Image classification using CNNs

EQ2425, Project 3

Aleix Espuña Fontcuberta
aleixef@kth.se

Ioannis Athanasiadis
iath@kth.se

October 15, 2021

Summary

In the context of this project, we trained a shallow neural network with the purpose of classifying images derived from the CIFAR-10 dataset. We were given a default structure and training set-up which we progressively modified according to the guidelines specified in the assignment's description. Based on the accuracy performance in the test set, we identify which practises lead to the most evident improvements in the test-set accuracy. The greatest improvements were achieved when increasing the number of filters, adding a dropout layer and using batch normalization with a high learning rate.

1 Introduction

In this project, we try to find the convolutional network (CNN) that performs best in the classification problem given by the popular CIFAR-10 dataset. The dataset contains 50000 training images and 10000 test images. Each image is a color image of size 32×32 and it can belong to one of 10 different classes (multi-classification problem).

ORDER	OPERATION	PROPERTIES
1	convolution	kernel size=5×5; stride=1; number of filters=24
2	ReLU	non-linear activation function
3	pooling	kernel size=2×2; stride=2
4	convolution	kernel size=3×3; stride=1; number of filters=48
5	ReLU	non-linear activation function
6	pooling	kernel size = 2×2, stride=2
7	convolution	kernel size = 3×3, stride=1, number of filters=96
8	pooling	kernel size=2×2; stride=2
9	linear transform	output neurons 512
10	ReLU	non-linear activation function
11	linear transform	output neurons 10
12	Softmax	non-linear activation function

Table 1: The initial CNN model structure that will be gradually improved.

2 Problem Description

The CNN model that we use is the one sketched in Table 1. It receives a feature vector $\mathbf{x}$ representing an image and it outputs a vector $\mathbf{p} \in \mathcal{R}^{10}$ of class probabil-

HYPERPARAMETER	Value
learning rate	0.001
batch size	64
number of epochs	300

Table 2: Initial set of fixed hyperparameters that will be used to determine the best network structure.

ities. Our prediction y^{pred} corresponds to the class with the highest probability output. The model is trained via maximum-likelihood using backpropagation. Therefore, an appropriate choice for the loss function is the cross-entropy loss. As optimizer, we use a simple stochastic gradient descent method with mini-batches. In order to find the best network, we gradually make changes to the initial network structure shown in Table 1 under the fixed set of hyperparameters listed in Table 2. The performance metric we use is the accuracy on the test set, which is the number of correct classifications made on the set.

2.1 Software and hardware

We implement the algorithms with Python 3 using the popular Pytorch library. We generate the results with a laptop powered with an NVIDIA GeForce GTX 1650 graphics card.

3 Results

The default configuration displayed in Table 1 gives a training accuracy of 73.97% and a test accuracy of 66.45% using the hyperparameters of Table 2. In the next sections we will modify the network's structure and well as the training settings, according to the provided instruction, in order to investigate which modifications improve the performance the most. The modifications are carried out in the order they were provided while the best performing configuration is kept as the basis for the upcoming ones.

3.1 Network structure

At this stage, we experiment with various modification related to the network's structure.

3.1.1 Number of layers vs. Number of filters

At first, we investigated two independent changes. The first change consists of changing the number of filters of each convolution layer. We change the number of filters in each convolution layer from 24, 48, 96 to 64, 128 and 256 respectively. The second experiment consists in keeping the configuration shown in Table 1 but adding a fully connected layer of 128 units with a ReLU function between the two linear layers. The training and test accuracy is listed in Table 3. Based on the conducted experiment, we observed that both changes resulted in increased performance, in terms of prediction accuracy in the test set. However, we identified that increasing the number of filters is more beneficial compared to increasing the number of fully connected layers. Therefore, we will proceed with the standard structure but with the new number of filters [64, 128, 256].

STRUCTURE CHANGE	TRAINING ACCURACY	TEST ACCURACY
Number of filters	83.50%	**68.14**%
Additional layer + ReLU	74.88%	67.15%

Table 3: Comparison between the accuracy obtained in two independent changes of the network structure. Starting from the structure described in Table 1, the first change modifies the number of filters in each convolutional layer to 64, 128 and 256 respectively. The second change adds a fully connected layer of 128 neurons and a ReLU between the steps 10-11 listed in Table 1.

