703642, Advanced Machine Learning course 3rd 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.

2. 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 size** 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 epochs 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 epochs and batch-size and stored the accuracy for the training and testing set.

Figure 1 shows all the different configurations 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, hence we discard them.

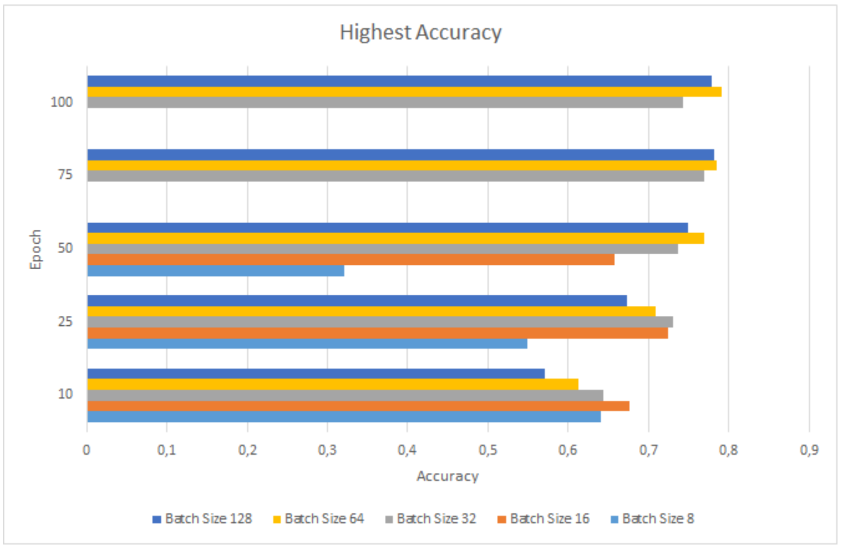

Figure 1 shows the Accuracy of the testset of Cifar10 with different configurations of the number of epoch and the batch-size.

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 the kernel that we convolve the image with, and the number of chosen filters which decide our problem complexity, and describe 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 1st convolution 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 1st convolution layers pair 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.

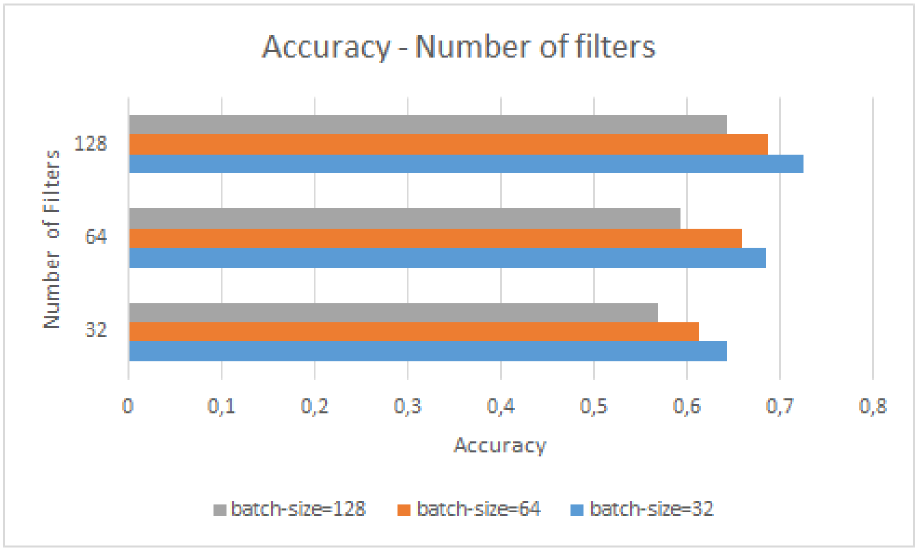

Figure 2 shows the Accuracy of the model with different number of filters. Model trained with 10 epochs.

During tuning the convolution layer, we have suffered 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 **dropping out** techniques, where you turn off part of the networks' layers randomly to increase regularization 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 1st convolution layer as well as the other 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, best practice was between 15% to 25% for the 1st conv. Layer.

