Final project report on Wide ResNet with CIFAR10

Hasnain Ali Shah[1]

[1]Department of Artificial Intelligence, Kyungpook National University

June 11, 2022

Abstract

In this project Wide ResNet50 model is implemented to the CIFAR10 dataset. in order to improve the model accuracy following additions were made. to improve the data changing image size, data augmentation, normalization. To improve the model weight decay, momentum and cosine learning rate scheduler were used. At the end all model tuning steps proved effective but only after increasing epoch size 3 times bring the final accuracy of 95.53 percent.

Keywords
Deep Learning, Classification, Cifar10

1 Introduction

Deep residual networks were shown to be able to scale up to thousands of layers and still have improving performance. However, each fraction of a percent of improved accuracy costs nearly doubling the number of layers, and so training very deep residual networks has a problem of diminishing feature reuse, which makes these networks very slow to train. To tackle these problems, in 2016 authors S Zagoruyko and N Komodakis conducted a detailed experimental study [4] on the architecture of ResNet blocks and proposed a novel architecture where residual networks depth decreased and width increased. Authors call the resulting network structures wide residual networks (WRNs) and show that these are far superior over their commonly used thin and very deep counterparts.

1.1 Wide Residual Networks

[1] showed that the widening of ResNet blocks (if done properly) provides a much more effective way of improving performance of residual networks compared to increasing their depth.

group name	output size	block type = $B(3,3)$
conv1	32×32	[3×3, 16]
conv2	32×32	$\begin{bmatrix} 3\times3, 16\times k \\ 3\times3, 16\times k \end{bmatrix} \times N$
conv3	16×16	$\begin{bmatrix} 3\times3, 32\times k \\ 3\times3, 32\times k \end{bmatrix} \times N$
conv4	8×8	$\begin{bmatrix} 3\times3, 64\times k \\ 3\times3, 64\times k \end{bmatrix} \times N$
avg-pool	1×1	$[8 \times 8]$

Figure 1: Structure of wide residual networks. Network width is determined by factor k. Original architecture is equivalent to k = 1. Groups of convolutions are shown in brackets where N is a number of blocks in group, downsampling performed by the first layers in groups conv3 and conv4. Final classification layer is omitted for clearance. In the particular example shown, the network uses a ResNet block of type B(3,3).

In particular, wider deep residual networks that significantly improved over, having 50 times less layers and being more than 2 times faster. Wide 16-layer deep network has the same accuracy as a 1000-layer thin deep network and a comparable number of parameters, although being several times faster to train. Compared to the original architecture [1] the order of batch normalization, activation and convolution in residual block was changed from conv- BN-ReLU to BN-ReLU-conv. As the latter was shown to train faster and achieve better results. Furthermore, socalled bottleneck blocks were initially used to make blocks less computationally expensive to increase the number of layers. As [1] paper studied the effect of widening and bottleneck is used to make networks thinner it didn't consider it, focusing instead on basic residual architecture.

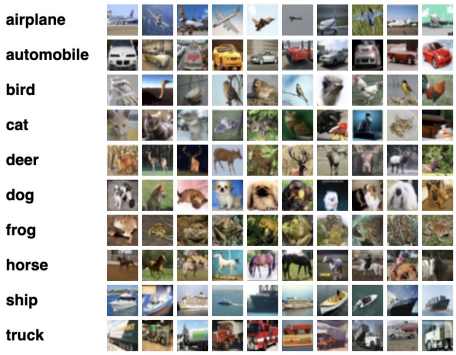

Figure 2: CIFAR-10 dataset

1.2 CIFAR10 dataset

The CIFAR-10 dataset (Figure 2)(Canadian Institute for Advanced Research, 10 classes) is a subset of the Tiny Images dataset and consists of 60000 32x32 color images. The images are labelled with one of 10 mutually exclusive classes: airplane, automobile (but not truck or pickup truck), bird, cat, deer, dog, frog, horse, ship, and truck (but not pickup truck).

