Data Augmentation and Stochastic Weight Averaging in ResNetV2

Amir Darwesh*
adarwesh@tamu.edu

Abstract

This project evaluates Data Augmentation techniques and Stochastic Weight averaging (SWA) in ResNet networks for Deep Learning on the CIFAR-10 dataset. An original (ResNet-20) and a proposed version based other works is used for comparison for SWA and data augmentation. Our best accuracy achieved 95.15% on the CIFAR-10 test set via the standard augmentation technique (flip, crop, and normalize). Although other data augmentation techniques did not succeed, a hypothesis is formed explaining potential causes for failure. For SWA, improvement gains seen from other papers were not realized - though this may be due to untuned learning rates.

1 Introduction

1.1 ResNet

The original Residual Network paper, published by Kaiming et. al. [1], utilized a shortcut function after applying non-linearity. In the updated version (ResNet-V2) [2], the shortcut function was changed such that they do not go through a non-linear transform, subsequently named "identity mappings" due to the direct connection of model inputs to outputs. Within either version of the ResNet, two flavors of residual blocks were used: (1) standard blocks, and (2) bottleneck blocks. Standard blocks were used in-cases where the network architecture was very shallow (e.g. less than 50 layers), where as bottle-neck blocks were used in deep networks due to performance draw-downs of the standard blocks.

In this project, we have focused on trying to develop improvements in different areas in ResNetV2 architecture. Improvement experiments for this project included changing the network architecture by following some previous work on GitHub[4], our own data augmentation experiments, and model generalization via Stochastic Weight Averaging (SWA) [3]. The following sections will detail the original and proposed network architectures, data augmentation experiments, and the procedure for SWA implementation in PyTorch. The design of experiments section details the procedure for testing improvement experiments, and the training procedure section provides details on training such as the learning rate, epoch stopping, training visualization, and model check-pointing. Finally, we present our training results, and end by summarizing conclusions from the experiments.

1.1.1 ResNetV2-20

ResNet-20 is a shallow architecture, presented first in [1]. Modifcations of this network include the update of Identity mappings and Pre-activation functions detailed in [2], hence the name V2. A summary of model parameters, and figure of the network architecture is presented in Table 1 and Fig. 1, respectively.

*Graduate Student, Texas A&M University. https://amirdarwesh.com

Table 1: ResNet20 Structure

Layer Name	**Output Size**	**Weight Structure**	**# of Parameters**
Conv1	$32 \times 32 \times 16$	$3 \times 3, 16$	432
ConvStack1	$32 \times 32 \times 16$	$\begin{bmatrix} 3 \times 3 \\ 3 \times 3 \end{bmatrix} \times 3$	13,824
ConvStack2	$16 \times 16 \times 32$	$\begin{bmatrix} 3 \times 3 \\ 3 \times 3 \end{bmatrix} \times 3$	51,200
ConvStack3	$8 \times 8 \times 64$	$\begin{bmatrix} 3 \times 3 \\ 3 \times 3 \end{bmatrix} \times 3$	204,800
AvgPool	$8 \times 8 \times 1$		
F.C.	10×1	65×10	650
Total			272,154

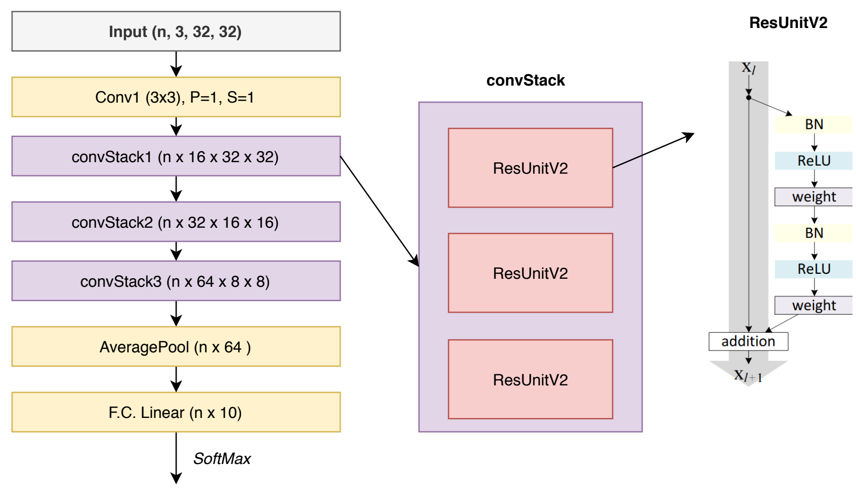

Figure 1: ResNet-20 Architecture[2]

1.1.2 ResNetV2-Proposed

After some searching on various pytorch implementations of ResNet, an interesting version was found in [4]. Since the main components of the report is data augmentation and SWA, a similar implementation is done from the GitHub. The original network code was cleaned up, restructured, and organized for code readability. Further, the authors of the original repository missed some implementation details from the ResNetV2 paper, and subsequently mis-represented the network as ResNet-18 - when infact it was more similar to ResNet-164.

