Image Classification using CNNs
EQ2425, Project 3

Author 1
remich@kth.se

Author 2
sebcas@kth.se

October 15, 2021

Summary

The goal of this research is to use a neural convolutional network to recognize pictures. The goal of this paper is to illustrate our work and our thoughts on how to apply various modifications in the fundamental parameters and arrangements in network architecture development. We'll also go through the different sorts of layers and algorithms that are employed. We'll go through how the net learns in different epochs using backpropagation and stochastic gradient descendent in particular. We will utilize the accuracy of the top 1 as a consequence of every modification we make by evaluating the right proportion of comparisons between expected and accurate outcomes. Because of the network's inherent features, not all of the results obtained are straightforward to evaluate. The findings demonstrate that it is important to be aware of which type of data we are dealing with, and sometimes the outcomes don't go as expected.

1 Introduction

The goal of this project is to retrieve the best network structure and training parameters for image recognition using convolutional neural networks. The project begins with the construction and training of a three-layer convolutional neural network using a provided setup. Then, we modify the networks parameters and the training settings to evaluate the best configuration. To do such a decision, the accuracy performance is taken into account. In the end, we will also analyze the time performances of different implementations.

A convolutional neural network (CNN or ConvNet from the English convolutional neural network) is a type of feed-forward artificial neural network inspired by the organization of the animal visual cortex, in which the individual neurons are arranged in such a way that they respond to the overlapping regions taxing the visual field

They are particularly well suited to image recognition. The ConvNet architecture can be used to train a network and subsequently to obtain a category or numerical label.

Different sorts of layers make a convolutional neural network. Convolution layers are typically the initial layers, and they execute a dot product of the convolution kernel with the layer's input matrix. The activation function of this product is generally ReLU. The convolution procedure creates a feature map as the convolution kernel slides along the input matrix for the layer, which then contributes to the input of the following layer. Other layers such as pooling layers, fully connected layers, and normalizing layers can be further added [2].

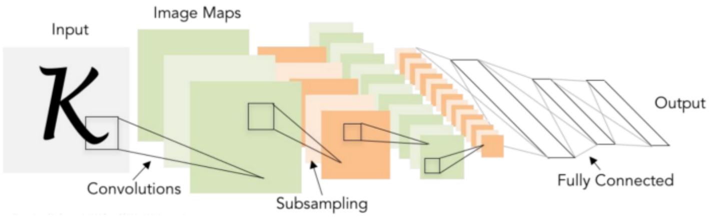


Figure 1: Example of CNN with a standard configuration.

2 Dataset & Environment

The dataset we are using is the CIFAR-10 [1] which consists of 60000 32x32 color images distributed homogeneously in 10 classes, where 50000 are training images and 10000 test images. The classes are mutually exclusive, meaning that there is no overlap between the 2 classes. Given that we decided to use Python for our implementation, we chose the relative version of 163 MB.

The framework we decided to work with is Pytorch [5]. It is the most used open-source machine learning framework nowadays.

To manage our implementation we used Google Colaboratory given its advantages such as no need to configure an environment and free access to GPUs. GPU is a powerful component for Data scientists since they are made to efficiently compute matrix operations, exploiting the simple concept of parallelization.

3 Data Pre-processing

To prepare the information to feed the network, some passages should be followed to obtain correct learning. Given that we analyze the “properties” of the images, we have to consider them under the same parameters. So, the “features” that describe the images need primary attention.

To normalize the distribution of features, we need to compute a normalization of the input images pixels. This is performed to take away the mean information to have a more homogeneous distribution. If we don’t consider such, the feature values will be proportionally different from each other, making the Convolutional Neural Network fail. Indeed, this could lead to an oscillating behavior, making the network unable to center onto minima, slowing down all training.

In our case, the image pixels have been normalized between -0.5 and 0.5.