3.1.2 Filter size

Secondly, we experimented with the filter size of the convolutional layers. More specifically, we changed the filter sizes of the first two layers, from 5×5 and 3×3 to 7×7 and 5×5 respectively. In Table 4 we report the result of this change. Based on that, we observed that increasing the number of filter decreases the network's performance. Consequently, we keep the best structure found in section 3.1.1.

STRUCTURE CHANGE	TRAINING ACCURACY	TEST ACCURACY
Filter sizes	82.11%	66.13%
Keeping best config.	83.50%	**68.14**%

Table 4: Comparison of accuracy when the filter sizes of the best configuration change to 7×7, 5×5 and 3×3. The best test accuracy is obtained when the filter sizes are not changed and the previous best configuration is kept.

3.1.3 ReLu vs. Leaky ReLu

In here, we experimented by changing the activation functions from being ReLu to Leakly ReLu. In Table 5, we report how this change affects the network performance. Based on that, we identified that when Leaky ReLu is used, the performance slightly decreases. This decrease in performance is counter-intuitive, since one would expect the Leaky ReLu to perform better compared to the naive ReLu activation. More specifically, Leaky ReLu mitigates the issue of dead neuron which results in having vanishing gradient according to [1]. However, in another source [2] is mentionted that changing the activation from ReLu to LeakyReLu does not result in consistent performance improvement. Based on these and considering that the accuracy performance is very similar for both activation functions, not observing the expected improvement should not concern us much.

STRUCTURE CHANGE	TRAINING ACCURACY	TEST ACCURACY
Best config. with LeakyReLU	82.94%	67.99%
Best config. with ReLU	83.50%	**68.14**%

Table 5: Performance comparison when the non-linear activation function of the best configuration is changed from ReLU to LeakyReLU.

3.1.4 Dropout vs. without Dropout

Here we add a dropout layer with a probability rate p=0.3 between the fully connected layer 1 and 2. Table 6 shows that adding dropout is beneficial to the network. The training accuracy drops from 83.50% to 79.83% but the test

accuracy jumps from 68.14% to 69.73%. This aligns with the fact that dropout is a form of regularization in which only a randomly sub-sampled portion of each layers' output units are passed onto the next layer. This procedure is only carried out during training and it essentially enforces the network to learn each layers' parameters more independently which in turn decreases the probability of overfitting. Therefore, we keep the dropout layer in the network structure.

STRUCTURE CHANGE	TRAINING ACCURACY	TEST ACCURACY
Best config. with Dropout	79.83%	**69.73**%
Best config.	83.50%	68.14%

Table 6: Performance comparison when a dropout layer with rate p=0.3 is added between the fully connected layers 1 and 2. Adding dropout gives a significant performance improvement.

3.1.5 Batch Normalization Layer

Finally, we add a batch normalization layer after each ReLU function. Table 7 shows that batch normalization produced a slight performance gain. Batch normalization offers stability to the network. It allows increasing the learning rate and converging faster to the minimum [3]. Keeping in mind that in the next section we will increase the learning rate, we keep the batch normalization layers in our final CNN structure, which is shown in the Appendix, in Table 11.

STRUCTURE CHANGE	TRAINING ACCURACY	TEST ACCURACY
Best config. with batch norm	99.84%	**70.12**%
Best config. without batch norm	79.83%	69.73%

Table 7: Performance comparison when a batch normalization layer is added after each ReLU function. Adding batch normalization gives a slight accuracy gain.

3.2 Training Settings

After having identified the best performing neural networks' structure, sketched in Table 11, we experimented with some modifications related to the training settings.

3.2.1 Batch size

First, we experiment witch the batch size. More specifically, we increase it from 64 to 256. The performance of that change is provided in Table 8. Based on that, we conclude that, given a learning rate of 0.001 increasing the batch size, slightly decreases the accuracy performance in the test set. So we will keep the batch size fixed to size 64. We also note that increasing the batch size decreases the training time. This is because increasing the batch size reduces the number of gradient updates per epoch. Moreover, although computing the gradient for larger batches does takes more time, by utilizing a bigger portion of the available computational resources, we achieve a decrease in the training time.

