EE-411 Reproducibility Challenge: Understanding Deep Learning Requires Rethinking Generalization

Jan Clevorn (377937), Gustavo Fonseca (374427). Ryoga Matsuo (374996) and Jérôme Mayolet (370949)
February 3. 2024, EPFL, Switzerland

***Abstract*—Generalization performance is an important measure for machine learning models which offers more insight into how they perform against unseen data points. Knowing the behavior of different models on a given dataset is crucial in choosing the best model for a specific application. In addition, this knowledge facilitates the engineering of more accurate and robust neural networks due to better interpretability. In this report, we will investigate this topic by reproducing the results of a chosen paper on machine learning.**

I. Introduction

This work embarks on a reproduction challenge of the article: "Understanding Deep Learning Requires Rethinking Generalization" [1]. The paper investigates fundamental aspects of deep learning generalization, shedding light on the limitations of classical generalization theories. More specifically, they explore the generalization performance of different machine learning models under various implicit and/or explicit regularizations.

II. Preprocessing and Pipeline

First of all, the CIFAR10 dataset containing 50000 training images and 10000 validation images of 10 classes was pre-processed. The first data transformation was center cropping applied to transform images of 32x32 pixels into 28x28 pixels while improving the richness of the dataset by adding some noise to the dataset. The second transformation was the standardization of the image dataset to normalize the features and improve the statistical stability during training. It should be noted that the authors of the paper seem to have used TensorFlow. As we have learned the Pytorch framework in the course, Pytorch was used instead to facilitate the reproduction. For the data standardization, a custom transformation function had to be implemented since the TensorFlow built-in function *per_image_whitening* is not implemented in PyTorch. This transformation independently normalizes each image by subtracting its mean and dividing by its standard deviation, ensuring that the dataset is properly standardized before training.

The overall pipeline begins with pre-processing for the training and validation datasets of CIFAR10. Then, data loaders are created to handle batching, shuffling, and parallel data loading while adhering to the chosen batch size. After which, some key functions are defined to be used for training and performance measurement. The *train_epoch* function conducts a single training epoch for a neural network model, iterating through batches of training data. It sets the model to training mode, computes forward passes to generate predictions, calculates the cross-entropy loss, computes gradients via backpropagation, and updates the model parameters using the provided optimizer. The running loss for the epoch is tracked, and the function returns the average training loss. The *fit* function trains the model for a specified number of epochs by repeatedly calling *train_epoch*, optionally recording validation losses and accuracies, and adjusting the learning rate if a scheduler is provided. The *predict* function evaluates the model performance on a test or validation dataset, calculating the average test loss and accuracy, and optionally displaying the statistics.

III. The models

All the figures were reproduced using three different models. For all experiments on CIFAR10, the training was achieved by SGD with a momentum=0.9, at a decay factor of 0.95 per training epoch, and an initial learning rate of 0.1 for Small Inception or 0.01 for Small AlexNet and MLPs.

A. Small Inception

The first model is a simplified Inception model, for which all the different layers of implementation were schematically given in Figure 1.

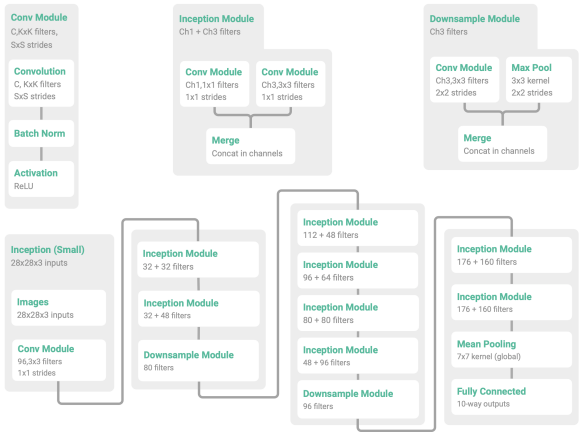

Figure 1. The small Inception model adapted for the CIFAR10 dataset, adapted from [1]

B. AlexNet

The SmallAlexNet model is designed with two distinct blocks. The first block, comprising two convolutional layers with 5x5 kernels followed by max-pooling using 3x3 windows and local response normalization, is responsible for feature extraction. These layers help detect low-level features such as edges and textures in the input data. The second block called the "classifier" block, follows the feature extraction block and consists of two fully connected layers with 384 and 192 hidden units, respectively. These layers process and transform the high-level features learned by the convolutional layers. Finally, a 10-way linear layer is used for classification, enabling the model to make predictions across the ten different classes.