	Dropout percentage with 100 epochs, 64 batch size	Train accuracy	Test accuracy	Train loss	Test loss
a	Without dropout without regularization	**0.99988**	0.8289	**0.00258**	0.756282
b	15% dropout in 1st conv. Layer without regularization	0.91444	0.8524	0.258466	0.43069
c	25% dropout in 1st conv. Layer without regularization	0.92466	0.8462	0.288863	0.456715
d	Without dropout with regularization L2	0.94764	0.8366	0.37794	0.50077
e	**15% dropout with regularization L2**	0.98764	**0.8904**	0.130619	**0.428236**
f	25% dropout with regularization L2	0.91084	0.8584	0.31013	0.475256
g	**15% dropout with regularization L2, (only 75 epochs)**	0.979	0.8802	0.18238	0.4714

Table 1 shows the accuracy and the loss for these different approaches, the model (a) without dropout suffered from overfitting and the predictive power was so poor, and the model (e) with 15% dropout and regularization has a reasonably good accuracy, but still suffering from overfitting as it trained more than it should, (g) is the same model (e) with early stopping after 65 epochs.

Trying a dropout of 25% of the 1st conv. layer, showed that increasing the dropout has reduced the accuracy and increased the loss, so it could achieve better values towards generalizing the model, We have also increased the dropout of the middle hidden layer to be %35 instead of 25%.
So, dropping out has reduced the overfitting slightly, however, it was clear that this reduction was not enough, hence we have added a **regularization** term using **L2** regularizer (**Ridge regression**), which adds "squared magnitude" of coefficient as penalty term to the loss function.
From table 1, we have noticed also that the model (e) has suffered from a little bit overfitting at the last few epochs, as it trained more than it should so we have Stopped training the model before reaching an overfitting state using **early stopping** techniques, stopping the model after 75 epochs has shown less overfitting.

Then, we have thought about the benefits of using data augmentation, **Data augmentation** is a method by which you can virtually increase the number of samples in your dataset using data you already have. For image augmentation, it can be achieved by performing geometric transformations, resizing, changes to color, brightness, contrast or by adding some noise.

We did not spend a lot of effort with it, we just tested if it is useful more to our problem to make a data augmentation or not, after few trials, we found that preventing data augmentation has reduced the accuracy a lot, which make sense, hence using data augmentation for that problem was useful.

It is also important to mention the optimization techniques, we have tried **ADAM** optimizer which stands for adaptive moment estimation, ADAM is an optimization algorithm that can be used to optimize the weights iteratively, by adapting its values according to the current state of the weights. Also, **RMSPROP** optimizer which stands for root mean square propagation, can utilize the magnitude of recent gradients to normalize the gradients. by uses a moving average of the root mean squared gradients, after some trials ADAM has shown a slightly better accuracy than rmsprop.

3. 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.