Still however, the network found in [4] reported good accuracy (95.1%). The network structure utilizes a standard Resiudal Unit Block (as opposed to the BottleNeck block), and is actually quite similar to the network found in [1] used in the ImageNet classifcation network. The main difference between this implementation and ResNet-164 is that there four residual network stacks, and the final output feature map is half that of the original (4x4) vs. (8x8), respectively. Fig. 2 and Table 2 detail the network architecture and layer structures.

1.2 Data Augmentation

For image augmentation, we test four different types of augmentation techniques. For image Augmentation, we initially utilized the ImgAug package which provides support for PyTorch; however, this package added 40s of additional computation per epoch. The augmentation library utilized was then switched to one from torchvision. The augmentations are applied randomly at each batch interval, which allows the training dataset to have the original size of 50k. Fig. 3 illustrates the the training data without any augmentation.

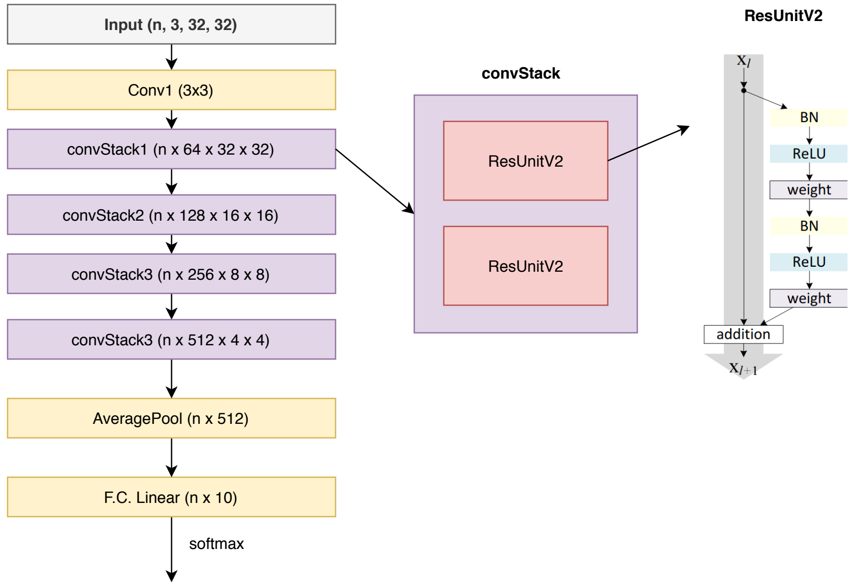

Figure 2: Proposed ResNet Architecture [2],[4]

Table 2: ResNet-Prop Structure

Layer Name	Output Size	Weight Structure	# of Parameters
Conv1	$32 \times 32 \times 64$	$3 \times 3, 64$	1,728
ConvStack1	$32 \times 32 \times 64$	$\begin{bmatrix} 3 \times 3 \\ 3 \times 3 \end{bmatrix} \times 2$	151,456
ConvStack2	$16 \times 16 \times 128$	$\begin{bmatrix} 3 \times 3 \\ 3 \times 3 \end{bmatrix} \times 2$	524,288
ConvStack3	$8 \times 8 \times 256$	$\begin{bmatrix} 3 \times 3 \\ 3 \times 3 \end{bmatrix} \times 2$	2,097,152
ConvStack4	$4 \times 4 \times 512$	$\begin{bmatrix} 3 \times 3 \\ 3 \times 3 \end{bmatrix} \times 2$	8,388,608
AvgPool	$4 \times 4 \times 1$		
F.C.	10×1	513×10	5,130
Total			11,171,146

In all techniques, we normalize each channel by subtracting the per channel mean, and dividing by the channel standard deviation, as shown in Eq. (1) below, based on the entire training dataset. For the CIFAR-10 dataset, the per channel mean and standard deviation is $[0.4914, 0.4822, 0.4465]$, and $[0.2470, 0.2435, 0.2616]$, respectively. These normalization constants are also applied to the CIFAR-10 and private testing datasets.

$$\hat{D} = \frac{(D - \bar{D})}{\sigma_D} \tag{1}$$

Table 3:

