Image Classification with Limited Sized ResNets

Faiz Ganz, Aditya Adya
Department of Computer Science
New York University

May 6, 2022

Abstract

The following paper explores the implementation of residual neural network (ResNets) with size limited to 5 million parameter and demonstrates how such models can still perform superbly on image classification tasks. Using the CIFAR10 dataset to train and evaluate our model, we propose suitable architectures for the given requirements and present a high-performing training approach, that resulted in our best model achieving a 95.67% test accuracy.

1 Introduction

With image classification being a fundamental task for cutting edge technology in fields such as cancer treatment and self-driving cars, it is important that these models achieve high accuracy with limited complexity. Nowadays, machine learning models can be integrated in personal devices and applications that can only reserve a limited amount of memory for such models. Large image classifiers are extenuating to train and hard to store, hence we propose the implementation of a Residual Neural Network (ResNet) model for image classification with less than 5 million trainable parameters and show that such models can still achieve high accuracy, while having much lower memory requirements.

1.1 Data

The dataset of choice is the CIFAR10 dataset, a widely used image dataset containing 50k training samples and 10k test samples. The samples are 32x32 RGB images of objects belonging to 10 different classes, such as dogs, cats, airplanes, and cars [1]. The models tested were trained without the use of a validation portion, on the entire 50k training samples, plus other samples obtained through data augmentation. The models were tested against the 10k test samples and the training data was organized in batches of 32 samples.

2 Methodology

2.1 Data Augmentation

For our data augmentation approach, we selected to opt for offline augmentation, applying the image transformations to separate copies of the dataset. The training dataset has been augmented five times, resulting in a total of 300k samples, including the original images. There were five augmentations of choice, each made on a copy of the original dataset. The inspiration for these augmentations has been taken by a variety of online sources and papers, and the techniques are widely used in the machine learning community [2][3].

Note: The sample images displayed were not normalized for illustrative purposes.

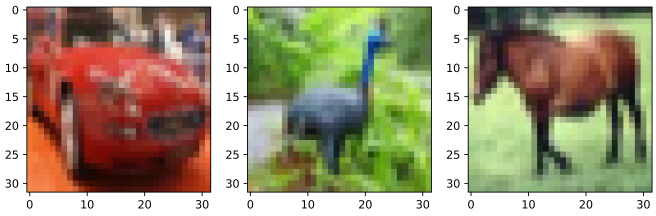

Figure 1: Original Images

First augmentation: The first augmentation of choice was to create a horizontally flipped version of the dataset, just to guarantee that the model would learn all objects symmetrically. For the following four augmentations, the dataset was horizontally flipped at random with a probability of each image getting flipped of 0.5, before applying the other augmentation techniques.

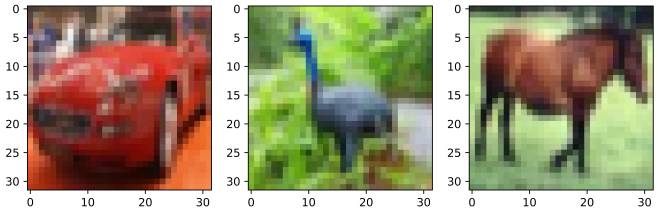

Figure 2: First Augmentation

Second augmentation: The second augmentation consists of padding the images with 4 pixels on each side, resulting in 36x36 images, and then cropping a 32x32 section from them, effectively resulting in a random translation of the images in between -4 and 4 pixels, along each axis.

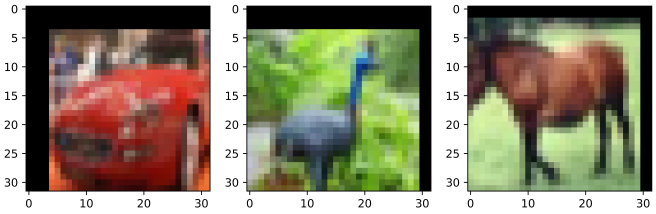

Figure 3: Second Augmentation

Third augmentation: The third augmentation scales the images to a 36x36 resolution, and follows by randomly cropping a 32x32 pixels area within it, effectively resulting in a fixed scaling by a factor of 1.125, followed by a translation.

