Report for PJ2

20307130030 Shen Jianzhi

2023.5.17

1 Task

In this project, we learn to build a CNN classifier on CIFAR-10 image dataset. First we will test some basic hyper-parameters like depth, channel, different Res-blocks, optimizers, activations, on a toy model. The explanation for batch-normalization is also included there. Second we will try some advanced methods like cycle learning rate to train a light model with decent test accuracy in the least possible time. Third we will train a larger model with regularization like Cutout, Mixup, Label Smoothing, et al., to achieve the best performance.

2 Result

0.92 in 0:02:27 (12 epochs) on 1080Ti, 9 layers, 6,573,130 parameters, using cycle learning rate

0.9565 in 150 epochs, 18 layers, 11,173,962 parameters, using Cutout, label smoothing loss, improved downsampling block, Mish activation, mixed optimizer and cosine annealing warm up learning rate schedule

0.9489 in 80 epochs, 12 layers, 7,827,786 parameters, using Cutout, label smoothing loss and Ranger optimizer

0.9127 in 80 epochs, 59 ConvLayers, 614,850 parameters, using dense connection

3 Data

The dataset is CIFAR-10 image dataset. It contains 50,000 in training set and 10,000 in test set 32x32 RGB images with ten classes including airplanes, automobiles, birds, cats, deers, dogs, frogs, horses, ships and trucks. By now, the fastest training towards 94% accuracy is Santiago Akle Serrano's Custom ResNet 9, in 0:00:10, according to Stanford's DAWNBench, which uses cycle learning rate, ghost batch-normalization and parallel implementation and is trained on V100. The non pre-trained CNN-based highest accuracy, 98.3%, according to Paper with Code CIFAR-10 Benchmark comes from ResNet50[1], which used novel training recipes like large-batch LAMB. For lack of training resources, we just try our best to get close to the two.

4 Test on Different Hyper-parameters

Warning: some results are derived with toy models of depth no more than 10, and may not apply to larger models. If that is the case, there will be supplementary explanations.

4.1 Kernel Size

By increasing the range of receptive field, the network learns to identify larger patterns. To achieve that, a CNN network need to either enlarge the kernel in its layers or increase the depth. The former, however, is more costly in parameters. Therefore, traditionally we get CNN deeper with 3x3 kernels, though some new models like DenseNet[2], larger datasets and higher computing power allows and may work better with larger kernels.

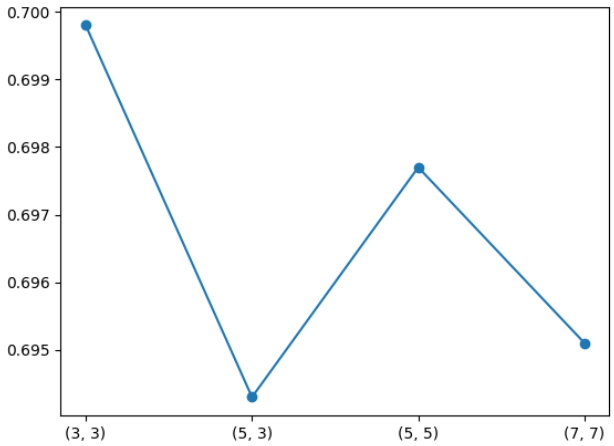

Figure 1: Accuracy of Different Kernel Size Combinations with a 2-ConvLayer CNN

As shown in Figure 1, increased kernel size does not improve the toy model steadily.

Big (7x7) kernels are visualized in Figure 2. Some kernels have learned some particular edge or color patterns.

4.2 Number of Channels

Number of channels decides how many patterns a CNN can distinguish. As the number of channels rises, at first the model's performance goes up since the representative capability rises, but later it seizes to improve and even get worse due to overfitting. The pattern is clearly shown in Figure 3.

Traditionally, the number of channels increases layer by layer, because deeper layer receive larger patterns, which have more variations.