Augmentation Name	Crop	LR Flip (%)	Translate	Rotate	Scale	Shear
Standard	TRUE	0.5	N/A	N/A	N/A	N/A
Transform1	TRUE	0.5	(-10,10) %	-20,20 deg	N/A	N/A
Transform2	TRUE	0.5	(-20,20) %	-20,20 deg	(90,110) %	(-8,8)

Figure 3: 12 CIFAR 10 Images with no transformation

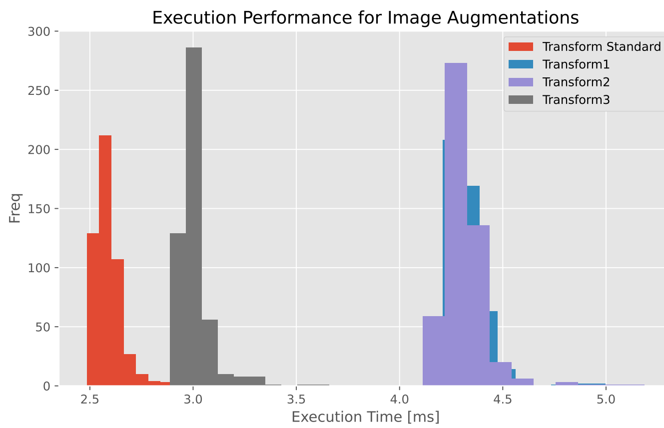

Figure 4: Execution time per batch with various image augmentation transforms using torchvision

1.2.1 Standard Augmentation

For the first augmentation technique, labeled as ImageAugmentationStandard, we apply random crops and random flip to the training dataset. This method is consistent with the original paper presented by Kaming et. Al., and hence the naming convention "standard". These transforms are illustrated in Fig. 5 below.

Figure 5: Standard Image Augmentation (crop and flip)

1.2.2 Augmentation Transform 1

For the 2nd augmentation, labeled as ImageAugmentationTranaform1, we include the standard augmentation and add a random rotation and translation. These transforms are illustrated in Fig. 6 below.

1.2.3 Augmentation Transform 2

For the 3rd augmentation, labeled as Image AugmentationTransform2, we increase the ranges of rotation and translations from the previous transform, and also include a shear transform, as illustrated in Fig. 7 below.

Figure 6: Transform 1 Image Augmentation (crop, flip, rotate, translate)

1.2.4 Augmentation Transform 3

A minor late-addition includes Augmentation Transform 3. This transform is the same as the StandardAugmentation, but includes a probability chance of 10% of any given image being converted into gray-scale. The motivation here is to have the network learn features via color independence.

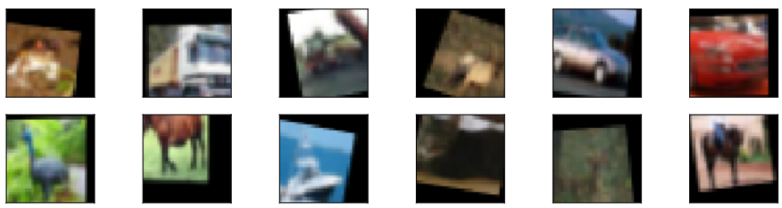

Figure 7: Transform 2 Image Augmentation (crop, flip, rotate,translate, shear)

Figure 8: Transform 3 Image Augmentation (crop, flip, and BW $p = 0.1$)

1.3 Stochastic Weight Averaging

Stochastic Weight Averaging (SWA), introduced by Izmailov .et. al., is a method that averages points on the gradient direction in Stochastic Gradient Descent to provide better model generalization [3]. The authors of SWA noted sizable increases in model performances in different networks trained on CIFAR-10. For ResNet-164, the authors were able to improve performance from 95.28 ± 0.1 to 95.56 ± 0.11 accuracies.

Starting from PyTorch 1.6, SWA is included in the optim library as swa-utils. As SWA is easy to implement and has low overhead, it was included in our Design of Experiments to try and improve model accuracy in both ResNet architectures studied in this paper. In implementation, SWA kicks in at around 75% of the training epochs. However, it was also tested with SWA kicking in at much lower epochs such as at 10%.

1.4 Code Architecture

The architecture for the code is summarized in Table 4. Mostly everything is modularized, with Network.py and Module.py serving as the main files containing the bulk of the operations. If Stochastic Weight Averaging is needed, lines 6/7 in main.py should be switched to utilize model.py or model_SWA.py.