C. MLP 1x512

The last model is an MLP 1x512. It consists of three layers: an input layer, one hidden layer with a ReLU activation function, and an output layer. The input size is determined by the product of the dimensions (28x28x3) of the input data, with the hidden layer having 512 neurons. The output layer predicts the 10 output units. During the forward pass, the input data is flattened and passed through the hidden layer with a ReLU activation function.

IV. Figure 1

Figure 1 of the paper [1] measures the generalization performance of the three models. Some unspecified hyperparameters such as batch size introduced some difficulties in reproducing the results. Therefore, these values were estimated from the results or simply chosen arbitrarily.

A. Figure 1a: Learning curves

The number of training steps is defined by

$$N_{step} = \frac{N}{B} \cdot N_{epoch} \tag{1}$$

where N, B, and N_{epoch} are the number of samples in the training dataset, batch size, and the number of training epochs respectively. In the training dataset of CIFAR10, we have $N = 50000$. From Figure 1a of the paper, the sixth datapoint starting from the datapoint at step 0 is seen at step 5000. By setting $N_{epoch} = 6$ and $N_{step} = 5000$, the batch size B can be calculated to be 60. This seems to provide a good balance between computational speed and statistical stability in convergence.

While already described in the paper, we would like to provide more details on the modifications of the labels and input images that we implemented.

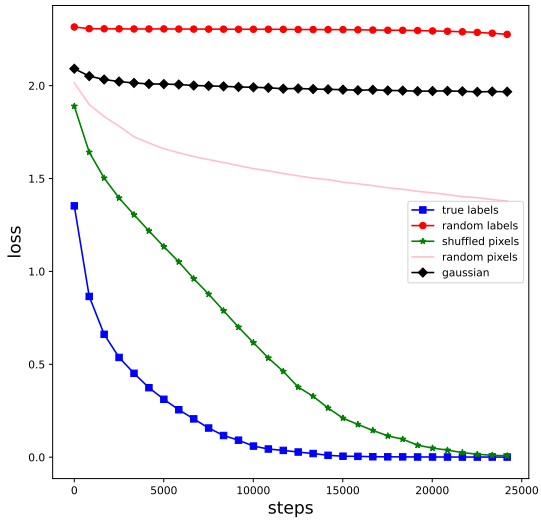

Figure 2. Attempted reproduction of Figure 1a [1] (running time: 1h40)

1) True labels: the original dataset without modification

2) Random labels: all the labels are replaced with random ones (uniformly)

3) Shuffled pixels: a random permutation of the pixels[1] is chosen and this permutation is applied to all images.

4) Random pixels: a different random permutation is applied to the pixels[2] of each image independently.

5) Gaussian: each pixel of each image is redrawn from a Gaussian distribution. Three means and variances are calculated for each channel of each image **from the initial dataset**, from which each of the RGB channels is resampled.

The reproduction of Figure 1a is shown in Fig. 2. The average losses at low training steps are similar to the ones seen in the paper. However, there are some differences in the speed of convergence to zero loss. Additionally, some initializations resulted in no convergence. This could be due to the difference in the hyperparameters used for the training. For example, if the batch size is chosen smaller for the same learning rate and the mean of each batch is close to the mean of the whole dataset, the model converges faster. Furthermore, we could also increase the number of training steps until convergence. However, these solutions were not implemented given the limited computational resources and the project duration.

B. Figure 1b: Convergence slowdown

In our attempt to reproduce Figure 1b, we transformed the data such that the image labels are corrupted with a probability of p, and each corruption is defined by a uniform probability between 0 and 9.

The reproduced paper measured the relative number of epochs that took to achieve overfitting for different values

[1] All 3 layers of the same image are permuted the same way.

[2] Same remark as before

of p. Here, they defined overfitting as when the training loss reaches zero. However, the complete convergence requires an extremely long time when labels are corrupted as was shown in Figure 2. With the provided computing resources, it is impossible to train such models within a reasonable time. Therefore, we applied alternative measures to reproduce Figure 3 instead of using zero training loss. One of the alternatives was to set a training loss threshold slightly higher than zero (we tried $0.1, 0.25, 0.5, 0.6$). The other alternative was to measure the time spent for the models to reach a pre-defined training prediction accuracy. This target accuracy was arbitrarily set to be 98%. Even with these modifications, the code execution took extremely long.