4.3 Depth

Depth also determines the representative capability of the network. However, apart from overfitting, deeper nets are also hurdled by gradient vanishing, so that many features extracted in the shallower layers are not fully exploited. As is depicted in Figure4, although the 3-convlayer CNN takes a huge leap, the

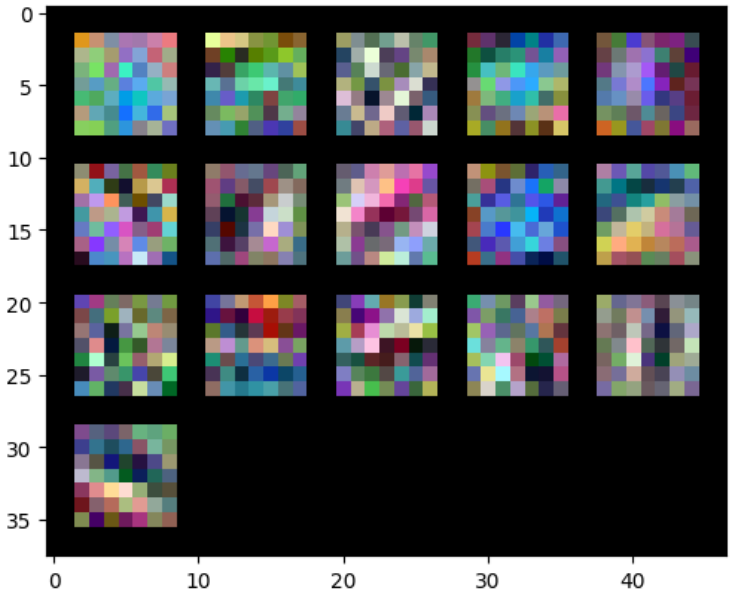

Figure 2: Kernel Visualization

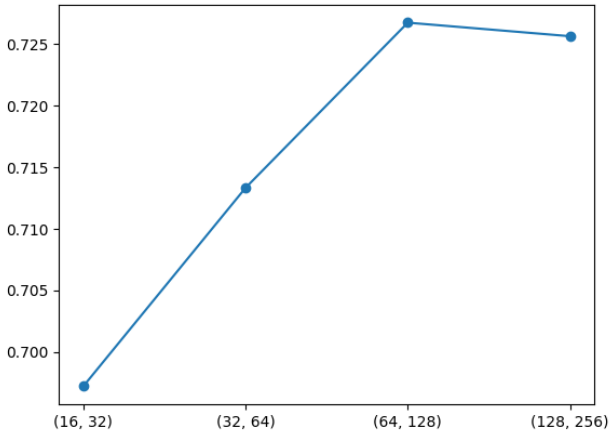

Figure 3: Accuracy of Different Channel Combinations with a 2-ConvLayer CNN

accuracy quickly fall after that. Therefore, we need regularization and different connection structure to fix this.

In order to figure out what hurdles the 5-ConvLayer toy net, we can take a look at the relative gradient norm, i.e. the Frobenius norm of the gradient divided by the norm of the parameter. As can be seen in 5, although gradient vanishing does not happen, the relative gradient norm is significantly different and the effect is accumulated through layers. Internal Covariate Shift[3] happens, which means the output of each layer is distinctively distributed, so the gradient update may be unbalanced, thus making the training unstable.