1.3 Resize

Since the emergence of CNNs and their staggering success in image classification, many attempts have been made by researchers to improve their accuracy and time performance. These improvements have targeted different aspects of CNNs, including network architecture, activation functions, regularization mechanisms, and optimization techniques among others. Since neural networks receive inputs of the same size, all images need to be resized to a fixed size before inputting them to the CNN. The larger the fixed size, the less shrinking required. Less shrinking means less deformation of features and patterns inside the image. This will mitigate the classification accuracy degradation due to deformations. However, large images not only occupy more space in the memory but also result in a larger neural network. Thus, increasing both the space and time complexity. It is obvious now that choosing this fixed size for images is a matter of trade-off between computational efficiency and accuracy. Resizing all images to a fixed size is a prerequisite for classifying them using CNN and interpolation has been widely and traditionally used to scale all the images to the same size. In this project original 32x32 pictures resized and turned into 224x224 size large pictures. The motivation is to keep the image size relatively larger then original while running through residual blocks. The result of this action will be evaluated at the later stage.

Algorithm 1 TrivialAugment Procedure

procedure TA(x: image)
  Sample an augmentation a from $\mathcal{A}$
  Sample a strength m from $\{0, \ldots, 30\}$
  Return $a(x, m)$
end procedure

Figure 3: Trivial Augmentation

1.4 Augmentation

TrivialAugment (TA) works as follows. It takes an image x and a set of augmentations A as input. It then simply samples an augmentation from A uniformly at random and applies this augmentation to the given image x with a strength m, sampled uniformly at random from the set of possible strengths 0, . . . , 30 and returns the augmented image. It is visualized in Figure 3. TA is not a special case of RandAugment (RA), since RA uses a fixed optimized strength for all images while TA samples this strength anew for each image.

1.5 Normalization

Normalize does the following for each channel: Image = Image - Mean/Std. At the beginning traditional mean = [0.5, 0.5, 0.5], and standard deviation(std) = [0.5, 0.5, 0.5] values are applied but later new values mean = [0.4914, 0.4822, 0.4465], std = [0.247, 0.243, 0.261] applied. These are specific mean and std values so the model accuracy should increase from the action.

1.6 Stochastic Gradient Descent(SGD) with Momentum

SGD with momentum is method which helps accelerate gradients vectors in the right directions, thus leading to faster converging. It is one of the most popular optimization algorithms and many state-of-the-art models are trained using it. Momentum is a moving average of the gradients. It is used to update the weight of the network. This could written as follows:

$$Vt = BVt - 1 + (1 - B)\nabla wL(W, X, y) \quad (1)$$

$$W = W + Vt \quad (2)$$

With SGD one does not compute the exact derivate of the loss function. Instead, estimating it on a small batch. Which means the model is not always going in the optimal direction, because derivatives are 'noisy'. So, exponentially weighed averages can provide a better estimate which is closer to the actual derivate than noisy

calculations. This is a reason why momentum works better than classic SGD.

1.7 Weight decay

Data augmentation helps deep learning models generalize well. That was on the data side of things. What about the model side of things? What can one do while training models, that will help them generalize even better. Here one can use weight decay. To solve complex problems, one needs complex solutions which in turn means more parameters. More parameters mean more interactions between various parts of neural network. And more interactions mean more non-linearities. These non-linearities helps to solve complex problems.However, these interactions might get out of hand. Hence, what if one can penalize complexity. One will still use a lot of parameters, but will prevent the model from getting too complex. This is how the idea of weight decay came up. One way to penalize complexity, would be to add all our parameters (weights) to the loss function. That won't work because some parameters are positive and some are negative. So what if one adds the squares of all the parameters to the loss function. However, it might result in the loss getting so huge that the best model would be to set all the parameters to 0. To prevent that from happening, one can multiply the sum of squares with another smaller number. This number is called weight decay (wd).