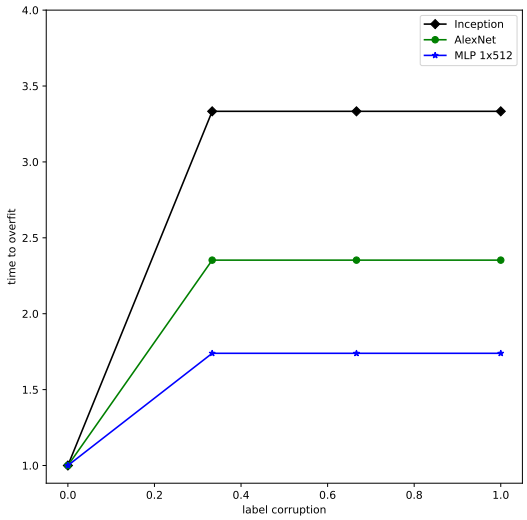

Figure 3. Attempted reproduction of Figure 1b (execution time: 5h)

Therefore, to prevent the endless executions of our models, we set the maximum number of epochs to 40. As a result, using the target accuracy threshold, the curves reach the maximum epochs for all $p > 0$: Figure 3. Here, the time to overfit is a relative value of the actual number of epochs normalized by the initial number of epochs on the uncorrupted dataset. As the number of epochs without corrupted labels is lower for Inception as compared to AlexNet and MLP, it shows a greater time to overfit. However, we would like to emphasize the fact that this result is invalid. While the original paper ran all models until the training loss reached 0, we had no choice but to stop the training way before the convergence. This is the reason why the graphs between AlexNet and MLP are inverted compared to the paper [1].

C. *Figure 1c: Generalization error growth*

To generate Figure 1c, the trained models from the previous section (with different corruption levels) were evaluated on the test set. It should be noted that our models *did not fully converge* to zero training losses as they hit the maximum number of epochs.

Interestingly, while we didn't reach full convergence, the order of relative test error between the three models is the same as in the paper (MLP $>$ AlexNet $>$ Inception) and follows the same rising trend. Intuitively, this result is somewhat expected since they were trained for the same number of epochs. In this fair competition between the models, the most robust model (Inception) was shown to provide the best generalization performance.

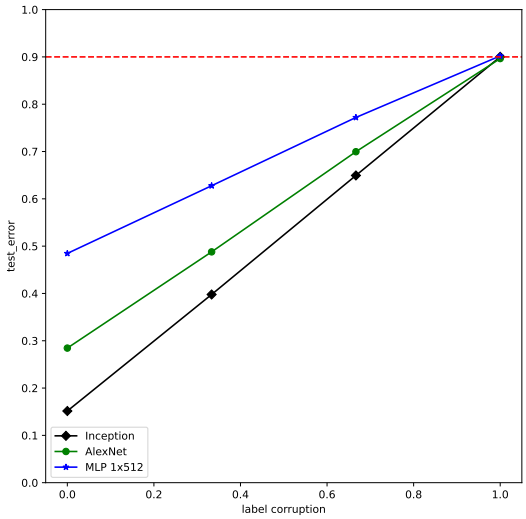

Figure 4. Attempted reproduction of Figure 1c (execution time: 5h)

V. Figure 2: The Role of Regularizers

As in the previous sections, the same lack of guidance on the hyperparameters is present. The same definition of *batch size*, *step*, and *epochs* will be used from section IV.

A. *Figure 2a: Learning curves with implicit regularizers*

Figure 2a assumes the ImageNet dataset. Since ImageNet is too large, we used CIFAR-10 for the replication attempt to study the effect of regularization.

There were 3 regularizers that were used in figure 2a: **Data augmentation**, **Weight decay**, and **Dropout**. We used 100 epochs for these models as adding implicit regularizers causes the model to take more epochs to converge.

Data augmentation (**aug**) refers to the augmenting of the dataset before sending to the model. In the paper, the augmentations specifically mentioned were: random cropping, brightness, saturation, hue and contrast. Within the recreation, we used all of these except hue as there were compatibility issues with the CIFAR-10 dataset. In addition, we also used random horizontal flipping to add more regularization. As in the paper, we reduced the image size to 28x28; thus, we set the output of the random cropping to the same size. We used a random perturbation range of 0.9 to 1.1 for brightness, saturation, and contrast. For random horizontal flipping we used a probability of 50%.

Weight decay (**wd**) refers to using the ℓ_2 regularizer on the weights within the SGD optimizer. This value was not clearly stated within the paper. Since common practice calls for a value between 10^{-2} and 10^{-4}, we used 10^{-3} [2].

Dropout (**dropout**) refers to a method that masks out each element of a layer output randomly with a given dropout probability. Since the exact probability was not specified within the paper, we defined a probability of 5% for each element of an inner layer output to be masked.