4 Default Network Structure

The project requirement was about to build a three-layer convolutional neural network based on the following configuration:

Convolutional layer 1: Filter size: 5 x 5, stride: 1, zero-padding: ’valid’, number of filters: 24

ReLu

Pooling Layer: Max pooling, filter size: 2 x 2, stride: 2

Convolutional layer 2: Filter size: 3 x 3, Stride: 1, zero-padding: 'valid', number of filters: 48

ReLu

Pooling Layer: Max pooling, filter size: 2 x 2, stride: 2

Convolutional layer 3: Filter size: 3 x 3, Stride: 1, zero-padding: 'valid', number of filters: 96

Pooling Layer: Max pooling, filter size: 2 x 2, stride: 2

Fully connected layer 1: Output size: 512

ReLu

Fully connected layer 2: Output size: 10

Softmax classifier

As it is, this configuration should perform around 60% of accuracy, being already a good starting point.

4.1 Default Training Parameters

To set the behavior of the network, some fundamental parameters should be set. The first one we analyze is the learning rate. This value is multiplied by the weights while the network is learning, so the higher its value the greater the change of these weights. It is logical to conclude that if the value is high, the network will converge faster, but at the same time, it may skip minima, resulting in a non-optimal solution. On the other hand, if the value is low, the time required to converge can be considerable, but it assures us to reach the minima.

Another important parameter is the number of minibatch. To reduce the complexity in computing the weights, a subset of data is preferred to take. Inside this subsection, the back-propagation is performed, so the smaller the size of this subset, the faster the computation. On the other hand, if the minibatch dimension is too little, the result will be noisier, not taking into consideration the correlation with all the rest of the dataset. The back-propagation performed in this project has seen the Stochastic Gradient Descent as a discriminator. To further improve the performances of a CNN, the method Adam has shown to be more appropriate.

The last parameter we touched is the number of epochs. This is the most obvious setting we have to consider, expressing how many times the network should iterate the same dataset. The more the epochs the better the chances to reach better learning. On the other side, forcing the number of interactions too high can lead to over-training causing a worsening in the efficiency and learning capabilities.

For this project, the default training parameters was set as follow:

Learning rate = 10^{-3}

Minibatch size = 64

Number of epochs = 300

5 Training Parameters

In this section, the trials are performed, changing the parameters and the settings seen before. In order to determine the best configuration, at each test is chosen the one that gives the best accuracy. Timing is also evaluated to find additional relations within the tests.

5.1 Number of layers vs. Number of filters

The major changes are performed in this section. Firstly, the number of filters per convolutional layer is substituted with $[64, 128, 256]$ respectively. Now, the number is increased and we will have more data to compute and more dynamism in the learning process. Usually, this change leads to better values in the accuracy levels, indeed, we got almost 6% of improvement.

Proceeding then with more filters, the next test is about adding a fully connected layer between the first and the second fully connected layers. The number of units is equal to 128, followed by a ReLu activation function.

The output from the convolutional layers represents high-level features in the data. The output could be flattened and connected to the output layer, but adding a fully connected layer is usually a cheaper way of learning non-linear combinations of these features. Of course, adding more fully connected layers increases the computational complexity. In our test, we got almost 2% of improvement adding such layer.

At the end of this first section of the test, the total accuracy reached was about 41%.

5.2 Filter size

In this section we try to use filters with bigger dimensions, the first convolutional layer is the dimension of 7x7 and the second layer has dimensions of 5x5. We prove that with this approach we can reach a lower accuracy of 40,89% in front of the previous result of 41,05%.

We assume that this behavior is given by two factors the first is that being very small images filters so large in relation, they can not recognize the high frequencies. Secondly, by not adding padding the filters lose a lot of information not being able to analyze the edges of the images, already very small.

5.3 ReLU vs. Leaky ReLU

In this section we analyze two different activation functions ReLU and Leaky ReLU, these two functions are defined as from image. The ReLU function stands for the rectified linear unit and is a type of activation function. Mathematically, it is defined as $y = max(0, x)$. Rectified Linear is first used in Restricted Boltzmann Machines [4].