Table 4: Summary table for code files in submission

File Name	Description
Configure.py	Configuration File that contains dictionaries defining the training Parameters. Passed to main.py.
DataLoader.py	Loads the CIFAR-10 and Private Image Testing sets. Uses torchvision for the CIFAR 10 datasets, and custom class file for private dataset. Returns torch DataSet format with optional transform definition for images. Images are (channel,width,height) (3,32,32).
ImageUtils.py	Defines Image Augmentation transform sequence. Loaded in Configure.py and passed to DataLoader.py
Network.py	Defines netork structure for ResNetV2-20 or Prop. Called in model.py / model_SWA.py and returns Torch nn.module.
Model.py	Defines training procedure and checkpoint feature. Logs training statistics to tensor board, and manages terminal text output.
Model_SWA.py	Same as Model.py, but with stochastic weight averaging for training procedure.
main.py	Main function to load all modules and run either training, testing, or prediction. Loads either Model_SWA.py or Model.py depending on which one is uncommented.
utils.py	Helper file for common functions
PrivateDataset.py	Torch dataset class to define custom private dataset. Loads the numpy array.

2 Design of Experiments

For the Design of Experiments, an initial set of experiments was conducted with training with and without Stochastic Weight Averaging for both ResNetV2-Proposed and ResNetV2-20 using the standard image transformation. For other image transformations, most were tested in both network models, with the exception of Transform 3 - which was only tested in the proposed network.

2.1 Training Procedure

In total, over 50 experiments were conducted exceeding over 150 hours of training times - though only 9 are presented in this work. Most of the experiments were initial testing with training / validation splits at 45/5k to tune training hyper-parameters. For the 9 experiments, presented in Table 5, the entire CIFAR-10 50k image training set was used, with accuracies reported from the test set. For training, a learning rates are presented in the tensorboard log file, with a batchsize of 128. Starting too high $(lr > 0.1)$ with a learning rate caused the gradients to explode, while starting too low $(lr < 0.05)$ resulted increased iterations. For logging, Tensorboard is utilized, where the loss, train/test accuracy, and learning rate is logged. . Batch sizes $256, 512, 1024$ were tested; however, though these batch sizes led to better utilization of the computational resources, the model accuracy suffered.

2.2 Computing Resources

For computing resources, two machines were primariy utilized: (1) An Alienware Laptop with GTX 1060, (2) a Lambda system with two NVidia Titan RTXs (48 GB GPU RAM available). Training generally took six hours on the 1060, and two hours on the Lambda system. An extensive analysis has not been conducted, but it appeared that the main bottleneck was the CPU capacity.

3 Results

Results for the experiments presented in Table 5. Although extensive hyper-parameter tests were done for the Data Augmentation, it appeared that the standard augmentation (crop and flip) appeared to have the best network performance at 95.15% test accuracy. Note that the shallow network ResNet-20 suffered extremely in performance with image augmentation, compared to the proposed network.

Table 5: Experimental Results after hyper-parameter testing. Model Checkpoints avaialble here

Network / Augmentation		**Log**	**Test Acc (%)**
ResNetV2-20	Standard	link	91.23
ResNetV2-20	Transform1	link	88.19
ResNetV2-20	Transform2	link	88.17
SWA-ResNetV2-20	Standard	link	91.23
ResNetV2-Prop	Standard	link	95.15
ResNetV2-Prop	Transform1	link	94.64
ResNetV2-Prop	Transform2	link	93.02
SWA-ResNetV2-Prop	Standard	link	94.64
ResNetV2-Prop	Transform3	link	94.89

4 Conclusions & Future Work

This project targeted two areas detailed in the project description:

- *"Data pre-processing, normalizations and augmentations"*, via 4 transform augmentation techniques
- *"Training strategies, optimizers, parameter initializations, regularizations, etc."* via Stochastic Weight Averaging in the training procedure

Further, we also updated a publicly available pytorch based ResNet with better documentation, better code readability, and small improvements, dubbed as ResNet-prop.

Although the expected imrpovements from SWA and image augmentation were not realized, various insights were developed throughout the course project. For data augmentation, Transforms 1 and 2 had trouble getting good accuracy due to the large amount of blacked out data caused by rotating the image while maintaining the same feature input map. Improvements to this area could include lesser rotations, such as $< 5\,\text{deg}$, or full rotations such as $90\,\text{deg}$.

For SWA, the proposed network actually suffered a degradation in performance. This may be due to the fact the learning rate adjustment for SWA was not tuned, and hence did not update the model average as precisely. Future work should include an extensive test on hyper-parameters with regards to SWA.

References

[1] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[2] Kaiming He et al. *Identity Mappings in Deep Residual Networks*. 2016. arXiv: 1603.05027 [cs.CV].

[3] Pavel Izmailov et al. *Averaging Weights Leads to Wider Optima and Better Generalization*. 2019. arXiv: 1803.05407 [cs.LG].

[4] Kuang Liu. *pytorch-cifar*. 2020. URL: https://github.com/kuangliu/pytorch-cifar.