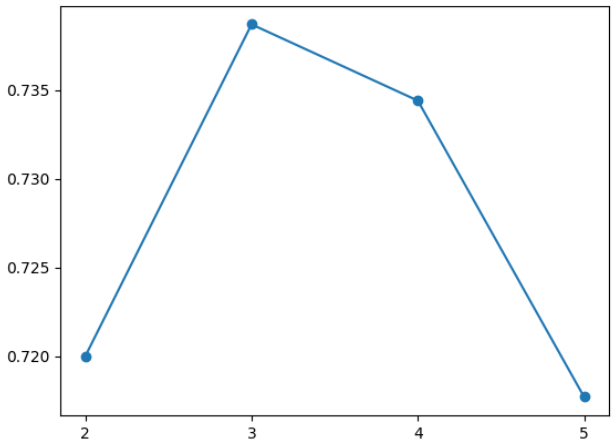

Figure 4: Accuracy of Different Depth

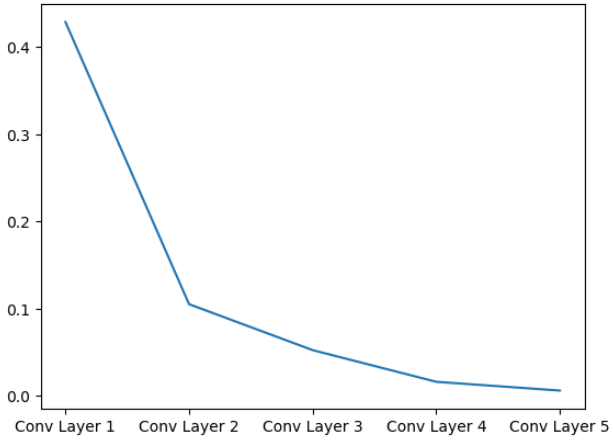

Figure 5: Relative Gradient Norm of Different Layers

4.4 Loss Functions

Label Smoothing Loss[4] is a regularization strategy. Instead of one-hot label, it assigns a continuous label as Equation 1

$$\hat{L}_i = \begin{cases} \frac{\epsilon}{C-1}, & L_i = 0 \\ 1-\epsilon, & L_i = 1 \end{cases} \tag{1}$$

where L is the original one hot label, C is the number of classes, and ϵ is a smoothing hyper-parameter.

The reason why it acts as regularization is that it does not treat the training data as absolute, because there may be confusing patterns or even mislabelling in the training set. By introducing noise to the training labels, it to some extent prevents learning some bad patterns exclusive to the training set.

On the toy model, the loss function does not improves the model, because the model is too simple to overfit. When the model becomes larger in the preceded experiments, the importance of Label Smoothing Loss emerges.

4.5 Activations

ReLU is by far a default option. By Kaiming initialization[5] the non 0-mean problem is largely fixed. However, there is still a problem with ReLU that once a neuron is updated to a negative value, the neuron gets 'killed' and stop to learn, so ReLU is sensitive to large learning rate. Leaky ReLU can fix this problem by allowing a little gradient on the negative region, where the gradient is a hyper-parameter. PReLU further makes the gradient parameter learnable. ELU and CELU further makes the negative part exponential.

Mish[6] is a lately suggested activation function, as shown in Equation 2. It claims to bring smoother loss landscape in deep models. However, in this project the experiment result is inconsistent with the claims. When training three models with 30, 40, 50 batch-normalized ConvLayers, 0.1 SGD, the models with Mish activation all underwent gradient explosion while those with ReLU did not. However, when training the final model in the project, Mish works better than ReLU (from 94.97% to 95.65%). Maybe this activation function works better with more sophisticated optimizers.

$$Mish(x) = x \tanh(softplus(x)) = x \tanh(\ln(1 + e^x)) \tag{2}$$

The result in Figure 6 shows that Leaky ReLU and CELU are both advisable choices in shallow network. The reason PReLU does not perform better may be that the additional parameters worsen overfitting. And Mish does not perform well in the shallow network.

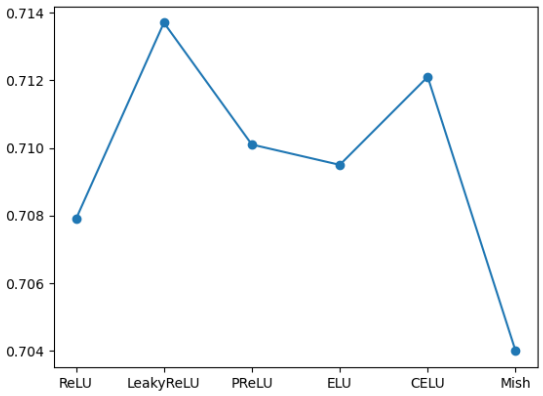

Figure 6: Accuracy of different activation on a 3-ConvLayer CNN

4.6 Batch-normalization

In order to fix the ICS problem, batch-normalization[3] is proposed. However, batch-normalization does not fix the unbalanced learning inside the network, as shown in Figure 7. Instead, according to MIT's work [7], it accelerates and stabilizes learning by making the loss function smoother and the training less sensitive to learning rate, as shown in Figure 8, which has been smoothed and trimmed for better visualization. It helps to prevent gradient explosion and vanishing.

