ECE-GY 7123 – Mini-Project 1

Constrained Residual Neural Network Design

Alaqian Zafar
NYU Tandon
Brooklyn, NY 11201
aaz7118@nyu.edu

Chih-Fan Lin
NYU Tandon
Brooklyn, NY 11201
cfl271@nyu.edu

Katsamapol Petchpol
NYU Tandon
Brooklyn, NY 11201
katsamapol@nyu.edu

Abstract

Residual neural networks (ResNet) have rapidly become one of the most popular architectures in various computer vision tasks such as image classification. However, deep convolutional neural networks need a large number of trainable parameters when compared to the traditional statistical models. In this paper, we developed ResNet architecture to classify the CIFAR-10 dataset at a fixed resource budget while still having a decent performance. Given a 5 million trainable parameter limitation, we provide comprehensive evidence using machine learning techniques to improve our model accuracy. An ensemble of our proposed scaled ResNet architecture and its hyper-parameter settings achieved 95.03% accuracy based on 4,977,226 trainable parameters. Our source code is available at https://github.com/katsamapol/ResNets

1 Introduction

Convolutional Neural Networks (CNNs) have achieved some outstanding results for the fundamental image classification problem, such as when classifying the Canadian Institute for Advanced Research (CIFAR) dataset. Big Transfer (BiT): General Visual Representation Learning, Kolesnikov et al., used 926 million-parameters to achieve 99.37% accuracy [1]. GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism, Huang et al. used 557 million-parameters to achieve 99% accuracy [2]. However, achieving the highest accuracy isn't the most optimal solution if it requires high memory, high computational power, and more time to run. To address this issue, we conducted experiments on a well-known CNN architecture, the residual neural network (ResNet) to find the most efficient training structure that does not exceed 5 million learnable parameters.

1.1 ResNet architecture

Figure 1 shows an image of the ResNet architecture, where N is the number of residual layers, B_i is the number of blocks in residual layer i where each of the residual blocks has two convolutional layers, F_o is a convolutional kernel size of the input layer, P_o is the convolutional padding size of the input layer, F_1 is a convolutional kernel size of every residual layer, P_1 is the convolutional padding size of every residual layer, K is the skip connection kernel size, C_{in} is the number of channels in the input layer, C_i is the number of channels in each of the residual layers.

Figure 2 shows an example of a ResNet formed by the following parameters: Three residual layers, $N = 3$. Two residual blocks in the first residual layer, one residual block in the second residual layer, and two residual blocks in the third residual layer, $B = [2,1,2]$. The convolutional layer kernel size of the input layer is 7, $F_o = 7$. The convolutional padding size of the input layer is 3, $P_o = 3$. The convolutional layer kernel sizes of every residual layer are 3, $F_1 = 3$. The convolutional padding sizes of every residual layer are 1, $P_1 = 1$. The convolutional channel size of the input layer is 64, $C_{in} = 64$. The convolution channel sizes of the residual layers are calculated by the input layer channel, $C_{in} \times (i-1)$; i.e., $C_1 = 64$, $C_2 = 128$, $C_3 = 256$. The skip connection kernel sizes of every layer are 1, $K = 1$.

