ECE6953 - Mini Project Report

Meet Udeshi (mdu2004), Zeng Wang(zw3464), Sara Ghazanfari(sg7457)

Abstract

The paper presents our experiments with the ResNet18 model trained on the CIFAR-10 dataset. We show the impact of different values of hyper-parameters and design choices, namely weight decay, data augmentation, optimizer, batch size, and learning rate scheduler. Based on the experiments, we determine that a deeper model is needed to reach higher accuracy, and add another layer in the ResNet18. We also use MaxPool in the skip connections to ensure no features are dropped due to down-sampling. Altogether, our model[1] uploaded in the GitHub achieves **91.01%** test accuracy.

Introduction

We started our investigation by loading the original ResNet18 model. As it has over 11 million parameters, we tried to engineer the filter numbers and sizes in each layer to decrease the model parameters to 5 million. After doing investigations, we chose 64, 192, 192, 256 as the number of filters in each layer respectively. The filter sizes are 3×3 for all the layers. In the next part we will discuss the experiments that were done to choose any of the hyper-parameters and finally learning the out-performed model.

Choosing Hyper-Parameters

In the process of training a model, we should always choose the model's hyper-parameters. As choosing the hyper-parameters is a step of training, we are using the validation dataset to learn our hyper-parameters and then report the final accuracy of the model using the test dataset. The hyper-parameters that are tuned in our work include the batch size, the data augmentation technique, the regularizer, gradient clipping, the optimizer, the depth level of the model and finally learning rate scheduler. In each section, we are mentioning the values that we have considered for the hyper-parameter and the accuracy of the model given each of the values. The hyper-parameter chosen in each part is used in the next parts, i.e. the batch size chosen in the first part is used in the rest of the experiments. The number of epochs for experiments with the four-layer model is 50 and for the five-layer model is 100.

[1] https://github.com/Sara-ghazanfari/DL_Miniproject.git

Batch size

This part will explore the impact of batch size on the training of our model. The batch size defines the number of samples that will be propagated through the network, which could be one of three options: batch mode, mini-batch mode and stochastic mode (Validated(2015)). Generally, mini-batch mode, a smaller batch size than the total sample count is always selected. Utilizing a mini-batch size not only aids in reducing memory consumption during training, which is crucial when the entire dataset cannot fit into the system's memory, but also accelerates the network training process. This is due to the fact that weight updates occur after each propagation when using mini-batches. However, a smaller batch size may result in a less accurate estimation of the gradient. Consequently, selecting an optimal batch size for our network training process is essential. Common mini-batch sizes include 64, 128, 256, and 512. Given the substantial size of our dataset, we will specifically evaluate the effects of using batch sizes of 128, 256, and 512 in our training. As we shown our batch size effect comparison experiment in the Figure1, we could know the 256 batch size should be our optimal choice.

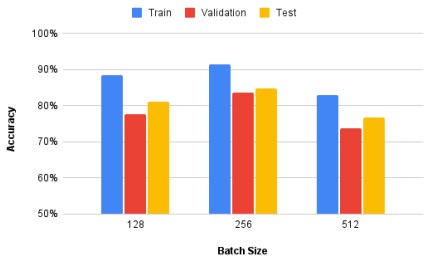

Figure 1: Batch size effect on model accuracy. Among the three values for the batch size, 256 has the best performance.

Data Augmentation

Data augmentation is the process of modifying the training dataset to add noise and randomness. This helps the model

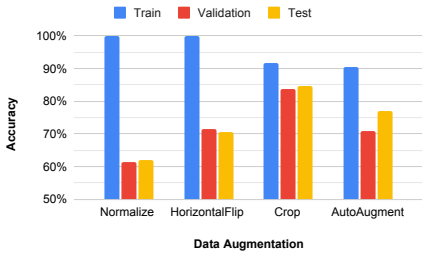

Figure 2: Impact of data augmentation of the training set on the accuracy

learn to counter this additional randomness and it can avoid overfitting to training data. We try four different data augmentation techniques, described as follows:

- *Normalize*: Subtract mean and divide by standard deviation of the entire CIFAR-10 dataset to normalize the data. This does not perform any random operation, and is used as a baseline for no augmentation.
- *HorizontalFlip*: Randomly perform a horizontal flip. Horizontally flipped images should also be classified as the same class.
- *Crop*: Randomly crop a section of the image. These cropped images would lose some information about the full image, but the parameters of cropping are such that most of the image should be included, and the model should be able to classify these images.
- *AutoAugment*: Perform auto augmentations recommended by Pytorch for CIFAR-10 (Pytorch(2023)).

