# CSE 569: Fundamentals of Statistical Learning and Pattern Recognition

Report on Project Part-3: Deep Learning with CNN

by Rohan Sambidi

December 9, 2022

## Introduction

The aim of this project is to perform a classification task using CNN and understand how the choice of hyper-parameters affects the performance of the model. The classification is performed on the samples form the *CIFAR 10* dataset. This dataset has 60,000 samples from 10 classes (6,000 samples per class), where each sample is a color image of size 32x32. There are 50000 training images and 10000 test images. A baseline code for classification using CNN is provided. This code is written in Python, and can be run on Google Colab. We are to experiment with this code by changing various hyper-parameters of the CNN architecture and observe the changes in the performance of the model.

The provided baseline code uses Keras API in Python to build the CNN. The CIFAR 10 dataset is also loaded onto the Google Colab console through this API. Once the data is loaded, it is pre-processed. All the values of the samples (i.e., all the pixels in the image) are converted to floating-point data-type, and then they are normalized. Then, the class vectors, i.e., labels of the samples, are converted into a binary class matrix.  Thus, a sample's feature is a 32x32x3 matrix and its label is a 1x1x10 binary matrix. After the pre-processing is completed, a CNN model is built.

Since CNNs have one input and output for every layer, and all the layers are "linearly stacked", an object, which will be the model that's built for classification, of the *Sequential* class is created. The CNN model built in the baseline code has twelve layers, followed by a dense (fully-connected) layer. This CNN has six convolutional layers in total, each of which is followed by a ReLU activation layer and a batch normalization layer. The convolution is performed with no padding; so, the output of a convolutional layer will be of the same size as its input. Each layer in the model has a different set of hyper-parameters. After the model is built, it is compiled with cross-entropy loss and Adam optimizer. The learning rate was set to 0.01. After the model is complied, it is fitted (trained) on the training samples. It is trained for 50 epochs with a batch size of 64. Once the training is completed, it is tested on the test samples for evaluation. The test loss and accuracy are reported.

## Method

A series of tasks were given, where each task specifies which hyper-parameter modification needs to be done. The tasks were:

a. Change learning rate to— i) 0.05 ii) 0.0001
b. Change kernel size for first convolutional layer to 7x7
c. Change optimizer to RMSProp
d. Remove all the batch normalization layers in the network
e. Change all dropout values (units to drop)— i) 0.7
f. Change batch size— i) 16 ii) 256

To perform these tasks, I simply have to change the corresponding lines in the code. The respective changes in the performance (test loss and test accuracy) are to be reported.

## Results

| Task | Hyper-parameter setting | Test loss | Test accuracy |
|---|---|---|---|
| 2 | Baseline model | 0.588895320892334 | 0.837199985980988 |
| 3a— i) | Learning rate = 0.05 | 1.67247807979584 | 0.79009997844696 |
| 3a— ii) | Learning rate = 0.0001 | 0.52049195766449 | 0.837300002574921 |
| 3b | kernel size for first convolution layer = 7x7 | 0.576073527336121 | 0.83490002155304 |
| 3c | optimizer = RMSProp | 0.634817898273468 | 0.82779997587204 |
| 3d | Without batch normalization layers | 2.30307197570801 | 0.100000001490116 |
| 3e | All dropout values = 0.7 | 0.700871229171753 | 0.765600025653839 |
| 3f— i) | batch size = 16 | 0.688322365283966 | 0.811100006103516 |
| 3f— ii) | batch size = 256 | 0.654051840305328 | 0.819700002670288 |

## Observation

In general, test loss is a kind of distance metric between the actual values and the predicted values. So, low test loss implies that the model made small errors in predicting the label of a sample. In contrast, high test loss means the model made huge errors. On the other hand, a low test accuracy implies that the model made many wrong classifications, and high test accuracy means model made only a few wrong classifications. Thus, a model with high accuracy and low loss will be the best classifier.

The observations for each task are listed below:

- Task 3a:

  Fundamentally, CNN uses stochastic gradient descent algorithm for updating the weights during back-propagation. The learning rate hyper-parameter is used to control how much the weights are updated. It scales the weight updates with an aim to minimize the loss function. Thus, having very low learning rate will lead to slow training progress since the weights updates are done by small magnitude. Alternatively, hight learning rate will lead to drastic updates in weights that may not result in convergence of the loss function.

  From the experiment of changing the learning rate, we can see that the test loss increased when the learning rate was increased. This is because high learning rate led to bigger updates of weights that resulted in divergent behavior of the model. On the other hand, when the learning rate was decreased, the loss decreased as well. This is because the gradient took smaller steps towards a minimum, resulting in a better convergence. However, this training will take longer since smaller updates takes longer time to find a minimum. It can be seen that there was no significant change in accuracy when the learning rate was decreased, but there was some drop in the accuracy when the learning rate was increased. This depend on the dataset and weight initializations. In a different scenario, there may be an increase in accuracy when the learning rate in low and there may not be a significant decrease in accuracy when the learning rate is high.

