703642, Advanced Machine Learning course 2th December, 2018

**Assignment 2**

**Programming Project: Convolutional Neural Network**

Kamal Zakieldin, Yusuf Ipek

1. **Introduction**

Tuning a model in deep learning is always a tricky part, as each model is a special case, we have a lot of hyper-parameters that we can tune, and tweak our model will extremely depend on them, and they are totally depend on the type of the problem the model tries to solve. Most of the time following the normal procedures will get a good model, however getting a very good model without losing generalization is a difficult point that need a lot of understanding of your problem, your dataset, and your architecture frame. all these points should drive us to tune our model with the suitable hyper-parameters.

In this report we are describing our fine-tuning procedure we have applied to the well-known **CIFER10** problem, and our experiments we have made, first we introduce our problem and our tuning showing the experiments we had, and the results we got, then we discuss our chosen model and illustrating the differences between our model and the original one, and comparing the accuracy and the loss values, and finally adding some recommendations for future tuning.

1. **Hyper-Parameters:**

As we have a lot of parameters to tweak, we tried to be precise about our chosen parameters.

We started first with tuning our **batch-size** which determines the number of samples in each **mini batch sizing** we have, it helps to adjust between the two extremes: accurately move the slope of errors down toward a minimum error value and without taking too long time, so, we tried to find the correlation between the number of epoch and the batch-size, so we ran the model with different values of them. For epochs we choose 10, 25, 50, 75 and 100. A high batch size requires much memory, thus in order to fit the data into the graphic memory we take 8, 16, 32, 64 and 128 for the batch size. We ran it with the different configurations for epoch and batch-size and stored the accuracy for train and test set.

figure 1 shows all the different configuration and its accuracy, which helps us to compare them and decide which configuration to discard. The best accuracy could be achieved for batch size 64 with 100 epochs, where the accuracy is 0.79, but also batch size 32 and 128 reached a high accuracy. As the accuracy for batch size 8 and 16 is lower in contrast to 32, 64 and 128 we discard them.

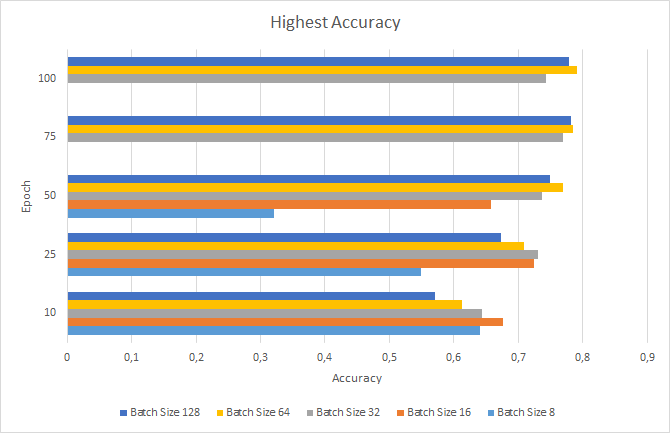
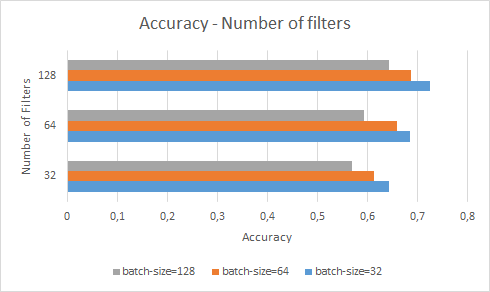


Figure 1 shows the Accuracy of the testset of Cifar10 with different configurations of the number of epoch and the batch-size.

Then we have continued our experiments looking for the convolution layers, we have started to enhance the first convolution layer as the convolutional layer is responsible for the convolutional operation in which feature maps identifies features in the images. Any convolution layer contains some vital parameters such as kernel that we convolve the image with, and the number of chosen filters which describe our problem complexity and choosing a good number of filters to detect all important features like edges, corners, and generally detect complex shapes and features in the next layers. increasing the number of filters in the initial input layer from 32 to 128 have made a promising progress, so we tried to find the accurate number of filters for the convolution layers. Increasing the number of the filters of the first and second convolution layers to 64 and 128 have achieved increasing of the accuracy for batch sizes 32, 64 and 128 as shown in Figure 2, this experiment is done for only 10 epochs.



*Figure 2 shows the Accuracy of the model with different number of filters. Model trained with 10 epochs.*

During tuning the convolution layer, we have suffered some times from overfitting specially when we ran our network to 100 epochs, that’s why we started to look for regularization options we have.

We first introduced a more **dropping out** where you turn off part of the network's layers randomly to increase legalization and hence decrease overfitting. We use dropout when the training set accuracy is much higher than the test set accuracy. We also thought about using dropout on the input layer as well as the hidden layers. This has been proven to improve deep learning performance, however, with a too low values we got negligible effects, and with too high values caused underfitting, as shown in table 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dropout percentage (100 epochs) | Train accuracy | Test accuracy | Train loss | Test loss |
| 0% dropout in the input layer | 0.99988 | 0.8289 | 0.00258 | 0.7562815 |
| 15% dropout in the input layer | 0.91444 | 0.8524 | 0.25846557 | **0.4306899** |
| 25% dropout in the input layer | 0.92466 | 0.8462 | 0.28886295 | 0.456715 |
| 0% dropout with regularization | 0.93464 | 0.8366 | 0.37794 | 0.50077 |
| 15% dropout with regularization | 0.91084 | **0.8584** | 0.31013 | 0.47525568 |

Trying a dropout of 25% of the input layer, showed that increasing the dropout of the input layer to 25% has reduce the accuracy and increase the loss, which means that increasing the dropout in the input layer has reduced the input features, but it does not achieve better values towards generalizing the model, so we neglected it, and consider the dropout of only 15%.

It was clear that the dropout term was not enough to reduce the overfitting, that’s why we have added a **regularization** term **Ridge regression** (**L2**), which adds “*squared magnitude*” of coefficient as penalty term to the loss function.

Data Augmentation

Optimizers

1. **Architecture**

In this section we describe our final architecture and parameters we thought it is more efficient, however, we are sure that there is still a lot of options to fine-tune the model more to get better result.

1. **Results and Comparison:**

In this section we would like to compare our results we got with the original results of the example, we have achieved