Leaky Rectified Linear activation is first introduced in acoustic model [3]. Mathematically, we have $y_i = \begin{cases} x_i & if x_i \geq 0 \\ \frac{x_i}{a_i} & if x_i < 0 \end{cases}$

where a_i is a fixed parameter in range $1, +\infty$. In the original paper, the authors suggest to set a_i to a large number like 100.

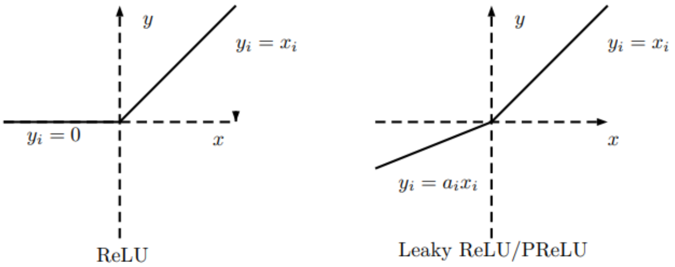


Figure 2: ReLU and Leaky ReLU.

Our previous result with ReLU is around 40% while with Leaky ReLU we are around 64%, we expected an improvement, but the result surprised us with a 24% increase in accuracy.

5.4 Dropout vs. without Dropout

The dropout is a powerful method when we are dealing with over-fitting. It may happen that in a neural network certain inputs and nodes are preferred rather than others. If this behavior happens consecutively in more epochs, means that the network is “stuck”, and the probability to learn decreases. To understand if this is the case, a value of dropout is set to 0.3.

Dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on more or less responsible for the inputs [6]. A value of 0.3 means that at each iteration the 30% of the total number of neurons are dropped randomly.

As a consequence, this should also help to improve timing performances.

Our outcomes gave us an improvement of almost 5% in accuracy, with a value at this phase of 69.03%, while instead, timing has increased.

5.5 Batch Normalization Layer

This method consists of normalizing activation vectors using mean and variance. Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1. In this way also the noise introduced by neurons and back-propagation is in a way filtered, leading to faster and better convergence. There are cases tho, where the batch normalization layer makes activation values explode during the evaluation phase, making the model returns loss equal to NaN.

Batch normalization is a method for training very deep neural networks that standardizes each minibatch batch’s to a layer. This stabilizes the learning process and significantly reduces the number of training epochs needed to construct deep networks.

From our simulations, we found a strong decrease in accuracy performances with a drop of almost 9%. So far we concluded that in our case it gave us an unexpected result. We could not easily determine why this happened, but since this is a fine-tuning approach, we just have to follow the instructions and drop the batch normalization layer from our configuration.

5.6 Batch size

The batch size refers to the number of instances from the training dataset utilized in the estimation of the error gradient. It is a critical hyper-parameter that affects the learning algorithm's dynamics.
Smaller batch sizes are noisier, but they have a regularizing impact and result in lower generalization error. It's simpler to fit one batch's worth of training data in memory with smaller batch sizes (i.e. when using a GPU).
When training neural networks, batch size affects the accuracy of the error gradient estimate. We try to have a bigger batch size of 256 and the result is 25,87% that prove a worst performance.

5.7 Learning rate

The learning rate is a hyper-parameter that regulates how much the model changes each time the model weights are changed in response to the predicted error. A low learning rate might lead to a lengthy training process with low accuracy, whereas a high number could lead to learning a sub-optimal set of weights too quickly or an unstable training process.In fact we try to use a learning rate of 0.1 and the result is 16,07%.

5.8 Data shuffling

Data shuffling, together with minibatch aims to minimize the training loss. The problem we could face is the same. Given a convex function, it may have multiple local minima making it hard to determine the global minima to a machine learning algorithm. What this technique tries to do is to shuffle the data at each iteration to produce better performances. This solves the scenario when the algorithm is stuck into local minima. It raises the probability to make it jump out in the next iteration to search for the global minima, changing the shape of the function. In our case we have an accuracy of 45% that does not increase a lot our results.