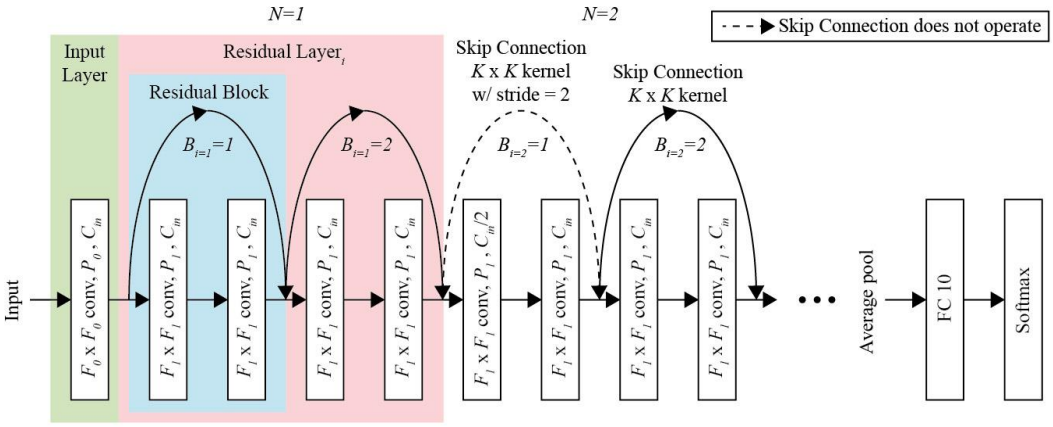


Figure 1: The ResNet Architecture

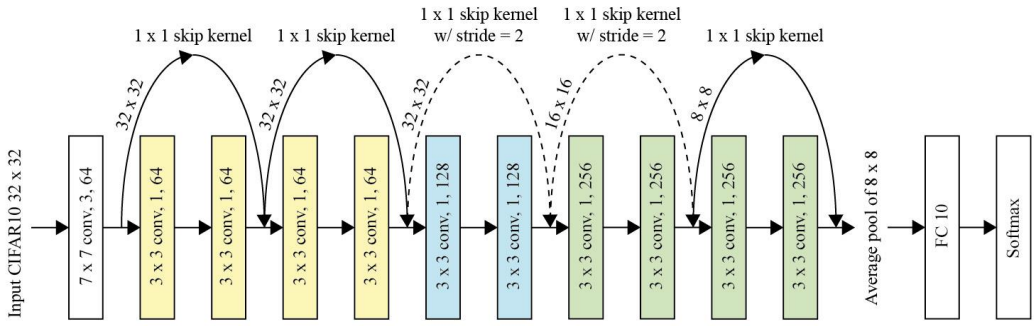


Figure 2: An example of a ResNet architecture with the parameters specified in 1.1

2 Methodology

Since there are many ResNet architectures and combinations of hyper-parameters that can affect the performance and accuracy of the neural network, our first step was to determine the best performing ResNet structure that would generalize well on an unseen dataset.

2.1 ResNet structure parameter setting

Table 1 shows the nine-combinations of residual blocks, residual layers, input channels, kernel sizes, and padding sizes we used to train and test our model. All structures satisfy the 5 million parameter constraint while preserving the spatial dimension of the CIFAR-10 dataset.

There are two main factors that influence how we found the best architecture of the ResNet model. First, because many of the top-performing neural networks have a large number of parameters [3], we want to have an architecture with as many trainable parameters as possible while still being within the parameter constraints. Second, one of the main advantages of the

ResNet architecture is the ability to train and achieve high accuracy in deep residual layers. Additionally, we also want to maintain the number of residual layers as much as it is possible. With these two ideas in mind, we found that the closest ResNet architecture that meets the limitations we have is the ResNet-18.

We designed the first five structures such that they preserved all four residual layers according to the ResNet-18 architecture. However, the residual blocks of the residual layers 2, 3 and 4 are reduced in order to stay within the parameter constraint. Next, the input channels of the first five structures are not changed since increasing the input channels will also increase the channels of the subsequent residual layers, which will exceed the parameter limit. In addition, we experimented with increasing input kernel sizes for structure 1 through 5 to find out if it plays a significant role in the accuracy of the model.

Table 1: ResNet structure parameter setting

Structure #	N	B	C_{in}	F_0	P_0	F_1	P_1	Param #
1	4	2 1 1 1	64	3	1	3	1	4,977,226
2	4	2 1 1 1	64	5	2	3	1	4,980,298
3	4	2 1 1 1	64	7	3	3	1	4,984,906
4	4	2 1 1 1	64	9	4	3	1	4,991,050
5	4	2 1 1 1	64	11	5	3	1	4,998,730
6	3	5 2 1	64	5	2	5	2	4,967,754
7	3	1 1 1	128	3	1	3	1	4,896,394
8	2	1 3	64	7	3	7	3	4,837,962
9	2	3 1	128	7	3	5	2	4,972,682

An alternative approach to build a deep ResNet model is to reduce the number of residual layers and increase the number of blocks for the remaining residual layers. In structure 6, we removed the layer with 512 channels and increased the blocks in the 64, 128 and 256 channel layers. In structure 7, we removed the 64 channel layer in the residual layer. In structure 8 and structure 9, we further reduced number of residuals layers by 2 and used different input channels. We use the appropriate kernel size and padding for structure 6 through 9 to maintain spatial dimension of the output.

