Deep Learning | Mini-Project

Yashika Khurana (yk2773), Raj Oza (ro2151), Amey Kolhe (apk9563)
New York University
yk2773@nyu.edu, ro2151@nyu.edu, apk9563@nyu.edu

https://github.com/khuranayashika31/DL_mini_project

Overview

The objective of our mini-project is to design a modified ResNet architecture for image classification on the CIFAR-10 dataset, under the parameter budget constraint of 5 million. To achieve this goal, we experimented with various design choices (explained at length in the Methodology section) and came up with a modified version of the ResNet-18 architecture, with 4 residual layers, with 1 block per layer. The model performs fairly well after data augmentation and learning rate scheduling and weight decay, rendering a test accuracy of 91%, while the parameter count is 4903242, i.e., around 4.9 million, within the provided limit. Our codebase is available here.

Methodology

This section discusses our observations (pros & cons of choices) and inference (lessons learned) from the various design and architecture choices we made. We have used [1] as a reference research paper for our study.

Data pre-processing & augmentation
The CIFAR-10 dataset consists of 60k 32x32 colored images in 10 classes(airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck). The dataset is segregated into 5 training batches and 1 test batch, containing10k images each. We used 80:20 split for training & testing respectively. We normalized both- training and testing data. However, data augmentation is applied to the training data using torchvision transform. Initially, we applied the following transformations:
Random cropping with a padding of 4 pixels
Random horizontal flipping
On testing, we observed an accuracy of 89%.
So, we added
Random rotation (between an angle of 0 and 30 degrees).

This boosted the test accuracy by 1%, making it 90%.

We visualized the data to evaluate the transformations applied successfully.

Inference
Data augmentation helps introduce diversity to the dataset by helping it learn more robust features and reduce overfitting, by leveraging randomness. This helps improve the model performance.

Network Architecture
We modified the ResNet-18 architecture. Our model comprises of 12 deep layers. The model architecture is as follows:

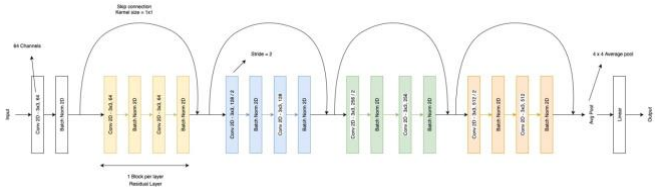

Figure 1: Network architecture

Each layer is made up of blocks, where each residual block has convolution, batch normalization layers and skip connections. Initially, we used 2 basic blocks (as implemented in code) in each layer. With this choice, we observed an accuracy of 91%. However, the number of parameters shot up to 6.7k, which was over-budget. Thus, we limited our model to include 1 basic block per layer.

Inference
Increasing model depth increases the model's capacity and complexity to learn more features from the data. Discriminative data representation allows the model to learn more intricate patterns that helps ameliorate classification performance. However, increasing model depth typically results in an increase in the number of parameters, as

reported in the observation above. Thus, a good strategy is to understand the requirements and find a suitable trade-off.

Loss function & optimizer

We tried Stochastic Gradient Descent, Adam and RMS Prop as optimizers. However, we observed best results with Stochastic Gradient Descent with a momentum value of 0.9, using a weight decay of 0.0001. Our loss function is CrossEntropyLoss.

Inference

During training, weight decay prevents the model from over-fitting by introducing a regularization term to the loss function. By penalizing concept, it helps the model learn robust features and generalize well. Momentum also helps improve model training by taking into consideration the gradient knowledge from previous iterations. This strategy helps the optimizer to remember the direction of previous updates, allowing it to determine the current step size accordingly. This typically leads to an increased optimization performance.

Training

We tried varying the number of epochs and gathered the following observation:

No. of epochs	Testing accuracy(%)
50	89
70	90
100	90.5
108	91

We train our model on Google Colab Tesla K80 GPU(provided free)on the training data and used the validation set to check its validation loss. We save the model only if there is an observed decrement in the validation loss. When the training is complete, we load the best saved model for testing on the test set. We also adjust the learning rate of the optimizer during training by using learning rate scheduler, by setting the validation loss as the criterion.