These augmentation techniques are added on top of each other. For each we first perform *Normalize*. For *Crop*, we also perform *HorizontalFlip*.

Figure 2 shows the accuracy for each augmentation technique. Looking at training versus validation accuracy of *Normalize* and *HorizontalFlip*, we see clearly that the model has overfit to get 100% training accuracy but perform poorly on the validation and test sets. This is because both these augmentation techniques do not add sufficient randomness to the dataset. Looking at *Crop*, we can see that the model performs much better on the validation and test set to get accuracy of $\tilde{8}3\%$, and does not overfit on the training set. The *AutoAugment* model also does not overfit and performs well on the validation and test set, but it cannot outperform the *Crop* model. We select *Crop* for data augmentation based on the highest validation accuracy.

Regularizer

In this part, we want to analyze the effect of having weight decay as a regularizer to our model:

$$\mathcal{L}_R(\omega) = \mathcal{L}(\omega) + \lambda||\omega||_2^2$$

As mentioned in the regularized loss function, the weight decay term has a coefficient that balances the effect of the regularizer and the model's primary loss function. Therefore there is a trade-off here between the generalization and accuracy of the model. We have trained our model using three values for $\lambda = 0, 10^{-4}$ and 5×10^{-4}.

Figure 3: Weight decay effect on model accuracy. Among the three values for the weight decay coefficient, 0.0001 has the best performance.

In figure 3 we can observe the trade-off explained. for $\lambda = 0$, the model is overfitted to the training data and the results on validation and test datasets are not acceptable. For $\lambda = 5 \times 10^{-4}$, the loss function is forcing the model to have small weights and preventing the model from learning the optimized weights. As a result of this experiment, $\lambda = 10^{-4}$ is used for upcomming experiments.

Gradient Clipping

Gradient clipping is a technique to clip the gradients to a maximum value during gradient descent. This prevents the gradient descent from being affected by exploding gradients. Equation 1 shows the equation for gradient clipping, where $\mathbf{x}$ is the gradient vector and c is the clipping hyperparameter. Figure 4 shows the accuracy for models trained without gradient clipping and with clipping value $c = 0.1$. The results show miniscule improvement in validation accuracy with gradient clipping. This is because ResNets are resilient to exploding gradients due to skip connections which back-propagate the gradients directly and batch normalization which normalizes the activations. We go ahead with gradient clipping with $c = 0.1$ for our model.

$$\text{Clip}(\mathbf{x}) = \begin{cases} c \cdot \frac{x}{\|\mathbf{x}\|} & \|\mathbf{x}\| \leq c \\ \mathbf{x} & \text{otherwise} \end{cases} \quad (1)$$

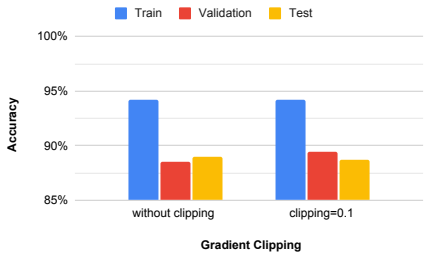

Figure 4: Impact of gradient clipping on accuracy.

Optimizer

This part would discuss two most widely used optimizing techniques, Stochastic Gradient Descent(SGD) and Adaptive Moment Estimation(Adam). Firstly, the concept behind SGD is to approximate the gradient by utilizing a random subset of data points, rather than calculating the full gradient, which would involve all data points in the dataset. Sometimes,employing a basic SGD optimizer may result in sub-optimal performance. To address potential issues, such as getting stuck in local minima, encountering saddle points, or experiencing high curvature, incorporating Momentum is often preferred to enhance the performance of neural networks. For our model training, we take the value of Momentum to be 0.9.