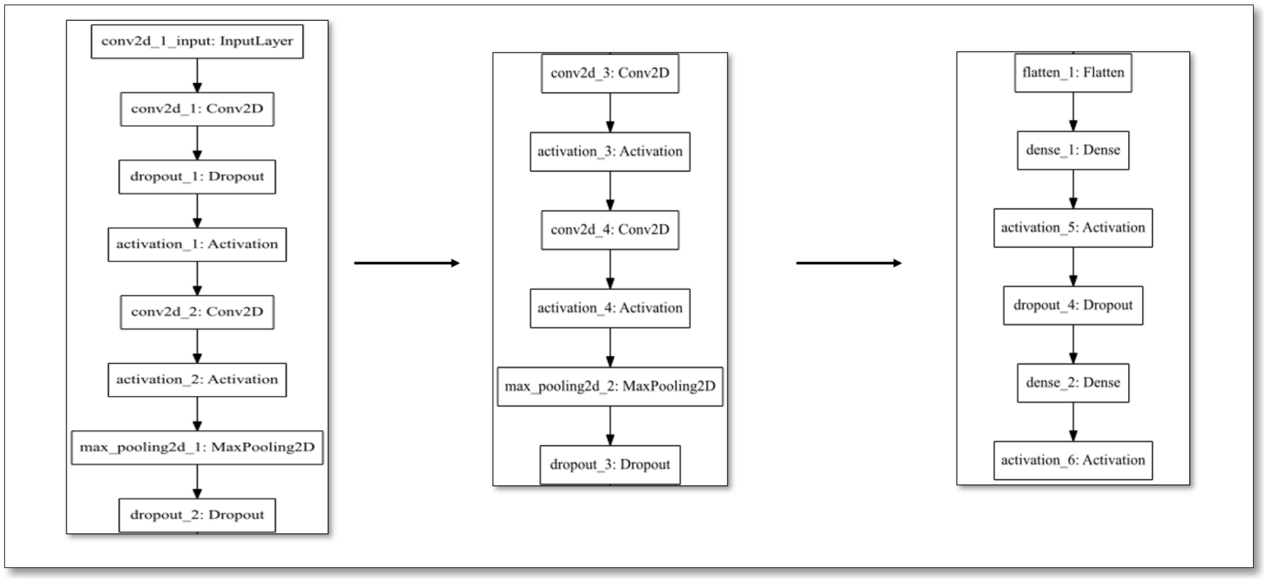

Figure 3 shows our final chosen architecture

As shown in figure 3, we have built our network similarly as the original provided network with just one update, we have added a dropout in the 1st convolution layer that receives the inputs, with 25% dropout for dropout_1, then we have updated the dropout of the hidden layers, so for dropout_2 we have increased to 35%.

a large network usually performs better with higher dropout, giving the model more opportunity to learn independent representations, that's why we have increased the dropout for dropout_3 and dropout_4 to be 50%, which helped us towards better generalization.

We have chosen 64 as our batch size, our convolution layers we have chosen 128 as our number of filters for the first conv. layer pair, and we double the number of filters for the next convolution layers after each maxpooling layer, to detect more complex features, we also added a L2 regualarizer as a regularization term, and finally we have updated our kernel size for the 1st conv. layer pair to be (5*5).

Kernel size is tricky, as the larger the kernel, the slower in time. On the other hand, large kernels enable you to learn more complex templates thus enable the network to have stronger prediction, but we should also consider the accuracy may saturate in specific range without significant progress, we could not check larger kernel sizes because of our limited GPUs, however (5*5) kernel has shown better results that we will discuss in section 4.

Finally, we trained our model using data augmentation under ADAM optimizer, table 2 contains the final results of our chosen parameters compared against the original model results.

	Models accuracy and loss with 75 epochs	Train accuracy	Test accuracy	Train loss	Test loss
a	Original model (No regularization,3*3 kernel, num. filters = 32)	0.80562	0.7857	0.575	0.652
b	15% dropout with regularization L2, kernel (5*5)	**0.97524**	0.8803	**0.162**	0.452
c	**Our model (higher dropout, L2 Regularization, kernel (5*5)**	0.95672	**0.8819**	0.21	**0.438**

Table 2 shows the results of the original model at (a), (b) shows a model with 15% dropout for the 1st conv. Layer and dropout of 35% for dropout_3, against our final model (c) with 25% dropout for the 1st conv. Layer and dropout of 35% for dropout_2, and 50% dropout for dropout_3.

4. Results and Comparisons

In this section we would like to compare our results we got with the original model results for 75 epochs[1]. we want first to highly enlighten the success of increasing the dropout by showing the difference between the model (b) and model (c) in table 2, model (b) has a 15% dropout in the 1st conv. layer and dropout of 35% for droupout_3 layer, in our final model we had a higher dropout values as we described earlier, the test accuracy was already better, and the difference between train and test accuracy is around 7% in the final model instead of 9.5% in (b) model, and also we have achieved a better loss value and the difference between train and test loss was less.

Finally, comparing our final model with the original one, we have achieved a test accuracy of around **88%** which was higher than the original test accuracy with more than around **10%.** We also got a better loss values, we have decreased the loss to become **0.438** instead of **0.652** in the test set. We still think that better results should have less difference between the training and the testing set.

In conclusion, we have tried some approaches hopefully to fine-tune our model to achieve better results, we are sure, even we got a good result than the original, that we can find a better approach to reduce the difference between the test set accuracy and the train set accuracy, by introducing more tuning, we also have thought about some other approaches that we have not consider that may achieve better results without overfitting, such as increasing the architecture layers by increasing the number of convolution layers of the model, or tuning the maxpooling layers, we can also try other activation functions, even **relu** has proved high accuracy for most of the similar problems, or we could also increase the kernel size trying (7*7) kernels or may be bigger, all these approaches are valid and may achieve higher performance.

5. References

- Deep learning (Adaptive Computation and Machine Learning) for Ian Goodfellow, Yoshua Bengio, Aaron Courville.
- https://machinelearningmastery.com/category/deep-learning/

[1] our final architecture has more than 6 million parameters to train compared with only 1 million parameters in the original architecture, that 's why we have trained it for 75 epochs, to reduce the training time.