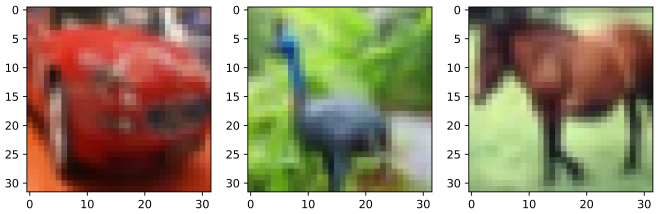

Figure 4: Third Augmentation

Fourth augmentation: The fourth augmentation rotates each image by a random angle in between -90 and 90 degrees. The choice of angle range was based on the assumption that the objects would never present themselves upside down, and hence that there is no need for rotating the images beyond that range.

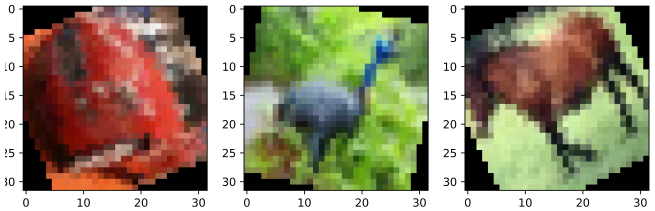

Figure 5: Fourth Augmentation

Fifth augmentation: The fifth and final augmentation was a random change in brightness, contrast, saturation, and hue of the image. Each change was in between 0.5 and 1.5 of the original value.

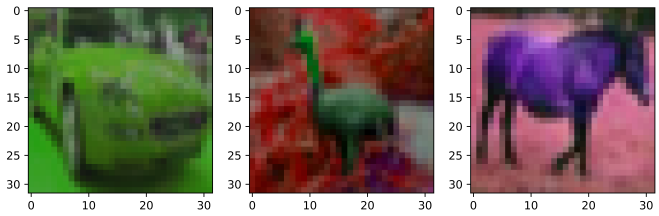

Figure 6: Fifth Augmentation

2.2 Training Regiment

The loss function used for the problem was a standard multi-class cross-entropy loss function, while the optimizer of choice was a vanilla stochastic gradient descent algorithm, with momentum of 0.9 and weight decay of 0.0001. The optimizer used a multi-step learning rate scheduler, with learning rate starting at 0.1 and decreasing by a factor of 10 at different epochs (milestones) in the training phase. It has been shown that a momentum based stochastic gradient descent optimizer, with a multi-step learning rate schedule, yields higher quality local minima than the more widely used adaptive techniques such as ADAM [4]. Hence, we have followed a similar training configuration for our experiments. The specific learning rate decay schedule we have used for our models was initially inspired by the schedule used by Huang and Liu [5], with the learning rate decreasing by a factor of 10 at 50% and 75% of the total training epochs. In the paper [6], the model was trained over 64k iterations, while our model was trained for only 300, due to the difference in training data used. The number of epochs we selected was based on an educated guess of the speed at which the test loss was decreasing and saturating. Later on we decreased the number of epochs to 150, with learning rate decays at 20, 50, 100, and 125 epochs. Ultimately however, for our highest accuracy models we have used an even tighter schedule, as we noticed that the test loss started saturating significantly before the each milestone–sometimes even showing signs of over-fitting. Hence, we decided to select milestones that were just a few epochs away from when the loss started saturating again. The final schedule used has milestones at 15, 25, 35, and 45 epochs, over a total of 50 epochs, and starts from a learning rate of 0.1. This scheduling gave us the highest performing models we trained.