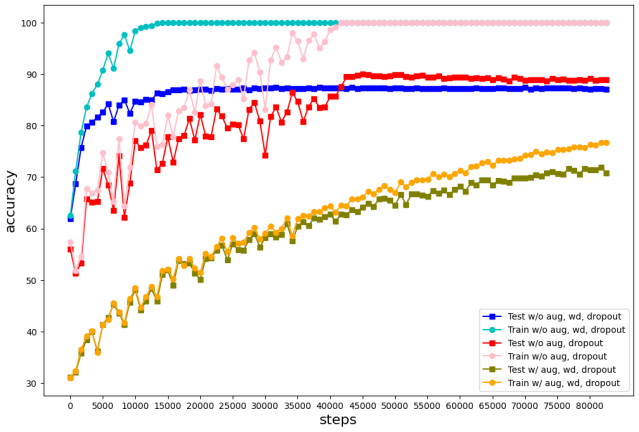

Figure 5. Attempted reproduction of Figure 2a (execution time: 5h)

The attempt for the reproduction of Fig. 2a is shown in Fig. 5. As in the paper, there are 3 different models being used: one without any regularizers (m_1), one with only weight decay (m_2), and one with all 3 regularizers (m_3).

When comparing the performance of the m_2 against m_1, we were successfully able to reproduce the same behavior that paper had: a model with implicit regularizers may perform slightly better. More precisely, after 50 epochs, the test accuracy of m_2 (88.9%) becomes higher than the accuracy of m_1 (87.2%). However, m_2 takes significantly longer to converge, a behavior also seen in the paper.

Finally, the performance of m_3 is inconclusive. As seen in the graph, neither the test or training accuracy of m_3 have saturated even after 100 epochs. Due to limited computing power, we were not able to train the model up to convergence. Given more epochs, the test accuracy of m_3 may increase past the test accuracy of m_1 and m_2.

B. *Figure 2b: Impact of Batch Normalization on Learning*

To recreate Figure 2b, we used CIFAR-10 as in the paper. The figure shows the impact of batch normalization (BN) on accuracy. BN refers to normalizing each layer response per mini-batch. We used 50 epochs for these models as this gave an ample amount of time to converge.

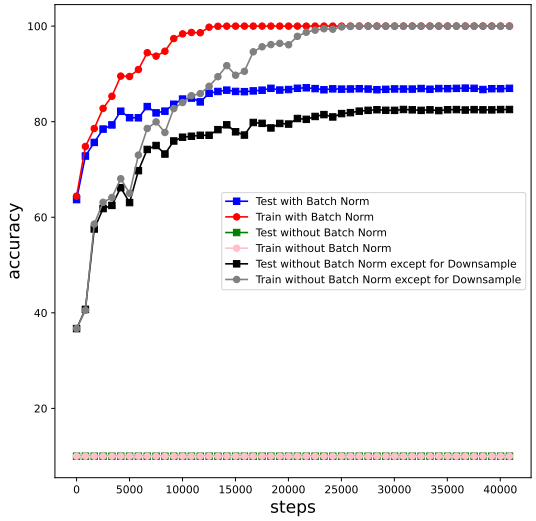

Figure 6. Attempted reproduction of Figure 2b (execution time: 3h)

When trying to reproduce the Inception model without BN, we observed irregular behavior: both test and training accuracies were at the random chance value of 10%.

Since the paper was not clear how BN was applied, we experimented with a third model, in which BN is applied only on the 2 downsample modules.

Using the third model, we were able to observe similar results as paper: while it reported that BN improved the test accuracy over the no-BN model by 1.5%, our third model showed improvement by 2%. This suggests that the author may have only applied no-BN to certain modules.

VI. Conclusion

We have seen in this work that the reproduction of the article is considerably limited by the availability of computing power, and the lack of information about certain hyperparameters.

However, in spite of this, we were able to observe the main conclusions of the paper:

Regarding Randomization tests, we observed, as in the paper, a rapid convergence to zero training loss, for the true labels and random labels case. However, for the other data transformation experiments carried out in the paper, our computational limitations did not allow us to observe the same results.

Regarding regularization, as the paper had concluded, adding regularizers does not provide significant improvement. Therefore, they are likely not the fundamental reason for generalization power of the Deep Learning.

References

[1] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," 2017.

[2] E. Lobacheva, M. Kodryan, N. Chirkova, A. Malinin, and D. Vetrov, "On the periodic behavior of neural network training with batch normalization and weight decay," 2022.