Secondly, the Adam optimizer (Science(2023)) is considered as a method of Stochastic Optimization is a technique implementing adaptive learning rate. It combines the advantages of the two algorithms AdaGrad(adaptive gradient algorithm) and RMSProp(root mean square propagation). AdaGrad's per-parameter learning rate helps increase the learning rate for sparser parameter and works well for sparse gradients. In RMSProp, the learning rate adapts based on the moving average of the magnitudes of the recent gradients. These advantages enable the Adam optimizer to efficiently and effectively tackle a wide range of problems.

Therefore, we will consider to compare the Adam and SGD with Momentum optimizer. From the Figure5, The Adam optimizer performs better than SGD.

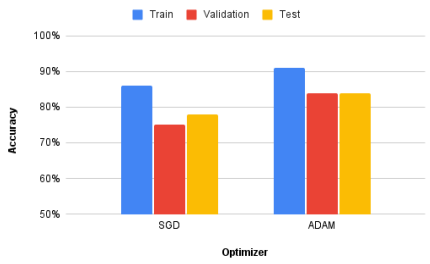

Figure 5: Optimizer effect on model accuracy. Among the two values for the optimizer, Adam has the best performance.

Model Depth

This part will investigate the impact of the different number of layers in our model structure in order to determine the optimal layer for our model. Given the constraint of keeping our model's parameters below 5 million, we would design our model to maximize the usage of available parameters while adhering to this limitation. As we calculate the number of parameter using the `torchsummary.summary` function, we have determined that both 4-layer and 5-layer model structures can be implemented within our constraints. As the Table 1 shown, we list our optimized designed model structure with parameters. Given that our input image size is not large, we will set all kernel sizes to 3x3.

	channel size	parameters
4 layers	[64,192,192,256]	4901962
5 layers	[64,128,144,192,256]	4960234

Table 1: Layer design structure with parameter

As we discussed in above parts, we already fixed all the basic parameters except for the layer number. Therefore, we devised the experiment to explore which models should be our optimized one. The result is shown in the Figure 6. Our 5 layer structure performs better; therefore, we decide to choose it for our final model implementation.

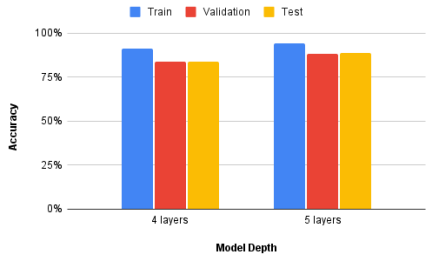

Figure 6: Num of layer effect on model accuracy. Among the two values for the num of layers, 5-layer model has the best performance.

Learning Rate Scheduler

After fixing the depth level of the model, we are going to choose one of the most important hyper-parameters which has a great impact on the learning process, dynamic learning rate. The intuition behind using a dynamic learning rate is that we start from a random point in the optimization landscape and move toward the optima. At first, we need to take bigger steps. However, as we approach the optimum we need to take smaller steps so that we reach the optimum sooner and prevent any oscillations near the optimum. For our evaluation, we have three experiments, without a scheduler, with OneCycleLR, and with CosineAnealingLR.

From figure 8 we can observe that a static learning rate has the worst accuracy. The competition between OneCycleLR, and CosineAnealingLR ends with the victory of OneCycleLR. Because OneCycleLR has annealing and other mechanisms in its algorithm.

Final Model

After doing all of the experiments described above, we have our final model with five layers as shown in figure 7. Our selected batch size is 265. The data augmentation includes normalizing, horizontal flipping, and cropping. Our regularizer is weight decay and gradient clipping is added to control the maximum value the gradients can have. Adam is chosen as the optimizer and the learning rate scheduler is used to have a dynamic learning rate through the learning process. For our final model, we have plotted the loss and accuracy curves in figure 9 and 10 respectively. Due to the complexity