3 Architecture

The ResNets trained used the original ResNet18 architecture as a template. We have not modified any activation function, nor included techniques such as bottlenecks for our experiments. The only aspects we experimented on were the number of residual layers, the number of residual blocks in each residual layer, the number of channels in each residual layer, and the kernel sizes in the convolutional layers and skip connections. The ResNet 18 is an 18 trainable-

layers residual neural network with 4 residual layers, each containing 2 residual blocks, with each block containing 2 convolutional layers and a skip connection. The 4 residual layers follow an initial convolutional layer and are followed by a final fully connected layer. The channels increase from 64 to 512 by a factor of two with each new residual layer. Each convolutional layers has a 3x3 kernel size and all skip connection have a 1x1 kernel size. ResNet18 however contains more than 5 million parameters and hence, our modifications to the architecture of the model were directed by the problem of what hyper-parameter should we reduce first to meet the size requirement.

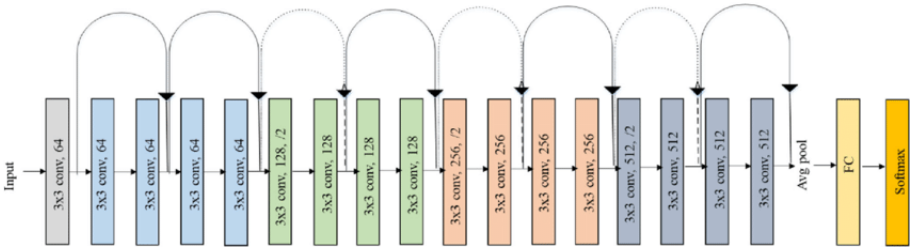

Figure 7: Original ResNet18 Architecture

Note: Most of the traditional ResNet architectures have either 3 or 4 residual layers [7], hence we have not experimented for number of residual layers smaller than 2 or greater than 4.

Approach I: Reducing Blocks and Kernel Size The first attempt at modifying the model was by reducing the number of residual block in each of the residual layers to one block per layer, with everything else equal. However, this model did not meet our size requirement and hence we had to also reduce some other hyper-parameter. Our choice was to leave the channels and number residual layers for separate experiments, hence we reduced the kernel size of the first convolutional layer in each residual block from a 3x3 to a 1x1 kernel. The decision was made based on the fact that reducing the kernel size in both convolutional layers would reduce the amount of parameters too significantly. After that we noticed we still had space for a few more blocks in the network and so we added 1 block to the second layer and 2 blocks to the third layer, resulting in 1 block in the first residual layer, 2 blocks in the second, 3 blocks in the third, and 1 block in the last residual layer. In assigning blocks to each layer took inspiration from well-known ResNet models [8] and tried to maintain a similar distribution of blocks across the residual layers. We had noticed that there is a tendency in models with 4 residual layers to have more blocks in the second and third layer in particular, or the middle layers more generally. With these choices the resulting model contained about 4.9 million parameters.

Approach II (a): Reducing Layers and Increasing Blocks Our second approach was to eliminate the last residual layer, containing convolutional layers with 512 channels, as these are the layers that have the highest parameter count. Eliminating the last residual layer was sufficient to bring the model significantly beneath 5 million parameters. Increasing channels was not feasible as it would have not yielded a model that met our parameter restrictions. For this approach, we decided to add residual blocks to the 3 remaining residual layers. In the next experiment we have instead opted for an increase in kernel sizes. The final architecture of this model contains 3 residual blocks in the first residual layer, 5 blocks in the second layer, and 3 in the last layer, bringing our model to about 4.9 million parameters. All other characteristics were kept the same as in the template.

Approach II (b): Reducing Layers and Increasing Kernel Similarly to the last approach, we got rid of the last residual layer, but instead of increasing the number of blocks, we increased the kernel size in the first convolution layer of each block to a 5x5 kernel. The resulting number of parameters was again around 4.9 million.

