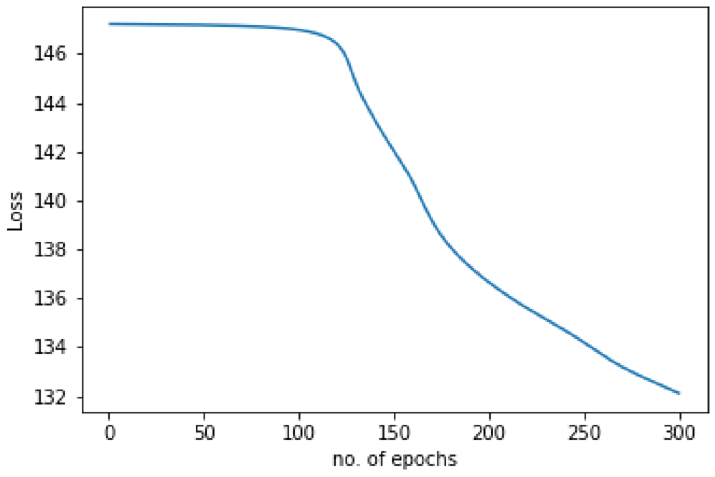


Figure 1: Default model loss.

VI.i. Architecture

Below, a succession of modifications are performed on the architecture. The following section talks about these modification and how these affect the accuracy results and training time in the end.

4-A. 1 *Number of filters:* The default number of convolutional filters in the three layers are changed from [24; 48; 96] to [64; 128; 256]. The accuracy achieved is **41.08%**. The results are slightly improved than before because more patterns are detected to better represent the data, but still is lower the random matching threshold.

2 *Number of layers:* A fully connected layer between the original fully connected layer 1 and 2, followed by a ReLu activation function, is added. The resulting accuracy is **21.32 %**, as it cannot capture significant used to classify different object labels. As the result becomes worse, we proceeded with the first changed configuration .

4-B. *Filter size:* Based on the preferred configuration above, the filter sizes of Conv1 and Conv2 are changed to 7×7 and 5×5, respectively. The recall decreases to **38.37 %**, so this configuration is not considered.

4-C. *Leaky ReLu:* All ReLu activation functions are replaced by Leaky ReLu. We opted for a slope = 0.2 in negative direction. The achieved accuracy is the current best in the process until this step: **41.35 %**, very little increased from the previous one. Indeed, with Leaky ReLu we can worry less about the initialization of the neural network. In the ReLU case, it is possible that the neural network never learns if the neurons are not activated at the start (lots of unknown dead ReLU). However, ReLU computes slightly faster, as we can see in the below table.

4-D. *Dropout:* Keeping the Leaky ReLu, a dropout regulation between FC1 and FC2 is added, with dropout rate = 0.3. The recall decreased barely to **39.58 %**, thus is not an improving change. Even though it could be hard to believe, dropout has recently become successful in computer vision applications because of its capacity to better generalize the same input, dropping down to zero the weights of some pattern with a certain probability. If in our case, this does not improve performance, it is most likely because of the shallowness of the architecture.

4-E. *Batch normalization:* Finally a batch normalization layer is added after each Leaky ReLu, resulting in the best accuracy achieved of **71.06 %**, as we had assumed previously, and a training time of 6742s (around 1 hour and 50 minutes). Batch normalization improves results because it normalize data also inside the network when non-linear operations are performed (the input was normalized at the beginning). It operates subtracting the mean, dividing by the variance, and multiplying by a parameter β and adding a parameter γ.

VI.ii. Training parameters

5-A *Batch size:* Keeping the last configuration, the batch size is then increased from 64 to 256, resulting in an a decreased accuracy of **60.86 %**, thus this configuration will not be used. The training time is decreased to 5702s because more data are fed in a bigger batch one at time.

5-B *Learning rate:* The learning rate is changed to $\alpha = 0.1$. We obtained a very low recall rate of **16.09 %** because it is not converging optimally to the *minima*, actually missing it. This is the worst achieved accuracy, as we can see in a bad loss trend in the plot below (Figure 2), and this configuration will not be used.