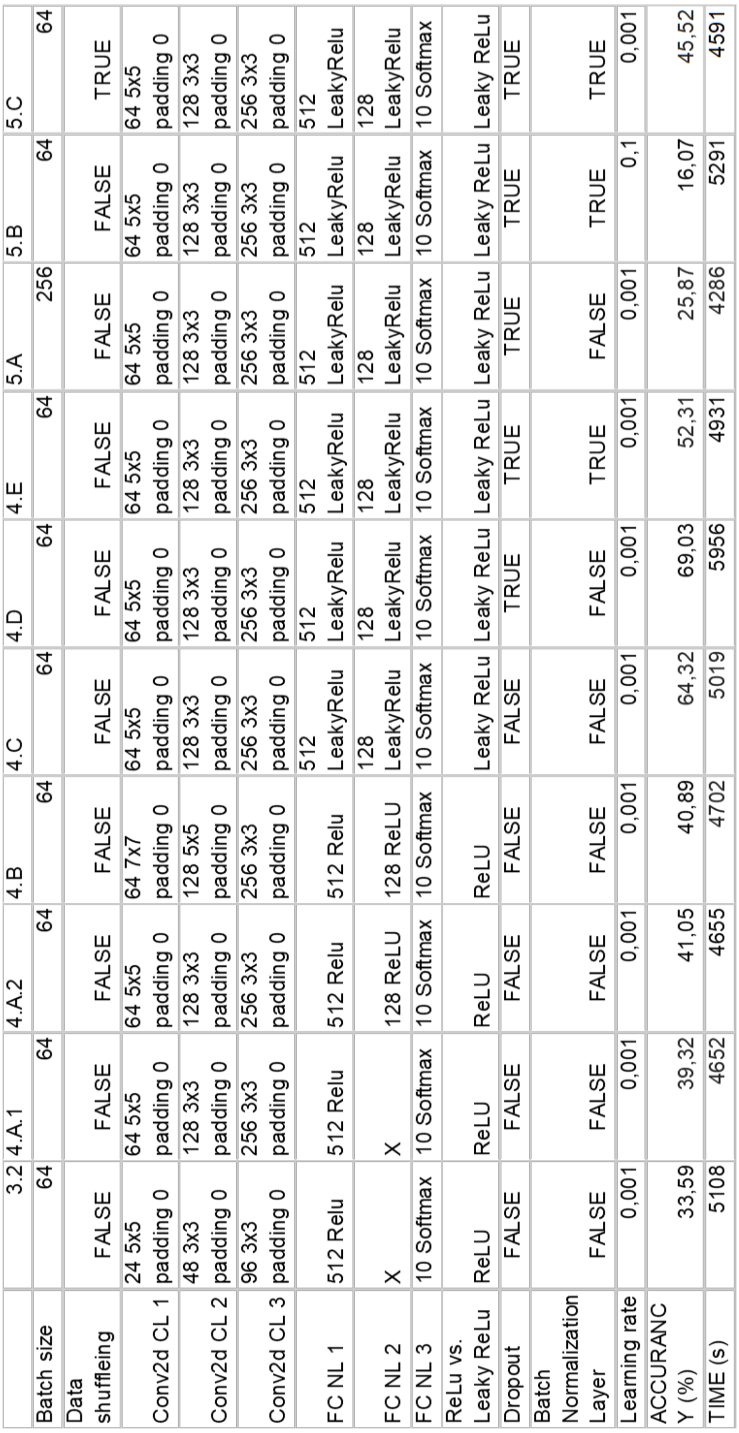
6 Conclusions

As a general conclusion, we can say that this project gave us a lot to think of, considering all the unexpected results. We had to restart the process of tests and evaluations multiple times because we realized small errors activating flags or setting parameters. It is easy to get a wrong result and it is not trivial to conclude if the outcome is wrong or it is correct under the current conditions.