Table 2: Optimizer hyper-parameter settings

Optimizer	Learning rate	Momentum	Weight Decay
Adam	0.001	No	Yes
Adadelta	1.0	No	Yes
Adagrad	0.01	No	Yes
SGD	0.1	Yes	Yes
SGD w/ Nesterov	0.1	Yes	Yes

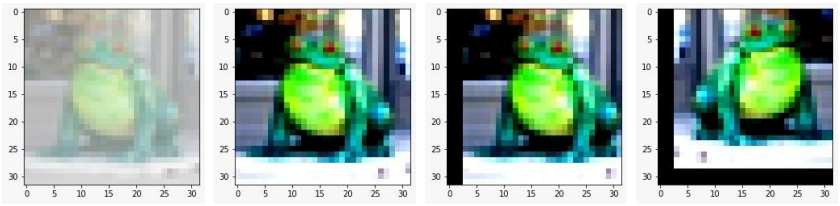
2.2 Optimizers and Regularization

Since different optimizers might work better on different models, we investigated the performance of a few popular deep learning optimizers, as showed in Table 2, on all or the ResNet structures we described in Table 1, with the Cross Entropy loss function. Although selecting hyperparameters for the optimizers is tedious, the CIFAR-10 dataset has been used in

research, and many researchers have open-sourced their model and hyper-parameters [3]. Thus, we selected the hyper-parameters of the optimizers based on the model by Liu K as shown in table 2, which achieved a high accuracy on various optimizers [6]. In their models, they use a momentum of 0.9 to reduce the oscillation in each of the gradient steps. They also use a weight decay, L2 regularization, of 0.0005 to prevent the model from overfitting.

2.3 Data augmentation and normalization

Another way to prevent the model from overfitting is through data augmentation. Since data augmentation virtually adds more training data, our model will be able to generalize better than a model trained on data that does not use augmentation. As shown in Figure 3, there are two augmentation techniques that we use in our model. The first transformation is random crop of 32 with padding of 4, which crops a random part of an image and returns it with the preset dimensions. The second transformation is random horizontal flip that randomly flips an image horizontally. Lastly, we normalized the data with the mean (0.4914, 0.4822, 0.4465) and the standard deviation (0.2023, 0.1994, 0.2010) of the CIFAR-10 dataset. We trained our models either with or without data augmentation/transformation to compare their performance.



(a) (b) (c) (d)

Figure 3: The effect of normalization followed by different transformations on an image. (a) Original image. (b) Normalized image. (c) Image normalized followed by Random Crop. (d) Image normalized, followed by Random Crop and then Random Flip

2.4 Learning rate scheduler

We use cosine annealing, as described in Figure 4, as the learning rate scheduler to increase the convergence speed of the optimizers while reducing computational requirements.

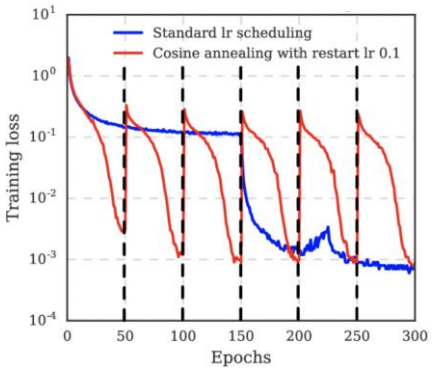


Figure 4: An adapted image from the example of cosine annealing by Huang G. et al. [4]. Cosine annealing starts the first training epoch with a large learning rate before rapidly decreasing to a minimum value and then resetting the learning rate to the base value. The training loss using cosine annealing (red line) exponentially decreases to a local minima at the 50th epoch while the training loss using standard learning rate scheduling moved steadily before rapidly decreasing at the 150th epoch to local minima.