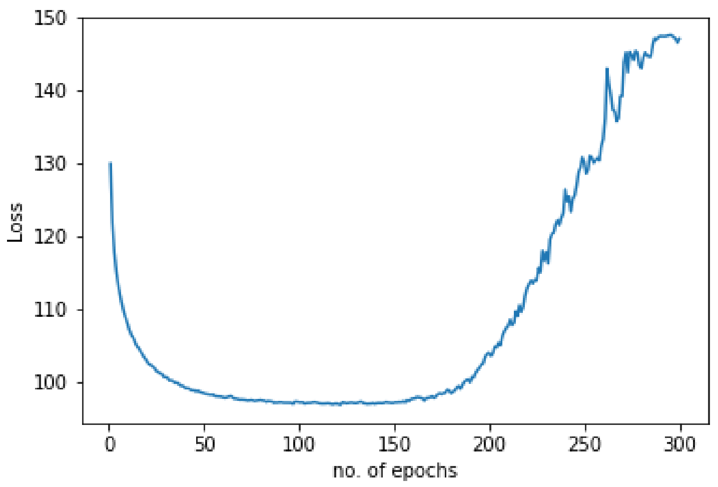


Figure 2: Worst model 5-B loss.

5-C *Data shuffling:* In the end, the last change made is shuffling the data. This process is conducted at once while loading the training dataset instead of each epoch to save time. Already with this operation, the training time takes 6975s (around 2 hours), being the maximum of all the settings. This is the best achieved configuration, with a **72.62 %** recall rate, as shown in the loss trend below which is converging better (Figure 3).

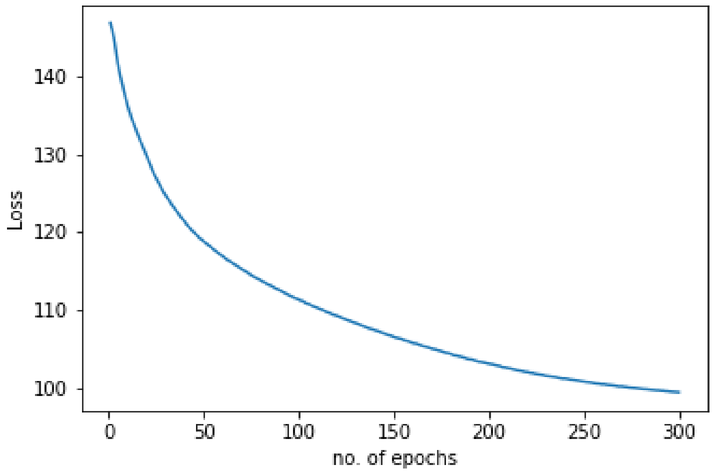


Figure 3: Best model 5-C loss.

Following, a summary of the obtained results, both in accuracy and training time, relating to the changes, is shown to the reader (Table 1).

Table 1: Obtained results.

Sr. No.	Model type	Accuracy	Training time
1	Default	39.22%	4109s
2	4-A.1	41.08%	6251s
3	4-A.2	21.32 %	4143s
4	4-B	38.37 %	4492s
5	4-C	41.35 %	6636s
6	4-D	39.58 %	6363s
7	4-E	71.06 %	6742s
8	5-A	60.86 %	5702s
9	5-B	16.09%	6596s
10	5-C	72.62%	6975s

VII. Conclusions

After trying different modifications of neural networks we concluded that the model with more convolutional filters, active batch normalization after each Leaky ReLu activation function and data shuffling provides best results when conducted for 300 epochs.

References

[1] D. Nister and H. Stewenius Scalable recognition with a vocabulary tree, in Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), June 2006.

[2] M. Flierl. Analysis and Search of Visual Data [EQ2425], Course Slides, KTH, 2019.

[3] Stanford University. Lectures: CS231n: Convolutional Neural Networks for Visual Recognition.

[4] University of Toronto, Department of Computer Science. CIFAR-10 Dataset.

[5] Paszke, Adam and Gross, Sam and Chintala, Soumith and Chanan, Gregory and Yang, Edward and DeVito, Zachary and Lin, Zeming and Desmaison, Alban and Antiga, Luca and Lerer, Adam. Automatic Differentiation in Pytorch.