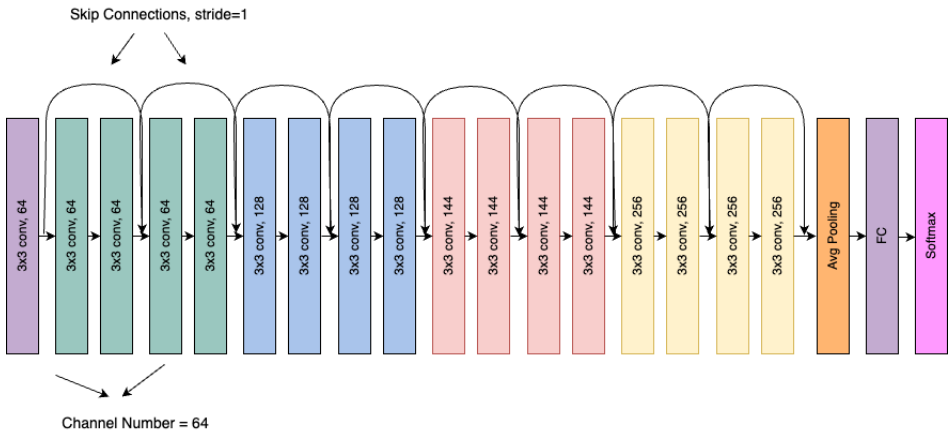

Figure 7: Final Model Architecture. We had two novelties in designing our final model: (1) A deeper model than ResNet18 with 5 layers (2) and Changing the structure of skip connections by adding a max pooling and using stride=1.

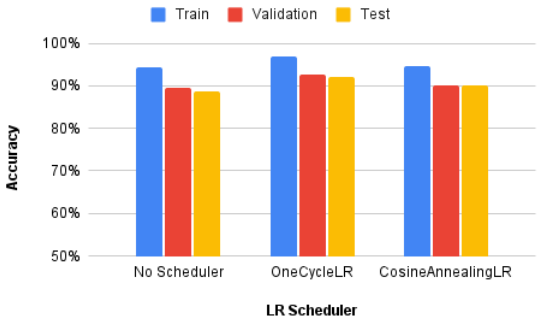

Figure 8: Static vs. dynamic learning rate. As shown a dynamic learning rate is better than a static learning rate and OneCycleLR is outperforming CosineAnealingLR.

of the model, the training process was prone to overfitting. However, After performing data augmentation and weight decay, we are observing that the training and validation loss decreased together during training. Also, the accuracy for both training and validation data increased at the same rate. Therefore, we didn't observe any overfitting and the test accuracy got to **91.01%**. Also, the number of parameter is **4960874**, which is less than the 5 million parameters.

References

[Pytorch(2023)] Pytorch. 2023. Pytorch AutoAugment for CIFAR-10. https://pytorch.org/vision/main/generated/torchvision.transforms.AutoAugment.html.

[Science(2023)] Science, T. D. 2023. Complete Guide to Adam Optimization. https://towardsdatascience.com/complete-guide-to-adam-\optimization-1e5f29532c3d.

[Validated(2015)] Validated, C. 2015. What is batch size in neural network? https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network.

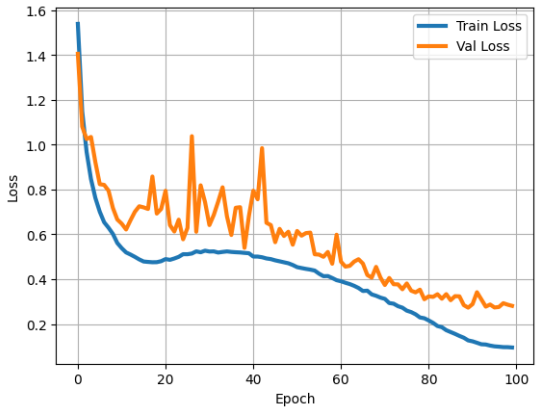

Figure 9: Loss curve with respect to the number of epochs.

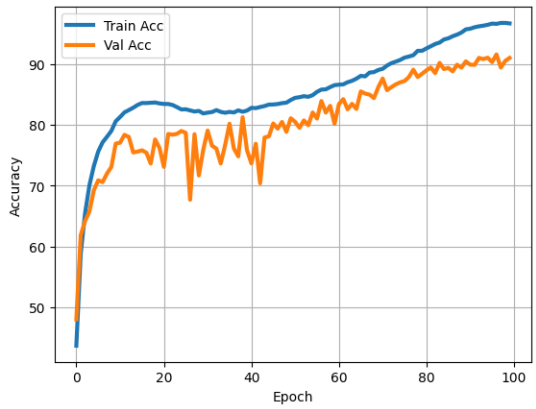

Figure 10: Accuracy curve with respect to the number of epochs.