3 Results

Each of the different ResNet architecture variations described in Table 1 were trained and tested on the CIFAR-10 dataset once, for 500 epochs using different optimizers. Table 3 shows results from these runs with and without data augmentation. A total of 90 different models were trained and tested (9 structures x 5 optimizers x 2 with/without augmentation). An upward arrow indicates that the structure performed relatively well while downward arrow indicates that the structure under-performed. The highlighted cells indicate the best performing optimizer for the structure.

3.1 Best performing structure

Structure 1 trained on augmented data had the best performance amongst all the different structures. In structure 1 – 6, the model accuracy decreases with the increasing convolutional layer kernel size in each of the structures. Structure 7 – 9 underperformed when compared to the first 6 structure.

When training on non-augmented data, the opposite behavior is noticed, as structure 9 has the best accuracy and structures 7 - 9 have a relatively higher accuracy compared to structures 1 – 6. The difference between the performance of training on data with or without augmentation is discussed in detail in section 3.3.

Table 3: Accuracy results ran on CIFAR-10 from 90 runs with and without data augmentation

	Structure #	*Adam*	Adadelta	Adagrad	*SGD*	*Nesterov*	***Average***
With augmentation	1	↑ 93.10	↑ 94.82	↑ 92.56	↑ 95.03	↑ 95.02	↑ 94.11
	2	↑ 92.88	↑ 94.58	↗ 91.68	↗ 94.72	↑ 95.03	↑ 93.78
	3	↑ 92.59	↑ 94.60	→ 90.78	↗ 94.72	↑ 94.77	↗ 93.49
	4	↑ 92.53	↗ 94.36	↗ 91.11	↗ 94.61	↗ 94.71	↗ 93.46
	5	↗ 92.32	↗ 94.13	↓ 88.74	→ 94.43	↗ 94.71	→ 92.87
	6	↗ 92.00	↑ 94.58	↗ 91.39	↑ 95.33	↑ 94.89	↗ 93.64
	7	→ 91.34	→ 93.73	↑ 92.12	↘ 93.91	↓ 93.68	→ 92.96
	8	↘ 90.65	↓ 92.97	↓ 88.82	↓ 93.32	↓ 93.50	↓ 91.85
	9	↓ 89.57	→ 93.80	↘ 89.57	↘ 93.89	↘ 93.87	↓ 92.14
Without augmentation	1	↗ 85.76	→ 88.83	↗ 84.17	→ 89.27	↘ 89.07	→ 87.42
	2	↗ 85.51	↘ 88.24	→ 82.64	↘ 88.87	↓ 88.64	→ 86.78
	3	→ 84.57	↘ 87.54	↑ 88.23	↘ 88.52	↓ 88.80	↗ 87.53
	4	↘ 84.04	↘ 87.56	↘ 78.75	↓ 87.76	↓ 88.20	↓ 85.26
	5	↓ 82.75	↓ 86.25	↓ 76.29	↓ 87.50	↓ 88.32	↓ 84.22
	6	↑ 86.54	↗ 90.21	↑ 88.17	↑ 90.85	→ 90.29	↑ 89.21
	7	↗ 85.48	→ 88.88	↑ 86.87	↘ 88.58	↘ 89.15	↗ 87.79
	8	↗ 85.48	↗ 89.41	↑ 87.12	↗ 90.28	↗ 90.52	↑ 88.56
	9	↑ 85.85	↑ 91.36	↑ 87.30	↑ 91.49	↑ 91.86	↑ 89.57

3.2 Performance of different optimizers

Stochastic gradient descent with a learning rate of 0.1, a momentum of 0.9 and weight decay of 0.0005 was relatively the best optimizer while training each of the 9 structures, regardless of whether data augmentation was used or not. SGD with Nesterov and regular SGD were both very close in accuracy and both performed better compared to other optimizers. On the contrary, Adagrad was found to be a relatively weaker optimizer and it produced a model with the lowest accuracy for almost all the models trained on augmented data, and 4 of the 9 models trained on non-augmented data.