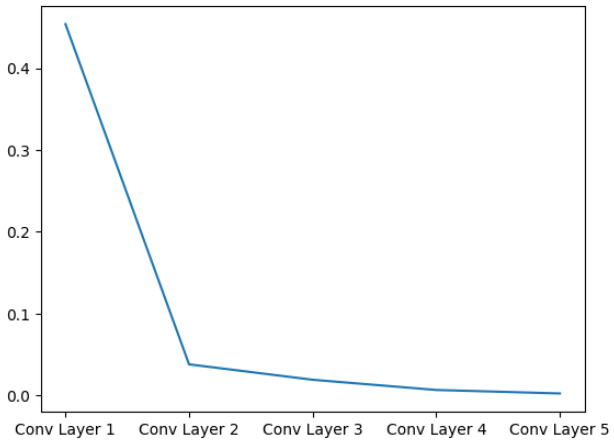

Figure 7: Relative Gradient Norm with BN

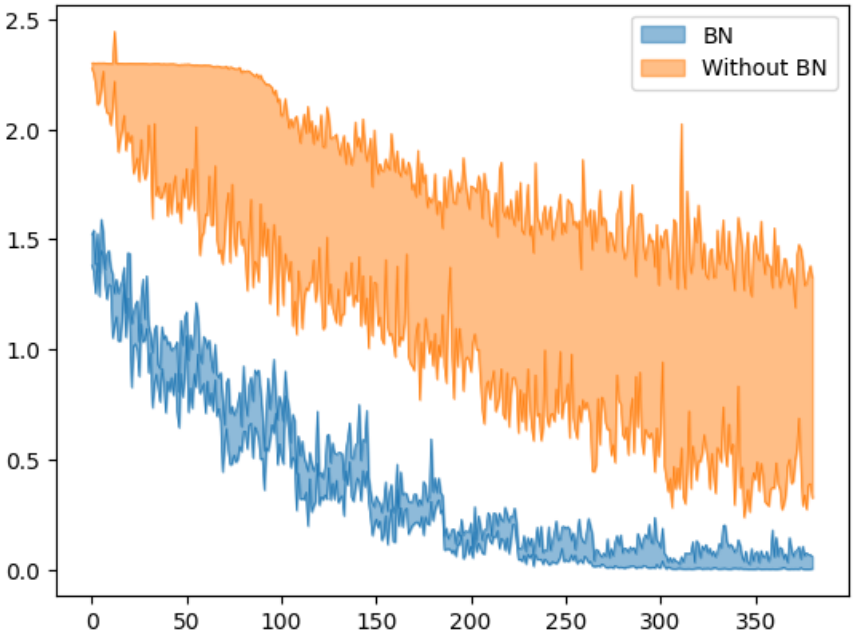

Figure 8: Comparison of Loss Landscape

4.7 Dropout

Dropout does not improve the model very much, compared to other regularization methods. The reason may be a 512-sized input features is compact enough for this task. When the model is compact enough, and the data (after augmentation) is abundant, the optimal dropout rate is no dropout. Cutout, which will be detailed in Section "Data Augmentation", can be interpreted as dropout too, but it is on the first ConvLayer.