BATCH SIZE	TRAINING TIME	TEST ACCURACY
64	8046 seconds	**70.12**%
256	7481 seconds	68.72%

Table 8: Performance comparison when the hyperparameter batch size is changed from 64 to 256. We also list the training time for each case.

3.2.2 Learning rate

In here, we experiment with the learning rate by changing it from 0.001 to 0.1. According to the result provided in Table 9, we observe that increasing the learning rate significantly the test-set accuracy.

LEARNING RATE	TEST ACCURACY
0.001	70.12%
0.1	**72.99**%

Table 9: Great improvement of test accuracy when the learning rate is increased to 0.1.

3.2.3 Data shuffling

So far, we have been training without shuffling the data during training. In here, we shuffle the data after each epoch. The shuffling results in having completely different batches during the gradient calculation. In the Table below we report the performance of data shuffling. Based on that, we notice that shuffling slightly decreases the performance in the test set. One would expect that by shuffling the data after each epoch would result in more representative batches which in turn would result in more competent training and consequently performance. One possible reason for this counter-intuitive result might be that the default data order give rise to quite representative batches and thus further shuffling does not provide any added value. Based on that and considering that the performance's decrease is only about 0.2% we could attribute it to random factors such as weight initialization etc.

Shuffling after each epoch	TEST ACCURACY
yes	72.78%
no	**72.99**%

Table 10: Performance comparison when shuffling the data or not after each epoch.

4 Conclusions

Summing up, in this project we were given a default network architecture and training set-up and we made various modification with the purpose of increasing the accuracy performance in the training set. We found that the best network structure is the one listed in Table 11 trained with a batch size of 64 samples and a learning rate of 0.1, without shuffling the data. According to the conducted experiments we identified that changing the filter size, the learning rate and employing dropout are the most beneficial modifications. Adding batch normalization gave minor improvements at the beginning, but it proved to be very valuable when increasing the learning rate later on. Additionally, we also

observed that increasing the number of fully connected layers also resulted in increasing the accuracy performance but to a lower extent compared to the aforementioned ones. On the other hand, increasing the filter size of the first two convolutional layers, changing the activations from ReLu to Leaky ReLu, increasing the batch size and data shuffling, decreased the performance in the test set. When it comes to the activation function change and data shuffling, the decrease is performance was very small which we could also attribute it to random factors. In fact, we conducted the modifications provided in the assignment's description a second time. In this second time, we identified that changing the activation function from ReLu to Leaky ReLu resulted in slightly increase accuracy in the test set. Overall, we can safely conclude that a simple architecture of 3 convolutional layers and two fully connected ones is sufficient to classify the CIFAR-10 images and achieve a relatively high accuracy of 72.99% in the test set.

Appendix

ORDER	OPERATION	PROPERTIES
1	convolution	kernel size=5×5; stride=1; number of filters=64
2	ReLU	non-linear activation function
3	Batch Norm	Normalization layer
4	pooling	kernel size=2×2; stride=2
5	convolution	kernel size=3×3; stride=1; number of filters=128
6	ReLU	non-linear activation function
7	Batch Norm	Normalization layer
8	pooling	kernel size = 2×2, stride=2
9	convolution	kernel size = 3×3, stride=1, number of filters=256
10	pooling	kernel size=2×2; stride=2
11	linear transform	output neurons 512
12	ReLU	non-linear activation function
13	Batch Norm	Normalization layer
14	Dropout(p=0.3)	Regularization layer
15	linear transform	output neurons 10
16	Softmax	non-linear activation function

Table 11: The best CNN structure found at the end of section 3. It gave the best test accuracy of 72.99% when trained with a batch size of 64 samples and a learning rate of 0.1.

Who Did What

In this project, we coded and run the experiments independently. We followed this approach to ensure that are results and conclusions are valid. When it comes to the report writing, Aleix wrote the Introduction and the Problem description section. We jointly wrote the Results and the Conclusion sections. Finally, we both went through the text and corrected minor errors.

References

[1] *https://www.quora.com/What-are-the-advantages-of-using-Leaky-Rectified-Linear-Units-Leaky-ReLU-over-normal-ReLU-in-deep-learning*

[2] *https://cs231n.github.io/neural-networks-1/*

[3] Sergey Ioffe and Christian Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, *https://arxiv.org/abs/1502.03167*