Image classification of CIFAR-10 using modified residual network (ResNeXt) Architecture

Divya Sai Tirumalasetty, Pranav Doma, Prathyusha Kadali
dt2482, pd2365, pk2669

Code link: https://github.com/dstirumalasetty/Deep-Learning-Mini-Project

Abstract

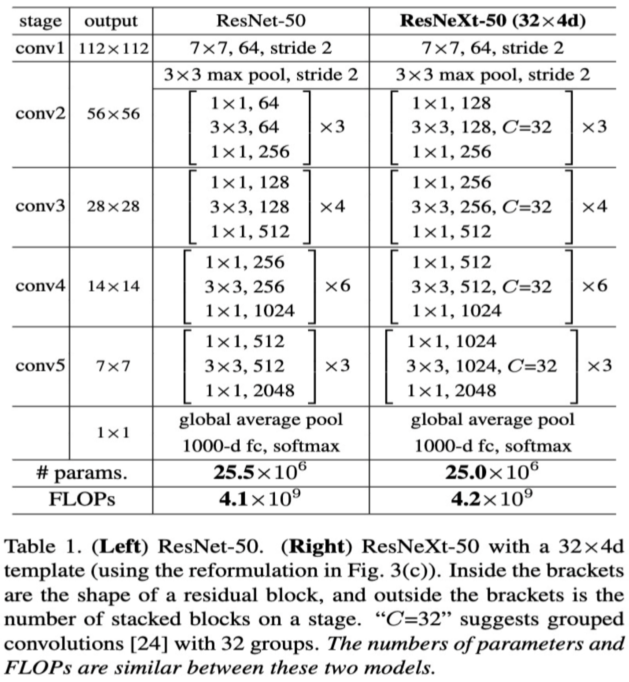
This paper aims to investigate the significance of residual blocks and their usage in image classification tasks. The study starts by analyzing some of the existing neural network architectures and utilizing that understanding to construct our own residual model. Specifically, we implemented ResNeXt architecture, trained using both SGD optimizer and AMP regularization technique, and focused on the decision-making process leading to this architectural choice. The goal was to achieve maximum accuracy while keeping the number of parameters below 5 million and minimizing the loss.

Overview

Image classification is a vital problem in computer vision that involves predicting the label of an image from a fixed set of categories. The ability to classify images accurately has numerous practical applications, including medical image diagnosis, self-driving cars, and facial recognition. The use of deep neural networks has dramatically improved image classification performance and residual neural networks (ResNets) have emerged as one of the most successful deep learning architectures However, recent studies have shown that ResNext architecture can outperform ResNet in certain scenarios by introducing a new dimension of cardinality that enables better feature reuse and diversity.

The motivation for choosing ResNext over ResNet was because ResNext introduces a novel feature extraction method using grouped convolutions, which can improve the performance of the network without significantly increasing its computational complexity. This allows ResNext to achieve state-of-the-art results on a variety of image classification benchmarks.

In this project, ResNext was trained and tested on the CIFAR-10 dataset using both SGD and AMP optimizers. The AMP optimizer was chosen due to its ability to handle adversarial examples and improve model robustness. However, it was observed that the use of AMP increased the training time compared to SGD, which could be a potential drawback for large-scale datasets. In addition to using the ResNext architecture, we made several modifications to the network to reduce the number of hyperparameters and improve its efficiency. Specifically, we decreased the number of layers in the network and adjusted the size of the convolutional filters.