Another important consideration while choosing an optimizer is the algorithmic efficiency of the model. The model should converge within the fewest number of epochs while still achieving a relatively competitive accuracy. Table 4 shows the number of epochs required for each of the optimizers to reach the highest accuracy. While training on augmented data, all of the optimizers converge within 180 – 230 epochs with the exception Adagrad, which required almost 500 epochs. While training on non-augmented data, convergence required much fewer epochs, roughly in the range of 120 – 150 epochs. Using augmented data to train the model may result in a model with a higher testing accuracy at the expense of a slower converging training process.

Table 4: The number of epochs we achieved best accuracy in Table 3 from 500 epochs runs.

	Structure #	*Adam*	Adadelta	Adagrad	*SGD*	*Nesterov*
With augmentation	1	218	207	486	212	197
	2	216	215	493	212	215
	3	216	220	492	220	228
	4	207	192	485	224	214
	5	211	224	483	228	197
	6	212	220	478	183	182
	7	175	202	462	191	174
	8	183	222	484	225	230
	9	187	177	446	190	179
Without augmentation	1	133	145	25	127	124
	2	134	123	399	127	131
	3	127	139	52	128	125
	4	119	126	81	120	129
	5	124	126	80	124	121
	6	140	118	73	127	129
	7	153	109	25	119	114
	8	169	150	439	149	147

Figure 5 and figure 6 show the testing loss against the number of epochs for each of the 5 optimizers using augmented and non-augmented data respectively. For the augmented dataset, all of the optimizers were able to stabilize and eventually converge the model. Adam and Adagrad were able to reach close to the lowest loss in around 160 epochs, whereas both versions of SGD and Adadelta converged at about 180 epochs. During the early stages of training the model, Adadelta displays the most erratic behavior in loss, followed by regular SGD and SGD with Nesterov. Adagrad and Adam both show relatively modest amount of variation during the early stages, despite producing models that have lower accuracy. For models trained on non-augmented data, we observe much wilder swings in loss with all the optimizers. Similar to non-augmented data, Adadelta fluctuates the most followed both versions of SGD, Adam and Adagrad. Augmentation and normalization improved the model convergence behavior.

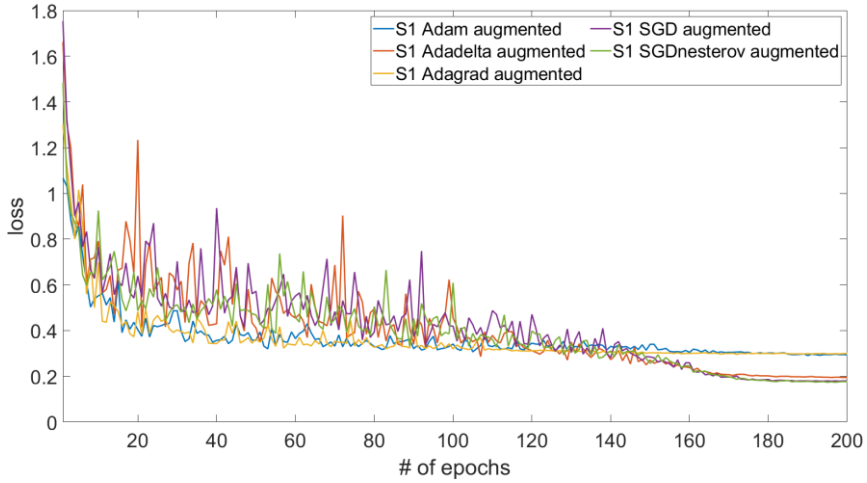


Figure 5: testing loss of structure 1 using different optimizers on augmented data