$$Loss = MSE(y', y) + wd * sum(w2) \quad (3)$$

1.8 Cosine Annealing Learning Rate

Cosine Annealing Learning Rate (CosineAnnealingLR) is a scheduling technique that starts off with a very large learning rate and then aggressively decreases it to a value near 0, before again increasing the learning rate. This variation of the learning rate happens according to the cosine annealing schedule. Mathematically, if the learning rate is given as,

$$nt = nmin + (nmax - nmin)(max(0, 1 - x)) \quad (4)$$

where nmin and nmax define the bounds of our learning rate, iterations represents the number of completed mini-batches, stepsize defines one half of a cycle length. As far as I can gather, 1−x should always be positive, so it seems the max operation is not strictly necessary.

		image
	Input	array (size: 3 × 224 × 224)
1	ConvolutionLayer	array (size: 64 × 112 × 112)
2	BatchNormalizationLayer	array (size: 64 × 112 × 112)
3	Ramp	array (size: 64 × 112 × 112)
4	PoolingLayer	array (size: 64 × 56 × 56)
5	NetGraph (12 nodes)	array (size: 256 × 56 × 56)
6	NetGraph (10 nodes)	array (size: 256 × 56 × 56)
7	NetGraph (10 nodes)	array (size: 256 × 56 × 56)
8	NetGraph (12 nodes)	array (size: 512 × 28 × 28)
9	NetGraph (10 nodes)	array (size: 512 × 28 × 28)
10	NetGraph (10 nodes)	array (size: 512 × 28 × 28)
11	NetGraph (10 nodes)	array (size: 512 × 28 × 28)
12	NetGraph (12 nodes)	array (size: 1024 × 14 × 14)
13	NetGraph (10 nodes)	array (size: 1024 × 14 × 14)
14	NetGraph (10 nodes)	array (size: 1024 × 14 × 14)
15	NetGraph (10 nodes)	array (size: 1024 × 14 × 14)
16	NetGraph (10 nodes)	array (size: 1024 × 14 × 14)
17	NetGraph (10 nodes)	array (size: 1024 × 14 × 14)
18	NetGraph (12 nodes)	array (size: 2048 × 7 × 7)
19	NetGraph (10 nodes)	array (size: 2048 × 7 × 7)
20	NetGraph (10 nodes)	array (size: 2048 × 7 × 7)
21	PoolingLayer	array (size: 2048 × 1 × 1)
22	ReshapeLayer	vector (size: 2048)
23	LinearLayer	vector (size: 1000)
24	SoftmaxLayer	vector (size: 1000)
	Output	class

Figure 4: ResNet50-2

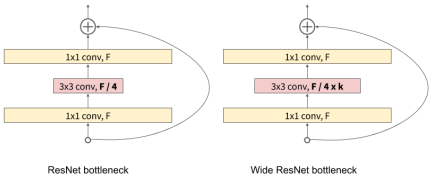

Figure 5: Wide ResNet Bottleneck

Methodology

In this project, the main goal is to receive higher accuracy than ResNet50-2 model. Secondary goal is to see the effect of every change that made throughout the experiments. Scheduled data improvement additions are changing the image size from 32 to 224, introducing data augmentation, normalization. Model improvement additions are weight decay, momentum and learning rate scheduler.

1.9 ResNet50 original model architecture

Released in 2017 by Sergey Zagoruyko and Nikos Komodakis, this model provides improvement on existing residual networks. By decreasing the depth of the architecture and increasing the width of the network, state-of-the-art accuracies and much faster training were achieved.