stage	output	ResNet-50		**ResNeXt-50 (32×4d)**	
conv1	112×112	7×7, 64, stride 2		7×7, 64, stride 2	
conv2	56×56	3×3 max pool, stride 2		3×3 max pool, stride 2	
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}$	×3	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C{=}32 \\ 1\times1, 256 \end{bmatrix}$	×3
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}$	×4	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C{=}32 \\ 1\times1, 512 \end{bmatrix}$	×4
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}$	×6	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C{=}32 \\ 1\times1, 1024 \end{bmatrix}$	×6
conv5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}$	×3	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, C{=}32 \\ 1\times1, 2048 \end{bmatrix}$	×3
	1×1	global average pool 1000-d fc, softmax		global average pool 1000-d fc, softmax	
# params.		$\mathbf{25.5}\times10^6$		$\mathbf{25.0}\times10^6$	
FLOPs		$\mathbf{4.1}\times10^9$		$\mathbf{4.2}\times10^9$	

Table 1. (**Left**) ResNet-50. (**Right**) ResNeXt-50 with a 32×4d template (using the reformulation in Fig. 3(c)). Inside the brackets are the shape of a residual block, and outside the brackets is the number of stacked blocks on a stage. "C=32" suggests grouped convolutions [24] with 32 groups. *The numbers of parameters and FLOPs are similar between these two models.*

Table 1: Example comparison of ResNet vs ResNeXt parametres

The comparison between the two optimizers showed that AMP outperformed SGD in terms of accuracy and loss using standard parameters. This highlights the potential benefits of using AMP for deep neural networks, as it can improve the robustness and accuracy of the model, especially when dealing with adversarial examples.

About the Dataset

The CIFAR-10 dataset is extensively used as a standard for image classification. It comprises 60,000 color images of 32x32 pixels, separated into ten groups with 6,000 images each. The dataset is split into a training dataset of 50,000 images and a test dataset of 10,000 images. The test dataset contains 1,000 images randomly selected from each category. The training dataset is partitioned into five batches, with each batch containing 10,000 images from diverse categories. The dataset includes regular objects such as dogs, cats, cars, and birds. Due to its manageable size and straightforward images, CIFAR-10 is valuable for assessing image classification models.

Data Augmentation

Data augmentation is a technique used to enhance the training dataset by applying different transformations such as cropping, flipping, and rotating to the images. It is used to prevent overfitting and improve the model's ability to generalize by increasing the diversity and quantity of the training data. When a model only learns to recognize the training images in their original form, it may not perform well on new images. Data augmentation addresses this problem by increasing the number of training examples and helping the model to recognize the same features in various versions of the images.

We performed random crop and horizontal flips on the training images. A random section of the image is cropped out and resized to the desired size. In this code, a crop of size 32 is taken and padded with 4 pixels on each side. This increases the variability in the training set, making the model more robust to different object sizes and locations within the image. The image is randomly flipped horizontally, which further increases the variability in the training set. This is useful for tasks like object recognition, where the orientation of the object does not affect its identity. The pixel values in the image are normalized to have a mean of (0.4914, 0.4822, 0.4465) and a standard deviation of (0.2023, 0.1994, 0.2010). This standardizes the input to the model, making it easier to train and improving performance.

Architecture

In this project we are using ResNeXt architecture, a variant of ResNet architecture that was introduced in 2016 and has been shown to achieve higher accuracy than ResNet. We chose ResNeXt based architecture due to its potential to provide better performance than ResNet18 on large-scale image classification datasets, while still being relatively efficient in terms of parameters.

The ResNeXt architecture introduces a new concept of cardinality, which allows it to achieve better performance by aggregating multiple paths of varying dimensions within each residual block. This makes ResNeXt more flexible in terms of network architecture and potentially more efficient in terms of parameters when the cardinality is properly chosen.

In our implementation, we used a ResNeXt29 based architecture with a bottleneck width of 4 and cardinality of 32. The network takes a 32x32 input image and produces an output tensor of shape [-1, 10], where -1 represents a variable batch size and 10 represents the number of output classes.