The result that gave us more disturbance was batch normalization. In all the papers we consulted this method seems to perform better in terms of speed and accuracy. We found very few resources where it was stated that sometimes it could even give a drastic result, but nothing sufficient to cite in the report. What we can say is that we tried to deal with this unexpected outcome for a while, but given the difficulty to understand clearly what happened and why it was, we just rely on this data and we kept going with the project.

Generally, given the fine-tuning approach, we can conclude that this experience made us aware of what machine learning is capable of, and why it is so important to study the input data. An ad-hoc pre-processing has to be done to avoid unexpected issues with the network later. In the end, better segmentation of the data may help in these cases, since the data fed into the network may differ every time, making the difficult to produce a generalized solution true.

Appendix



	3.2	4.A.1	4.A.2	4.B	4.C	4.D	4.E	5.A	5.B	5.C
Batch size	64	64	64	64	64	64	64	256	64	64
Data shuffleing	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE
Conv2d CL 1	24 5x5 padding 0	64 5x5 padding 0	64 5x5 padding 0	64 7x7 padding 0	64 5x5 padding 0	64 5x5 padding 0	64 5x5 padding 0	64 5x5 padding 0	64 5x5 padding 0	64 5x5 padding 0
Conv2d CL 2	48 3x3 padding 0	128 3x3 padding 0	128 3x3 padding 0	128 5x5 padding 0	128 3x3 padding 0	128 3x3 padding 0	128 3x3 padding 0	128 3x3 padding 0	128 3x3 padding 0	128 3x3 padding 0
Conv2d CL 3	96 3x3 padding 0	256 3x3 padding 0	256 3x3 padding 0	256 3x3 padding 0	256 3x3 padding 0	256 3x3 padding 0	256 3x3 padding 0	256 3x3 padding 0	256 3x3 padding 0	256 3x3 padding 0
FC NL 1	512 Relu	512 Relu	512 Relu	512 Relu	512 LeakyRelu	512 LeakyRelu	512 LeakyRelu	512 LeakyRelu	512 LeakyRelu	512 LeakyRelu
FC NL 2	X	X	128 ReLU	128 ReLU	128 LeakyRelu	128 LeakyRelu	128 LeakyRelu	128 LeakyRelu	128 LeakyRelu	128 LeakyRelu
FC NL 3	10 Softmax	10 Softmax	10 Softmax	10 Softmax	10 Softmax	10 Softmax	10 Softmax	10 Softmax	10 Softmax	10 Softmax
ReLu vs. Leaky ReLu	ReLU	ReLU	ReLU	ReLU	Leaky ReLu	Leaky ReLu	Leaky ReLu	Leaky ReLu	Leaky ReLu	Leaky ReLu
Dropout	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE
Batch Normalization Layer	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE
Learning rate	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,1	0,001
ACCURANCY (%)	33,59	39,32	41,05	40,89	64,32	69,03	52,31	25,87	16,07	45,52
TIME (s)	5108	4652	4655	4702	5019	5956	4931	4286	5291	4591

Figure 3: Results.

Who Did What

The work was divided equally based on the previous knowledge of the components and the work was organized and developed in parallel.

References

[1] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. "CIFAR-10 (Canadian Institute for Advanced Research)". In: (). URL: `http://www.cs.toronto.edu/~kriz/cifar.html`.

[2] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: `10.1109/5.726791`.

[3] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013.

[4] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines." In: *ICML*. Ed. by Johannes Fürnkranz and Thorsten Joachims. Omnipress, 2010, pp. 807–814. URL: `http://dblp.uni-trier.de/db/conf/icml/icml2010.html#NairH10`.

[5] *PyTorch*. URL: `https://pytorch.org/`. (accessed: 15.10.2021).

[6] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: `http://jmlr.org/papers/v15/srivastava14a.html`.