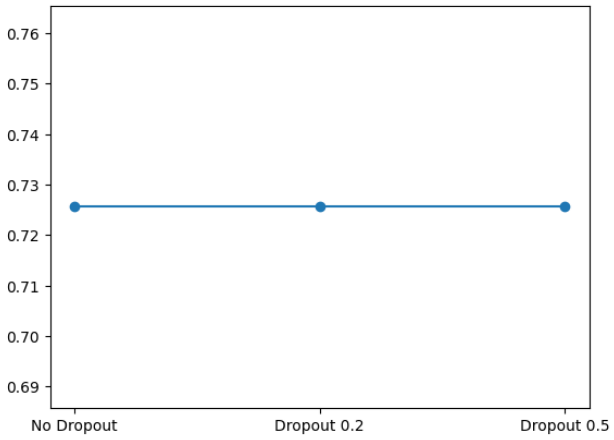

Figure 9: Accuracy when Different Dropout is Assigned to FCLayer

4.8 Residual Connection and Different Res-block

Residual connection [5] allows the net to be deeper free from gradient vanishing, by keeping a main data flow while treating the ConvLayers as additional side tracks, so that every parameter has at least an identity gradient through back-propagation, and the deeper layers can be treated as residual or complementary feature extraction.

After batch-normalization is introduced, the initial ResBlock has multiple alternatives [8], given different order of Conv, BN, activation and addition.

1. Original Block: Conv→BN→ReLU→Conv→BN→addition→ReLU
2. Batch-normalization after addition: Conv→BN→ReLU→Conv→addition→BN→ReLU
3. ReLU before addition: Conv→BN→ReLU→Conv→BN→ReLU→addition
4. ReLU-only Pre-activation: ReLU→Conv→BN→ReLU→Conv→BN→addition
5. Full Pre-activation: BN→ReLU→Conv→BN→ReLU→Conv→addition

As shown in Figure 10, BN after addition and Full Pre-activation converges faster than others. That is slightly inconsistent with the claimed result that BN after addition impedes information propagation since it alters the signal on the main flow, because the paper tested them on ResNet-110 and 164 and ours are far simpler. When testing them on the formal models, Full Pre-activation stands out. It avoid altering the main flow signal, allows negative residual correction, and fully taps the potential of batch-normalization.

4.9 Optimizers and Learning Rate Schedule

SGD with momentum and Adam are default optimizers in CV research. As can be seen in Figure 11, Adam achieves fast early convergence, and SGD is robust in later phases. "MyAdaDelta" is our implementation of AdaDelta with weight decay that do not change bias, and "Ranger21" is an optimizer implementing all the advanced step size choosing strategies.

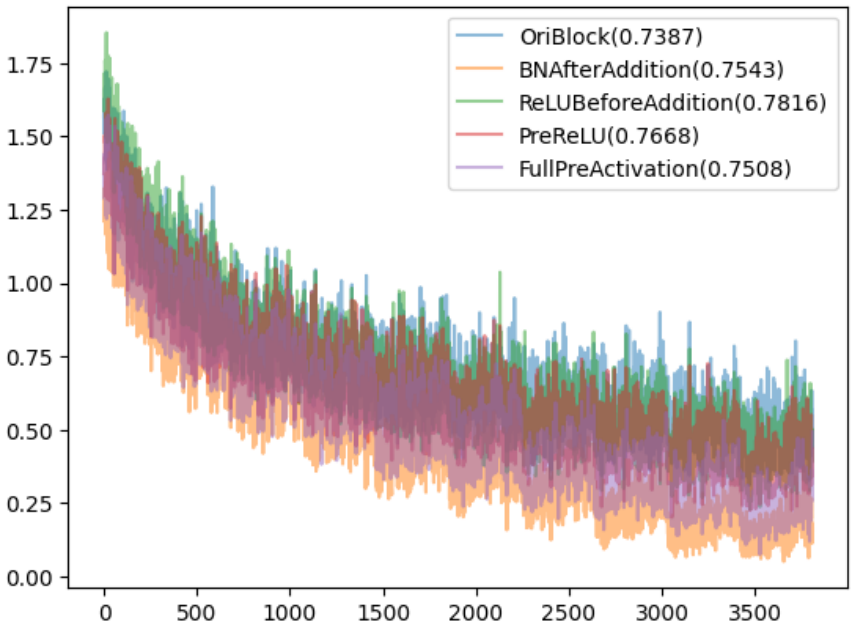

Figure 10: Convergence History with Different ResBlocks

The problem with Adam is that the early estimation of momentum and order-2 momentum is too uncertain (with large variance), and the aggressive step size may trap the model in local optimal. One solution is using warmup, which gradually raise the learning rate at the beginning.