The residual blocks in the model consist of a shortcut or skip connection, which allows the input to by-pass one or more layers and be added to the output of the block. This helps to ensure that the information from the input is preserved and propagated through the network, even as the network gets deeper.
In the model, the residual blocks are grouped convolutional blocks, which consist of three convolutional layers: two 2D convolutional layers with batch normalization and ReLU activation, and a third 1x1 convolutional layer with batch normalization. The first convolutional layer reduces the number of input channels, while the second convolutional layer applies a group convolution operation with a specified cardinality and bottleneck width, and the third convolutional layer increases the number of output channels. The second layer uses group convolution with the number of groups equal to the cardinality, which is set to 32 in this case. The shortcut or skip connection allows the input to be added to the output of the block before the final activation is applied.

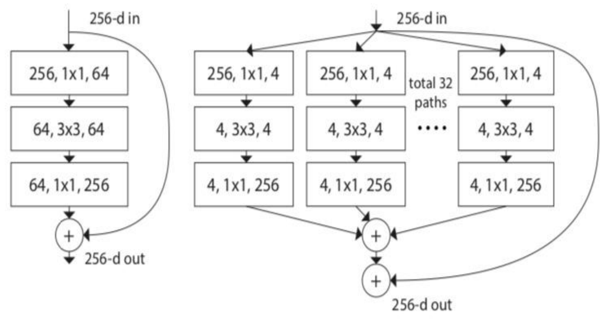
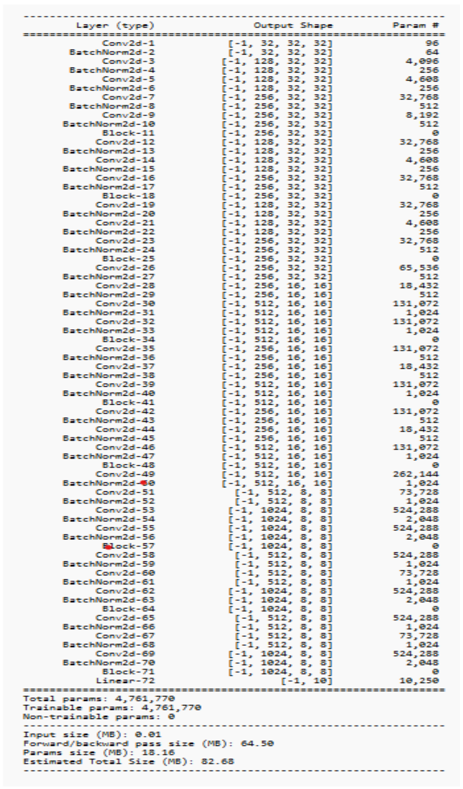