We used torchviz library to construct a graph of our model. (refer to the pdf file in the Github repository: Digraph.gv.pdf)

Inference

Increasing the number of epochs generally leads to an improved model performance, as observed in our case. This can be attributed to learning of useful data representations, by going through the data multiple times, and thus better convergence. However, increasing the number of epochs leads to an increase in computation time and it also increases the risk of overfitting, if the model memorizes the data. Thus, it is a good practice to increase data diversity and use preventive methods such as weight decay(as described before).

Results

Our model has the following layers and corresponding number of parameters (generated using torchsummary):

```
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           1,728
       BatchNorm2d-2           [-1, 64, 32, 32]             128
            Conv2d-3           [-1, 64, 32, 32]          36,864
       BatchNorm2d-4           [-1, 64, 32, 32]             128
            Conv2d-5           [-1, 64, 32, 32]          36,864
       BatchNorm2d-6           [-1, 64, 32, 32]             128
        BasicBlock-7           [-1, 64, 32, 32]               0
            Conv2d-8          [-1, 128, 16, 16]          73,728
       BatchNorm2d-9          [-1, 128, 16, 16]             256
           Conv2d-10          [-1, 128, 16, 16]         147,456
      BatchNorm2d-11          [-1, 128, 16, 16]             256
           Conv2d-12          [-1, 128, 16, 16]           8,192
      BatchNorm2d-13          [-1, 128, 16, 16]             256
       BasicBlock-14          [-1, 128, 16, 16]               0
           Conv2d-15            [-1, 256, 8, 8]         294,912
      BatchNorm2d-16            [-1, 256, 8, 8]             512
           Conv2d-17            [-1, 256, 8, 8]         589,824
      BatchNorm2d-18            [-1, 256, 8, 8]             512
           Conv2d-19            [-1, 256, 8, 8]          32,768
      BatchNorm2d-20            [-1, 256, 8, 8]             512
       BasicBlock-21            [-1, 256, 8, 8]               0
           Conv2d-22            [-1, 512, 4, 4]       1,179,648
      BatchNorm2d-23            [-1, 512, 4, 4]           1,024
           Conv2d-24            [-1, 512, 4, 4]       2,359,296
      BatchNorm2d-25            [-1, 512, 4, 4]           1,024
           Conv2d-26            [-1, 512, 4, 4]         131,072
      BatchNorm2d-27            [-1, 512, 4, 4]           1,024
       BasicBlock-28            [-1, 512, 4, 4]               0
           Linear-29                  [-1, 10]           5,130
           ResNet-30                  [-1, 10]               0
================================================================
Total params: 4,903,242
Trainable params: 4,903,242
```

Figure 2: Model layers & parameters

We achieved a test accuracy of 91% and the following class-wise results:

Class	Correct predictions (out of 1000)
airplane	921
automobile	957
bird	869
cat	825
deer	927
dog	863
frog	946
horse	918
ship	956
truck	937

The following plot depicts how the training loss reduces over epochs.

Figure 3: no. of epochs vs training loss

The following plot depicts the reduction in validation loss over number epochs.

Figure 4: no. of epochs vs. validation loss

Summarizing our methodology, we obtained a test accuracy of 91% by using our variant of ResNet-18 model with 1 basic block per residual layer (as depicted in figure 1), with data augmentation including random cropping, random horizontal flipping and random rotation. We used Stochastic Gradient Descent optimizer and CrossEntropyLoss, with weight decay. We train our model for 108 epochs by making use of the learning rate scheduler with validation loss as the criterion, maintaining, 4903242 parameters, i.e. under 5 million parameter budget.

References

[1] Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun Deep Residual Learning for Image Recognition

[2] Tien Ho-Phuoc CIFAR10 to Compare Visual Recognition Performance between Deep Neural Networks and Humans