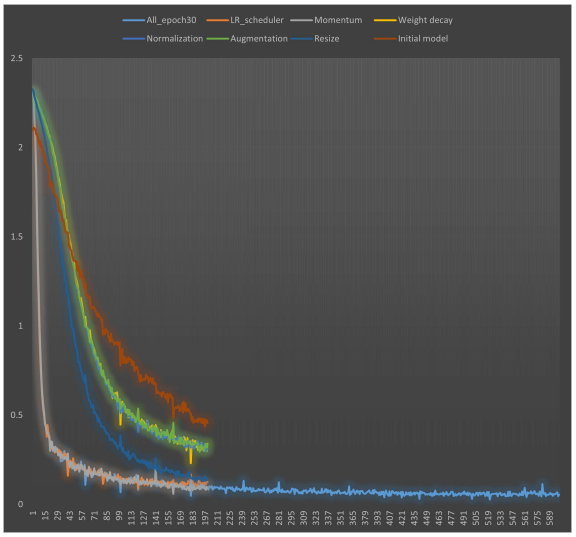

Figure 6: Loss of all models

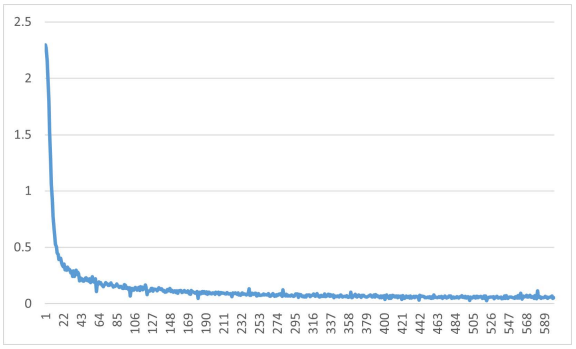

Figure 7: All additions with 30 epochs

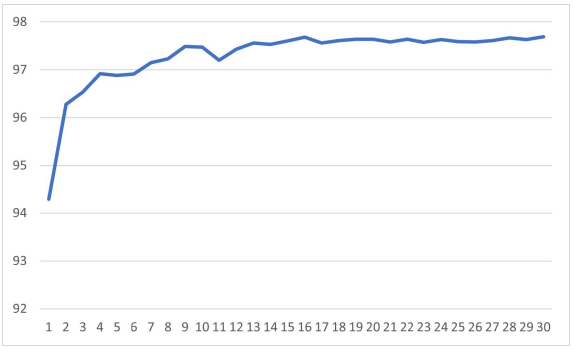

Figure 8: Test accuracy

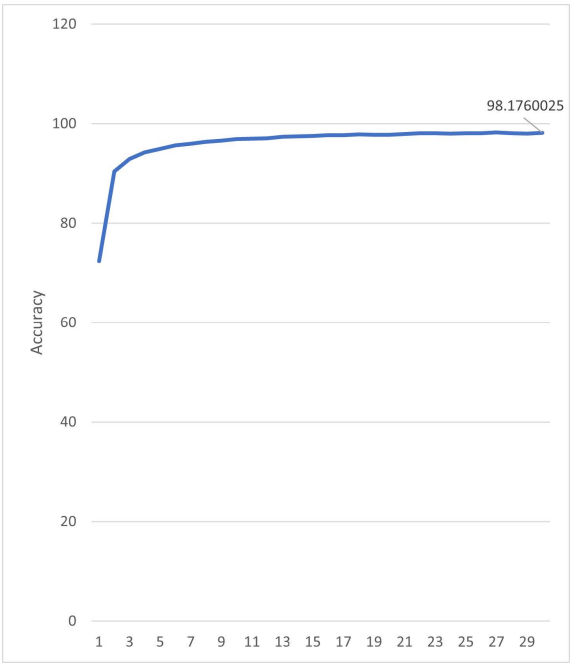

Figure 9: Train accuracy

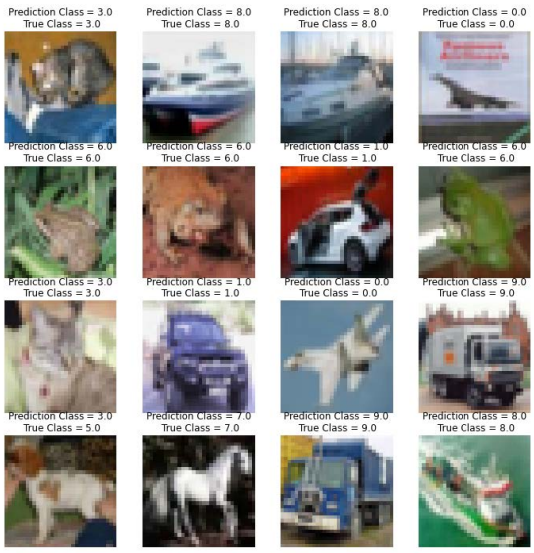

Figure 10: Correct Predictions

Results and analysis

As initially model accuracy was 10 percent. One can interpret it as model sanity check. Because the data-set CIFAR10 has 10 classes it is statistically obvious that any model without specific training can achieve 10 percent accuracy. Later when images size changed from 32 to 224 and model trained and accuracy increased more than 8 times and become 83.99 percent. Another jump of accuracy came when normalization introduced. Mean and standard deviation list for