After the parameters come close to the optimal, the learning rate should decay for finer searching. Annealing can be linear, exponential or cosine. The cosine annealing has gained popularity since it decay slower, preserving more chance to avoid local optimal.

Ranger[9] uses Look Ahead[10] mechanism, which tracks two group of parameters, one group of parameters update k steps a time, and the other is led by it as an interpolation. Such update policy is not only robust but also fast when doing fine searching.

As can be seen in Figure 11, in the warmup phase, Ranger learns cautiously and relatively slow, but in the annealing phase it rapidly catches up and converges stably.

4.10 Data Augmentation

Random crop and horizonal flip has regularization effect in this task. Color jitter and random rotation, however, impede the model. The reason maybe that color is a little deterministic for classification in this dataset, and the images are too low-resolution to make rotation sensible.

Cutout, i.e., randomly set a rectangular region of the image to 0, on the other hand, improves the model greatly (from 93.97% to 94.73%). It is a significant expansion of the representation range of the model, as Figure 12 shows. The magnitude of the kernels get enlarged. By covering a part of an image of a class, the network learns to classify based on different parts of the image, rather than focusing on the most significant ones.

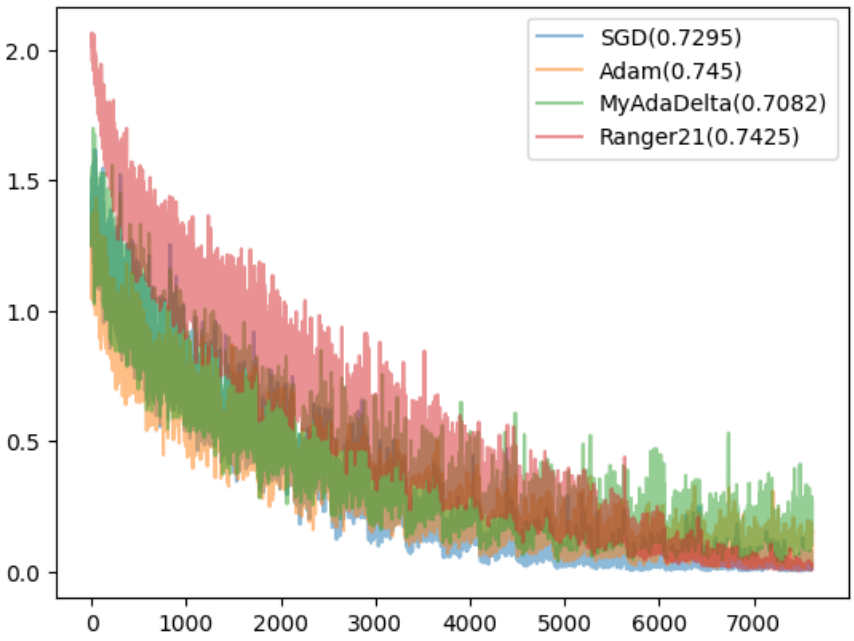

Figure 11: Convergence History with Different Optimizers

5 Train Faster

5.1 Module Augmentation

This is an implementation trick. When doing in-training data augmentation, instead of using Dataloader's transform parameter, which is conducted on CPU, we can pack tensor transforms as a module, so that the transforms can be implemented on GPU, doubling the training speed on average.

5.2 Put MaxPooling Forward

The traditional max pooling ConvLayer has the order Conv→BN→ReLU→Max Pooling. However, since BN and ReLU are monotonical, theoretically the position of max pooling can be shifted forward. After putting maxpooling forward, BN and ReLU just compute on the already compressed channels, thus being more efficient. In the experiment, the training of 10 epochs has been accelerated from 00:02:14 to 00:02:01, and the performance has also improved from 91% to 91.66%. The difference in performance may come from randomness and numerical stability.