Approach III (a): Reducing Channels and Increasing Blocks For the following approach we have selected to reduce the number of channels of the original model by half their value, starting at 32 channels instead of 64, and ending with 256 instead of 512. This operation brought the parameter count beneath 5 million, even with 4 residual layers, so the model was made to reach approximately 4.9 million parameters by adding more blocks in each of the 4 layers. The final architecture has 3 blocks in layers 1, 2, and 4, and 5 blocks in layer 3. All other characteristics were kept the same as in the template.

Approach III (b): Reducing Channels and Increasing Kernel Size Similarly to approach III (a), we reduced the channel sizes by half, but instead of adding more block in each layers, we increased the kernel size in the first convolution layer in each block to a 5x5 kernel, just as in approach II (b).

Approach IV: Reducing Channels Further, Increasing Kernel Size and Blocks For the final approach tested, the model channels were reduced even further, now starting at 16 and ending at 128. The resulting model had less than 1 million parameter and to compensate we chose to combine both an increase in kernel size and an increase in residual blocks. The kernel of each convolutional layer was increases to from 3x3 to 7x7 and the skip connection kernel size was increase from a 1x1 to a 5x5 kernel. The number of blocks was also increased, resulting in 2 blocks in the first residual layer, 3 blocks in the second layer, 4 in the third, and 2 in the last. The resulting architecture contained approximately 4.9 million parameters.

4 Results and Discussion

Table 1: Model Performance Table

Architecture	Test Accuracy
Approach I	94.24%
Approach II (a)	95.67%
Approach II (b)	95.17%
Approach III (a)	95.10%
Approach III (b)	93.82%
Approach IV	92.42%

The model architecture producing the best results was the model obtained through the approach II (a), which yielded a test accuracy of 95.67%.

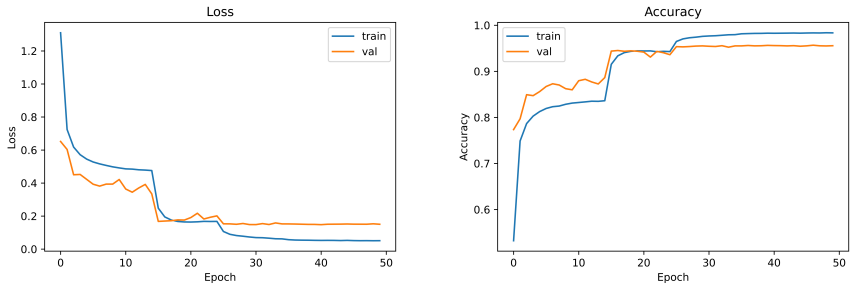

Figure 8: Best Model Loss and Accuracy

4.1 Architectural Decisions

From the experiments performed, we have noticed that the difference in model architecture did not yield widely divergent results, unless the number of channels was drastically decreased. This conclusion is consistent with the idea that in deep neural networks, the objective is to try and extract a greater number of hidden features from decomposing the original data features. Hence, it seems that sacrificing the number of channels for other model hyper-parameters is the least favorable choice. From approaches I and II, it can be inferred that, given a target number of parameters, sacrificing a residual layer to increase the number of residual blocks, when the residual blocks are too few (less than 2 per layer), such as in approach II(a), is preferable. In fact, all well-known ResNet architecture have at least 2 blocks per layer [7][8]. Furthermore, the reduction in kernel size in approach I also seems to have negatively affected model performance. By comparing approaches II (a) and II (b), it is clear that increasing model size by increasing the number of block yields better results than increasing kernel

sizes, in the context of the experiment. However, increasing the depth of the model by a disproportionate amount, either through the addition off too many of residual layers or residual blocks, will however also negatively impact model performance, as demonstrated by He and Zhang [6], which show how deeper and deeper ResNets start perform worse than their shallower counterparts. Hence, approach II (a) seems the more sensible architecture choice given the constraints, and is in fact the one yielding the best results. As mentioned at the beginning of the section, reducing the number of channels to preserve the amount of residual layers does not seem to be the best choice either, and if done, is only successful in moderation, such as in approach III (a) and III (b). By comparing the two approaches, it appears that compensating for the decrease in parameters is done better by increasing the number of blocks, rather than the kernel sizes. Reducing channels superfluously will lead to significant loss in performance, such as in the case of approach IV.