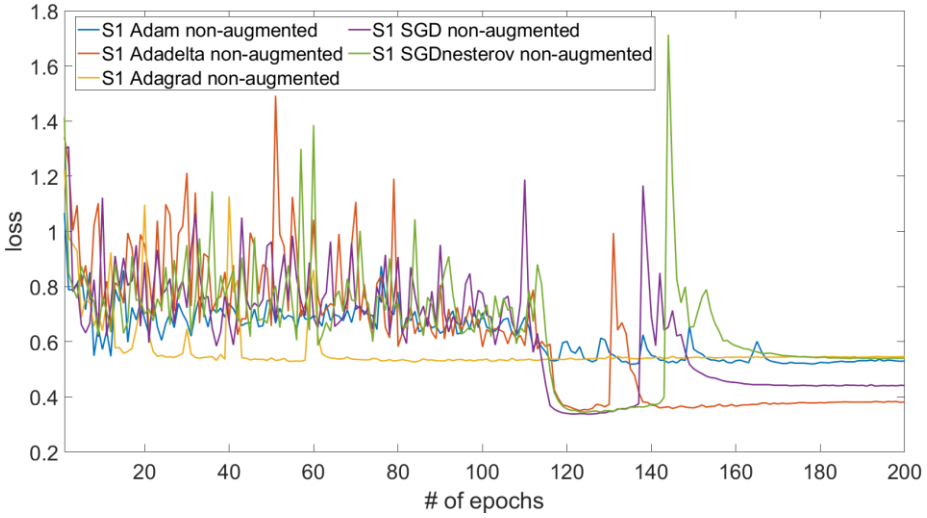


Figure 6: testing loss of structure 1 using different optimizers on augmented data

3.3 Data augmentation and normalization

In general, models trained on augmented images performed better. The highest average from the models trained on augmented data was 94.11% achieved by structure 1. With models trained on non-augmented data, the median average is about 6% lower (87.53% non-augmented, 93.46% augmented). Without augmentations, all of the architectures underperformed. Data augmentation has demonstrated to be an effective strategy to prevent overfitting to the training dataset by virtually adding new images to the dataset, in a way that preserves the original label of the image.

Figure 7 examines how data augmentation and normalization impacts the convergence behavior of the model while training on structure 1 for 200 epochs. Models trained with augmented data performed better than models trained on non-augmented data, regardless of normalization. The performance with and without normalization on augmented data had very similar performance. The model trained on non-augmented data performed slightly better with normalization. The least stable trend is when there is no data augmentation and no normalization whereas the most stable is when using them both. Thus, the transformations used improved model convergence behavior.

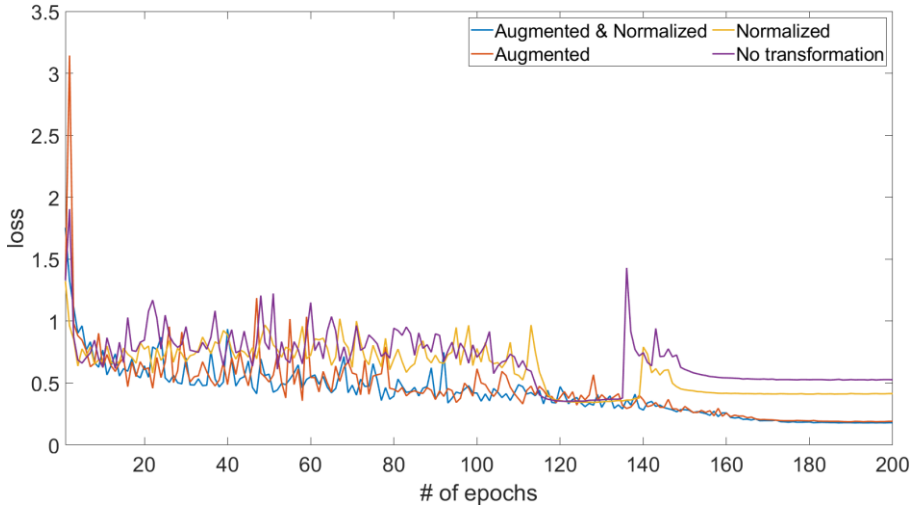


Figure 7: Testing loss over 200 epochs training structure 1 with and without data augmentation and normalization

3.4 Cosine Annealing and adaptive learning

Figure 8 shows the convergence behaviors with and without adaptive learning, cosine annealing, on different using the SGD optimizer with a learning rate of 0.1 and a momentum 0.9. The first one to converge is (blue line), the second is (orange line), and the third is (yellow line). Training without cosine annealing did not converge (purple line). This proves that the adaptive learning technique not only help to prevent the learner from getting stuck in a local minima, but also helped to exponentially increase the learning speed while still maintaining a decent accuracy.