5.3 One Cycle Learning Rate

One Cycle Learning Rate[11] has three phases, the first is warmup to a high learning rate, the second is recover to original level, and the third is further decay to 0. The relatively long period on a high level of learning rate acts as regularization. This learning rate scheduler works extremely well with Adam. By using One Cycle Learning Rate, ResNet 9 can converge to 91% accuracy in merely 10 epochs (00:02:07), and has better performance as reported in 12 epochs.

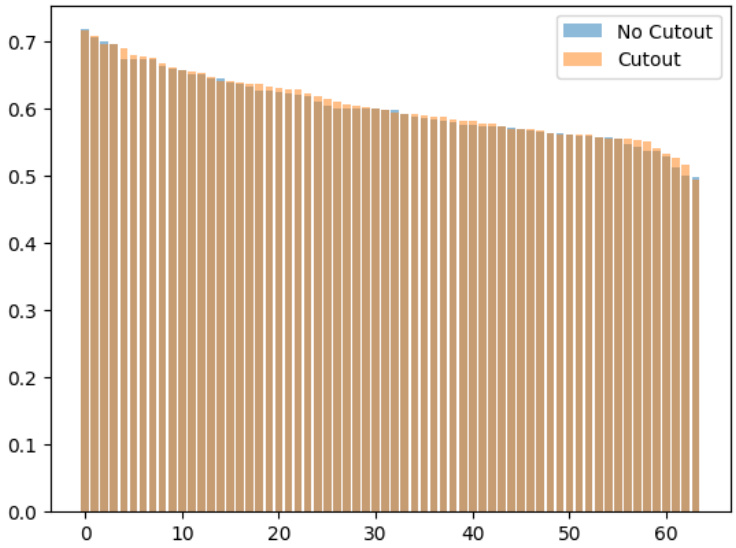

Figure 12: Magnitude of Layer 1 Kernel Weight, Using Cutout or Not

6 Train Better

6.1 Pooling as Downsampling

By replacing the ConvLayer's stride in downsampling ResBlock with Max-pooling, the models performance rises. It is tested on a 15-layer ResNet (Accuracy from 0.9374 to 0.9427) and ResNet18 (Accuracy from 0.9523 to 0.9565), since other models involved do not have a downsampling ResBlock. The reason may be that Max-pooling reserves more important information, while stride may skip some significant signal.

6.2 Mixed Optimizer

Ranger is fast and reliable. However, using Adam based optimizer alone cannot fully push the model to its limit (the highest record achieved by Ranger is 94.97%, with maximal number of epochs 200). Maybe sometimes one cycle learning rate (also implemented in Ranger) may fail to avoid all the local optimal. Therefore, we need more epochs to train a good enough model using Cosine Annealing and SGD. A suggested combination is 75 epochs of Consine Annealing Warmup Learning Rate Adam first, and then 75 epochs of Consine Annealing Learning Rate SGD, which has led to the 95.65% accuracy as reported.

6.3 Potential of DenseNet

The effect of DenseNet[2], which substitute channel concatenation for addition, is also tested. We implemented a light 59-layer DenseNet which only contains 614,850 parameters, after 80 epochs, the accuracy on test has reached 91%. It's worth noting since ResNet 9 has already contained 6,573,130 parameters. However, the GPU Memory is a limitation, since in the deeper layers, the concatenated channels has to be saved as a whole.

References

[1] R. Wightman, H. Touvron, and H. Jégou, "Resnet strikes back: An improved training procedure in timm," 2021.

[2] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2018.

[3] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.

[4] R. Müller, S. Kornblith, and G. Hinton, "When does label smoothing help?," 2020.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[6] D. Misra, "Mish: A self regularized non-monotonic activation function," 2020.

[7] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?," 2019.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," 2016.

[9] L. Wright and N. Demeure, "Ranger21: a synergistic deep learning optimizer," *arXiv preprint arXiv:2106.13731*, 2021.

[10] M. R. Zhang, J. Lucas, G. Hinton, and J. Ba, "Lookahead optimizer: k steps forward, 1 step back," 2019.

[11] L. N. Smith and N. Topin, "Super-convergence: Very fast training of neural networks using large learning rates," 2018.