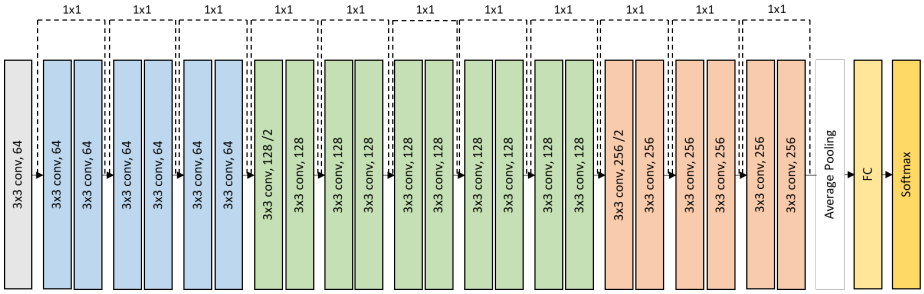

Figure 9: Architecture of Highest Performing Model

4.2 Impact of Data Augmentation and Optimized Training Regiments

The most influential factors in increasing model performance were the data and training regiment it received. Data augmentation alone was able to bring most models tested initially to about 90% or more accuracy on test data. The augmentations the data received were, as a whole, the ones that yielded the greatest increase in test accuracy, while keeping the training set size within a reasonable amount. We have not measured the contributions in test accuracy of each augmentation on its own. For the rotation transformation we thought that smaller angle ranges would be more appropriate, however, the range of -90 to 90 degrees was the one that gave us the highest results during testing. A batch size of 32 was chosen as the model tended to slightly over-fit at larger batch sizes, whenever the test loss started saturating. We have also tested for different initial learning rates, such as 0.01 and 0.001, but did not achieve better results, so we kept with the proposed initial learning rate of 0.1 [6]. The scheduler milestones used by He and Zhang [6] also seemed to be sub-optimal for our

training regiment and after training the model multiple times, we increase the number of milestones and decreased their separation, obtaining a final schedule, that is more similar to the one used by Wilson and Roelofs [4].

5 Conclusion

From the experiments conducted, a limited size ResNet can still be on par with, if not even better than, some of its larger competitors, relatively to its size, and if built with a sensible architecture and a fine-tuned training regiment. The architecture choice seems to indicate that a certain balance is required between the number of residual layers, number of residual blocks, and channel layouts. A kernel size of 3x3 seems to be a good choice for most architectures [7][8], and it seems like fine-tuning it does not yield significant benefits, although there are successful models out there that do use other kernel sizes [6]. The augmentations selected when training the model are crucial in determining how well the model can ultimately perform. As long as significant variations of the original training data are provided, the model should generalize well. Finally, using stochastic gradient descent and a multi-step learning rate schedule allows to achieve the highest quality minima and highest performing models. Through this methodology our best performing model achieved 95.67% accuracy on the CIFAR10 test dataset.

References

[1] [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[2] S. Tanwar, "Image augmentation - improving deep learning models," Aug 2021. [Online]. Available: https://medium.com/analytics-vidhya/image-augmentation-9b7be3972e27

[3] A. Gandhi, "Data augmentation: How to use deep learning when you have limited data," May 2021. [Online]. Available: https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/

[4] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," *Advances in neural information processing systems*, vol. 30, 2017.

[5] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[7] Y. Idelbayev, "Proper ResNet implementation for CIFAR10/CIFAR100 in PyTorch," https://github.com/akamaster/pytorch_resnet_cifar10, accessed: 2022-04-29.

[8] Kuangliu, "Kuangliu/pytorch-cifar: 95.47% on cifar10 with pytorch." [Online]. Available: https://github.com/kuangliu/pytorch-cifar