CIFAR10 dataset implemented and 90.06 accuracy achieved. Introducing momentum value 0.9 increased accuracy from 90.98 to 92.37. Finally after cosine learning scheduler did not improve accuracy number of epochs increased from 10 to 30. Until this step all models were running through 10 epochs with batch size 128. Increasing epochs affected positively and model accuracy turned to 95.02 where it stabilized. I got even higher accuracy such as 95.53 with epoch size 50 but I do not think it is right to over train model and achieve very small accuracy increase.

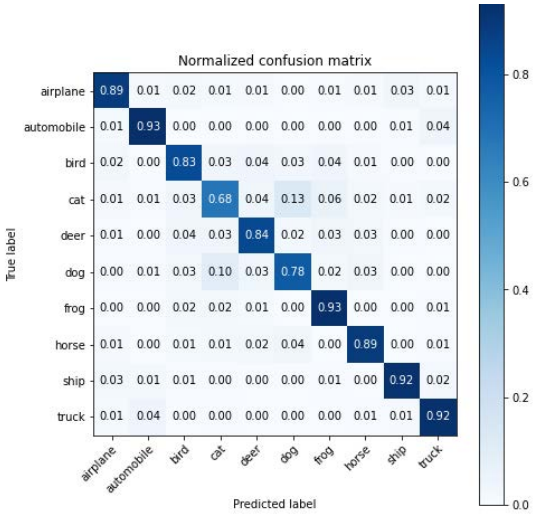

Figure 11: Normalized Confusion Matrix

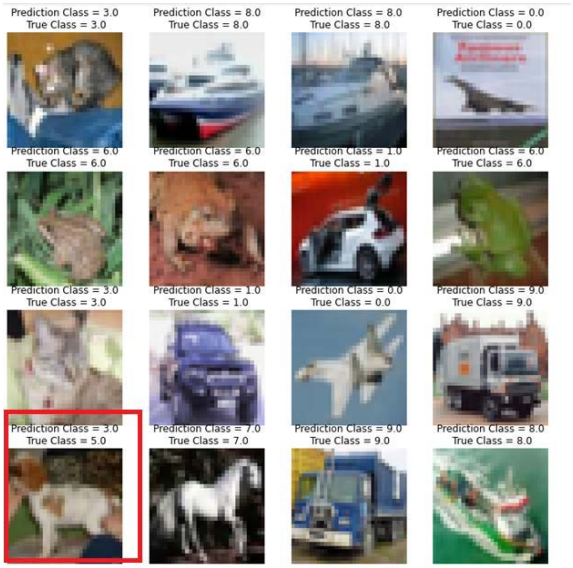

Figure 12: Wrong Predictions encircled with Red box

References

[1] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. arXiv preprint arXiv:1605.07146, 2016.