- Task 3b:

  Kernel size refers to the dimensions of the kernel (or filter). An nxn kernel is basically an nxn matrix. The convolution operation is simply the dot product between the kernel and an image patch of same size. Thus, bigger kernel implies more operations (multiplications and additions) and more weights. Having more weights would make the model learn even the subtle patterns in the image which might result in overfitting. This will drop the test accuracy. This is probably why the accuracy dropped when the kernel size was increased to 7x7. However, this hyper-parameter change was done only on the first layer, hence the drop was not very significant. Changing kernel sizes in all the convolution layers to 7x7 might lead to a significant drop in the accuracy. We can notice that the loss decreased by a little magnitude. This might have happened due to the choice of data and initial weights. This might not be the case all the time.

  Another reason could be that since we assume low-level features in the image are local, having large kernels would result in non-local pixels being considered as well for training/ leaning about the low-level features. The first convolutional layer will recognize basic feature like edges. Having a bigger kernel would make it hard for the model to focus on one locality to recognize edges. This might cause the model to make small errors while recognizing certain features in an image.

- Task 3c:

  Both Adam (Adaptive Moment Estimation) and RMSProp (Root Mean Square Propagation) optimizers are first-order gradient-descent-based optimization techniques, i.e., they use the first derivates of the loss function to minimize it. RMSProp has the ability to adapt the gradients in different directions. It uses the sum of gradient squared along with a decay factor. This decay factor stresses on the recent gradients rather than on the all the previous gradients, and also scales the sum of gradient squared which will help in faster descent of the gradient. Adam uses, in a sense, both the properties of RMSProp and Momentum. It uses sum of gradient (from Momentum) and the sum of gradient squared (from RMSProp) to measure the change in gradient. Thus it has the ability to escape local minima and adapt to the direction of descent. This may be the reason for the baseline model to perform slightly better than the model in this experiment. There might have been a local minima into which RMSProp might have fallen, resulting in poor performance when compared to the baseline model.

- Task 3d:
  While training a CNN, the weights are updated backwards, layer-by-layer, assuming that the prior layer parameters are fixed. But, since all the layers are changed after an update, the layers lack standardization. This will impact the training speed since the distribution of the weights that was assumed to be fixed will be changed in the next update. Batch normalization standardizes the outputs of a layer, i.e., it re-scales (standard deviation to 1) and re-centers (mean to 0) the activations. This is similar to normalizing the dataset before giving it to the classifier. Normalized data helps only the first layer of the model. Similarly, normalization of the activations will help the hidden layers, which change constantly, to coordinate better. Removing this normalization step would give unnecessary emphasis on less-important features. Since the outputs of each layer might follow different distributions, the training would not be perfect. This is why the model's performance was terrible when the batch normalization was removed. It would be better to have all the outputs of all the layers measured on the same scale. This would ease the training process, resulting in better performance.

- Task 3e:
  Dropout layer is used to prevent overfitting and improve the generalization ability of a model. They add noise (i.e., set a node to 0) to the output feature maps of each layer. This will improve the robustness of a model towards variations of images. The probability with which a node is set is 0 is given by the hyper-parameter called dropout value. Thus, higher dropout values implies more chances for a node to be set to 0. So, all the information in that node will be lost. This means, higher dropout value would lead to too much noise, obscuring the information needed for the model. This will lead to poor understanding of the features by the model, resulting in poor performance. This could be the reason for a significant drop in test accuracy, and an increase in loss, in this experiment. Since all the dropout layers had a dropout value of 0.7 (which is too high), most of the nodes are set to zero resulting loss of details for the model to recognize. Thus, the model performed poorly when compared to the baseline model.

- Task 3f:

  The batch size defines the number of samples that will be propagated through the network. If the batch size is n, each iteration uses n samples for training. For first iteration, it uses the first n samples; for second iteration, it uses the second n samples; and so on. Therefore, lower batch size implies more number of iterations per epoch. This means the weights are updated many times while training. Updating the weights too frequently would not lead to stable learning. Too many updates would, in a sense, constantly shift the model's focus on recognizing the patterns. This could be the reason for the model lower batch size results in a drop in performance. This is also why the loss increased— the model was making bigger magnitude of errors in predicting the labels.

  We can also see that the model with higher batch size also performed poor, both in terms of loss and accuracy, when compared to the baseline model. This is probably because higher batch size would lead to poorer generalization ability of the model. When the batch size is high, the number of iterations will be low, and thus, the frequency of updating the weights will be low. Giving too many samples at a time for training would be overwhelming of the model and it would get stuck in a local minimum.

## Conclusion

From this project, I have learned how the choice of various hyper-parameters will affect the training and performance of a CNN. One of the important observation I made was how important it is to choose the right set of hyper-parameters. The CNN is very sensitive towards a few hyper-parameters, whose change in small magnitudes might lead to significant change in the performance. Hyper-parameters like learning rate and batch size should be chosen wisely. Too high or too low would result in poor performance. A model's ability to recognize local features depend on the size of the kernel as well. Bigger kernel size will have too many pixels to read and the model might also recognize unwanted low-level features in an image. In some cases, the affect of a hyper-parameter would be significant depending on the data and initialization of weights. Batch normalization seemed to be very important for classification using CNN. Without this normalization, the performance was the worst. Hence, it is important to normalize the outputs of layers for every batch. Hyper-parameters like dropout value needs to be chosen in a way that it can prevent the model from overfitting while also keeping most the information in the nodes to help generalization. Choosing the right optimizer is also important. Optimizers like Adam combine the pros of multiple gradient-based algorithms and hence could be a better choice.