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

Figure 1: Example comparison of ResNet vs ResNeXt



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 32, 32]              96
       BatchNorm2d-2           [-1, 32, 32, 32]              64
            Conv2d-3          [-1, 128, 32, 32]           4,096
       BatchNorm2d-4          [-1, 128, 32, 32]             256
            Conv2d-5          [-1, 128, 32, 32]           4,608
       BatchNorm2d-6          [-1, 128, 32, 32]             256
            Conv2d-7          [-1, 256, 32, 32]          32,768
       BatchNorm2d-8          [-1, 256, 32, 32]             512
            Conv2d-9          [-1, 256, 32, 32]           8,192
      BatchNorm2d-10          [-1, 256, 32, 32]             512
            Block-11          [-1, 256, 32, 32]               0
           Conv2d-12          [-1, 128, 32, 32]          32,768
      BatchNorm2d-13          [-1, 128, 32, 32]             256
           Conv2d-14          [-1, 128, 32, 32]           4,608
      BatchNorm2d-15          [-1, 128, 32, 32]             256
           Conv2d-16          [-1, 256, 32, 32]          32,768
      BatchNorm2d-17          [-1, 256, 32, 32]             512
            Block-18          [-1, 256, 32, 32]               0
           Conv2d-19          [-1, 128, 32, 32]          32,768
      BatchNorm2d-20          [-1, 128, 32, 32]             256
           Conv2d-21          [-1, 128, 32, 32]           4,608
      BatchNorm2d-22          [-1, 128, 32, 32]             256
           Conv2d-23          [-1, 256, 32, 32]          32,768
      BatchNorm2d-24          [-1, 256, 32, 32]             512
            Block-25          [-1, 256, 32, 32]               0
           Conv2d-26          [-1, 256, 32, 32]          65,536
      BatchNorm2d-27          [-1, 256, 32, 32]             512
           Conv2d-28          [-1, 256, 16, 16]          18,432
      BatchNorm2d-29          [-1, 256, 16, 16]             512
           Conv2d-30          [-1, 512, 16, 16]         131,072
      BatchNorm2d-31          [-1, 512, 16, 16]           1,024
           Conv2d-32          [-1, 512, 16, 16]         131,072
      BatchNorm2d-33          [-1, 512, 16, 16]           1,024
            Block-34          [-1, 512, 16, 16]               0
           Conv2d-35          [-1, 256, 16, 16]         131,072
      BatchNorm2d-36          [-1, 256, 16, 16]             512
           Conv2d-37          [-1, 256, 16, 16]          18,432
      BatchNorm2d-38          [-1, 256, 16, 16]             512
           Conv2d-39          [-1, 512, 16, 16]         131,072
      BatchNorm2d-40          [-1, 512, 16, 16]           1,024
            Block-41          [-1, 512, 16, 16]               0
           Conv2d-42          [-1, 256, 16, 16]         131,072
      BatchNorm2d-43          [-1, 256, 16, 16]             512
           Conv2d-44          [-1, 256, 16, 16]          18,432
      BatchNorm2d-45          [-1, 256, 16, 16]             512
           Conv2d-46          [-1, 512, 16, 16]         131,072
      BatchNorm2d-47          [-1, 512, 16, 16]           1,024
            Block-48          [-1, 512, 16, 16]               0
           Conv2d-49          [-1, 512, 16, 16]         262,144
      BatchNorm2d-50          [-1, 512, 16, 16]           1,024
           Conv2d-51            [-1, 512, 8, 8]          73,728
      BatchNorm2d-52            [-1, 512, 8, 8]           1,024
           Conv2d-53           [-1, 1024, 8, 8]         524,288
      BatchNorm2d-54           [-1, 1024, 8, 8]           2,048
           Conv2d-55           [-1, 1024, 8, 8]         524,288
      BatchNorm2d-56           [-1, 1024, 8, 8]           2,048
            Block-57           [-1, 1024, 8, 8]               0
           Conv2d-58            [-1, 512, 8, 8]         524,288
      BatchNorm2d-59            [-1, 512, 8, 8]           1,024
           Conv2d-60            [-1, 512, 8, 8]          73,728