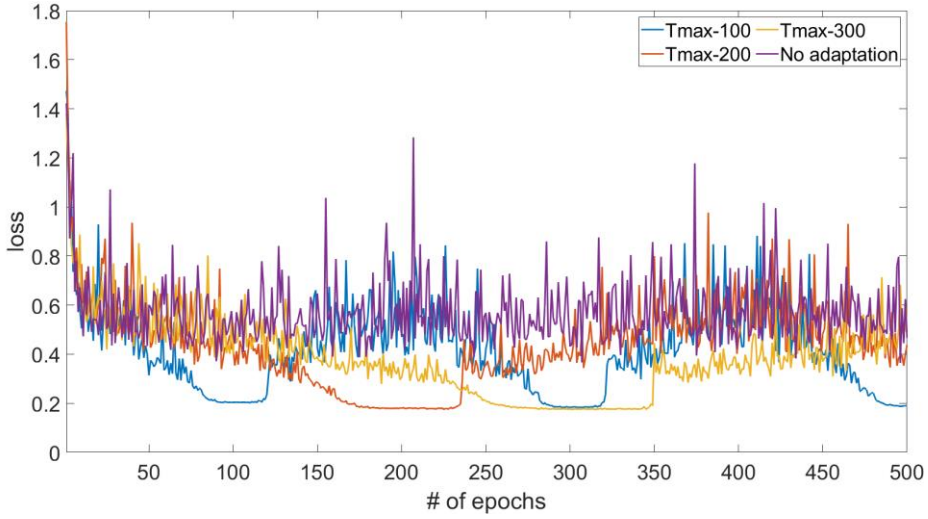


Figure 8: Convergence behaviors with and without adaptive learning using SGD

4 Conclusion

Our constrained ResNet design achieved the best accuracy at 95.03% with these following parameters setting: Four residual layers, $N = 4$. Two residual blocks in the first residual layer, one residual block in the second, third, and fourth residual layers, $B = [2,1,1,1]$. The convolutional layer kernel size of the input layer is 3, $F_o = 3$. The convolutional padding size of the input layer is 1, $P_o = 1$. The convolutional layer kernel sizes of every residual layer are 3, $F_1 = 3$. The convolutional padding sizes of every residual layer are 1, $P_1 = 1$. The convolutional channel size of the input layer is 64, $C_{in} = 64$. The convolution channel sizes of the residual layers are 64, 128, and 256 respectively, $C_1 = 64$, $C_2 = 128$, $C_3 = 256$. The skip connection kernel sizes of every layer are 1, $K = 1$.

References

[1] Kolesnikov, A. & Beyer, L. & Zhai, X. & Puigcerver, J. & Yung, J. & Gelly, S. & Houlsby, N. arXiv:1912:11370 (2020). Big Transfer (BiT): General Visual Representation Learning.

[2] Huang, Y. & Cheng, Y. & Bapna, A. & Firat, O. & Chen, M.X. & Chen, D. & Lee, H.J. & Ngiam, J. & Le, Q.V. & Wu, Y. & Chen, Z. arXiv:1811:06965 (2019). GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism.

[3] Paper with Code, Image Classification on CIFAR-10, https://paperswithcode.com/sota/image- classification-on-cifar-10

[4] Huang G. & Li Y. & Pleiss G. & Liu Z. & Hopcroft J.E. & Weinberger K.Q. arXiv:1704:00109 (2017) Snapshot Ensembles: Train 1, get M for free.

[5] Loshchilov, I. & Hutter, F. arXiv:1608:03983 (2017). SGDR: Stochastic Gradient Descent with Warm Restart.

[6] Liu K., Train CIFAR10 with PyTorch (2017), https://github.com/kuangliu/pytorch-cifar