      BatchNorm2d-61            [-1, 512, 8, 8]           1,024
           Conv2d-62           [-1, 1024, 8, 8]         524,288
      BatchNorm2d-63           [-1, 1024, 8, 8]           2,048
            Block-64           [-1, 1024, 8, 8]               0
           Conv2d-65            [-1, 512, 8, 8]         524,288
      BatchNorm2d-66            [-1, 512, 8, 8]           1,024
           Conv2d-67            [-1, 512, 8, 8]          73,728
      BatchNorm2d-68            [-1, 512, 8, 8]           1,024
           Conv2d-69           [-1, 1024, 8, 8]         524,288
      BatchNorm2d-70           [-1, 1024, 8, 8]           2,048
            Block-71           [-1, 1024, 8, 8]               0
           Linear-72                   [-1, 10]          10,250
================================================================
Total params: 4,761,770
Trainable params: 4,761,770
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 64.50
Params size (MB): 18.16
Estimated Total Size (MB): 82.68
----------------------------------------------------------------
```

Figure 2: Summary of modified ResNext29 architecture of model

The number of channels in the 3x3 convolutional layer is equal to the cardinality multiplied by a bottleneck width of 4. The bottleneck width is increased by a factor of 2 after each stage. Each block also has a shortcut connection that skips over the convolutional layers and is added to the output of the final convolutional layer before the ReLU activation.

The ResNeXt architecture consists of multiple stages, each containing multiple residual blocks, with the number of blocks and the stride of the first block varying by stage. The output of the final stage is fed into an average pooling layer, followed by a linear layer to produce the final output. The number of channels in the final output is determined by the number of classes in the classification task.

Methodology

We have trained the model using the Adversarial Model Perturbation (AMP) regularization technique along with the Stochastic Gradient Descent (SGD) optimizer to classify images in the CIFAR-10 dataset.

Stochastic Gradient Descent (SGD):
SGD is a widely used optimization algorithm that updates a model's parameters iteratively to minimize a loss function. It is a standard method for training deep learning models and is effective in most cases. SGD is simple, computationally efficient, and easy to implement.

Adversarial Model Perturbation (AMP):
Adversarial model perturbation (AMP) is a regularization scheme that improves generalization and reduces overfitting in deep learning. The key idea behind AMP is that flat local minima of the empirical risk lead to better generalization. Instead of directly minimizing the empirical risk, AMP applies the "worst" norm-bounded perturbation on each point in the parameter space, resulting in a more robust and generalizable model.
In AMP, the model is trained on CIFAR data with added adversarial perturbations. Specifically, the optimizer adds small perturbations to the input data in each iteration of the inner optimization loop, based on the gradients of the loss with respect to the data. The perturbations are constrained to a norm ball around each data point, with a radius specified by the epsilon parameter. The inner optimization loop is repeated for a number of iterations specified by the iner_iter parameter. The outer optimization loop updates the model parameters based on the gradients of the loss with respect to the parameters, using a base optimizer specified by the base_optimizer parameter (e.g., SGD). SGD converges to the local minimum of the loss function for the training data, while AMP may converge to a suboptimal solution due to the added adversarial perturbations. The additional inner optimization loop in AMP may increase the training time compared to SGD, but the difference in training time depends on the complexity of the model and the size of the dataset.

Model:
We used the ResNeXt29 based model which has 3 main components: the convolutional layer, the residual blocks, and the fully connected layer. This model has 29 layers and 2 sets of convolutional layers with 64 channels. The model has 3 blocks in each of the first three stages, and each block has a cardinality of 32 and a bottleneck width of 4. The fully connected layer has 10 output neurons, corresponding to the 10 classes in the CIFAR-10 dataset. The learning rate for the SGD optimizer is set to 0.001 and the weight decay is set to 0.0001. The batch size is set to 64 and the number of epochs is set to 100. We use cross-entropy loss as the loss function for training the model.
The AMP optimizer has been configured with a learning rate of 0.1, an epsilon value of 0.5, and a momentum of 0.9. The learning rate determines the step size during the optimization process, while the epsilon value specifies the maximum size of the adversarial perturbation. The momentum factor determines how much the optimizer should consider the previous update direction when computing the current update.

The input x is passed through a series of operations in a convolutional neural network, including convolutions, batch normalization, ReLU activation, and multiple Block modules. The output is then passed through an average pooling layer and a fully connected layer to produce the final output. The _make_layer function generates a layer made up of several Block modules based on the number of blocks input, each producing an output of shape (batch_size, out_planes, height, and width). The output shape of each block depends on the stride, with the second and third blocks reducing the spatial resolution by half and a quarter, respectively.

Results

Our study aimed to classify images in the CIFAR-10 dataset using a modified ResNeXt29 architecture with a bottleneck width of 4 and cardinality of 32, containing a total of 4,761,770 parameters. We implemented two regularization techniques: Adversarial Model Perturbation (AMP) and Stochastic Gradient Descent (SGD) optimizer, to improve the model's accuracy and reduce its loss.

Our experiments demonstrated that the AMP regularization technique produced a higher accuracy of 90-91% compared to the SGD optimizer's accuracy of 89-90% with a minimum loss. We trained the model over 100 epochs and plotted the training and testing accuracy for both regularization techniques. The graph indicated that the AMP technique resulted in a higher accuracy rate than SGD at each epoch.

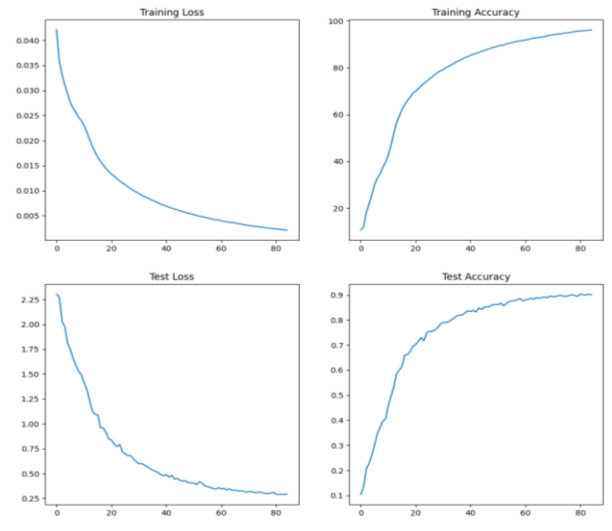


Figure 3: AMP training and testing loss and accuracy plots

From the above AMP graphs and the below SGD graphs, AMP shows better performance than SGD for image classification, as depicted in the graph comparing training and testing loss and accuracy values.

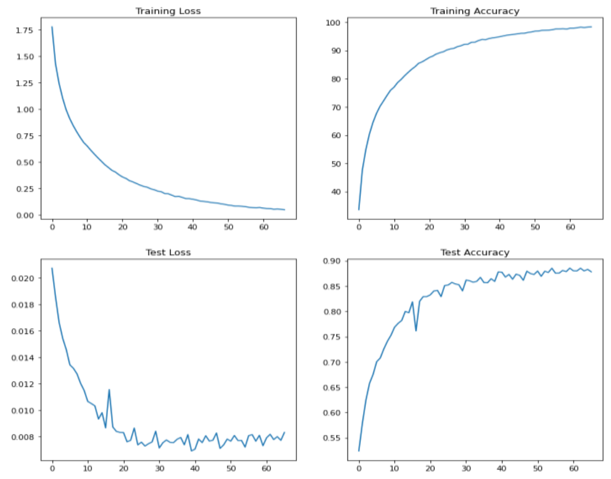


Figure 4: SGD training and testing loss and accuracy plots.

Thus, our results suggest that incorporating the AMP regularization technique can significantly improve the classification accuracy of the ResNeXt29 model on the CIFAR-10 dataset.

References

ArXiv Paper
Li, X., Wang, W., Hu, X., & Yang, J. (2019). Selective Kernel Networks. arXiv preprint arXiv:1903.06586

ArXiv Paper
Xie,S., Girshick,R., Dollár,P., Tu,Z., He,K., (2017). Aggregated Residual Transformations for Deep Neural Networks . arXiv preprint arXiv:1611.05431

ArXiv Paper
Zheng, Y., Zhang, R., & Mao, Y. (2020). *Regularizing Neural Networks via Adversarial Model Perturbation*. arXiv preprint arXiv:2010.04925

Website or online resource
H. (n.d.). *AMP-Regularizer/pyramidnet.py at master · hiyouga/AMP-Regularizer*. GitHub. https://github.com/hiyouga/AMP-Regularizer

Website or online resource
Residual Neural Network (ResNet). (2020, January 24). OpenGenus IQ: Computing Expertise & Legacy. https://iq.opengenus.org/residual-neural-networks/

Website or online resource
K. (2021, February 25). *GitHub - kuangliu/pytorch-cifar: CIFAR10 with PyTorch*. GitHub. https://github.com/kuangliu/pytorch-cifar