ResNet18 Efficacy Across Normalization Techniques, Label Noise, and Various Datasets

Authors: Yashika Batra, Anisha Chatterjee

Date: December 2023

Abstract

The goal of this paper is improve the performance of ResNet18 [3] on diverse datasets. Prior research shows that the normalization layer can have a strong impact on accuracy, efficiency, and training stability. By replacing the BatchNorm layer in ResNet18 with ConvNorm layer [1], a normalization method which is implemented within the Convolution Layer itself, we hoped to see test accuracy improve across our 2 chosen datasets: CIFAR10 and TinyCOCO. We also examined model performance under different label noise settings. While BatchNorm outperforms ConvNorm in accuracy, ConvNorm still shows significant benefits, such as label noise robustness and training stability, making it an important contender amongst normalization techniques.

1 Introduction

Within Convolutional Neural Networks (CNNs), normalization layers are typically used to avoid unfair input weightage and improve performance on difficult optimization tasks. The different input features can potentially fall under different ranges and this can lead to certain inputs receiving more weight simply because their values are larger. Training models on datasets such as this will take significantly longer to train as gradient descent takes longer to converge when input features do not all fall on the same scale. Additionally, abnormally high values can propagate throughout the network. This can lead to an accumulation of large error gradients (problem of exploding gradients) which makes the training process vary. To avoid issues with longer training time and instability, processing the input data using normalization can be helpful to transform the data to the same scale.

Currently, there are many different normalization strategies such as BatchNorm, LayerNorm, InstanceNorm, and GroupNorm among others [4]. These different strategies depend on the the layer or stage at which they normalize the elements, as well as the dimensions in which they normalize the elements. This paper explores a novel normalization technique called ConvNorm. We run all of our experiments on two datasets, CIFAR10 and a smaller version of COCO, Tiny COCO. While our current experiments show that BatchNorm outperforms ConvNorm in terms of test accuracy, ConvNorm's robustness to label noise, as well as its increased training stability show that this technique has significant benefits when compared to its popular peer, BatchNorm.

2 Background

BatchNorm is considered the standard method for normalization. This layer normalizes the elements among dimensions (N, H, W) where N corresponds to batches, H to height, and W to weight [2]. The normalization occurs independently of C where C corresponds to channels. This results in C normalization units. Though BatchNorm is commonly used, this technique breaks the independence assumption among samples in a batch. In certain settings, the effectiveness of BatchNorm is then dependent on the batch size.

This paper attempts to replace the typical BatchNorm in ResNet50 with a ConvNet Normalization [1] instead. It is easier to implement as it does not require any hyperparameter tuning or heavy computation. This method also improves network robustness and training.

3 Related Work

This paper uses the idea of ConvNet normalization. The math involves preconditioning and reparameterization. Using a circulant matrix of kernel a $\in R^n$, a circular convolution can be defined: $C_a := [s_0[\mathbf{a}]s_1[\mathbf{a}]...s_{m-1}[\mathbf{a}]]$. This can be reduced using the Fourier transform of a: $C_a = F * diag(a^1)F, a^1 = Fa$.

The idea beyond this step is to normalize using preconditioning. This is meant to elim-

inate bad local minimums; which would improve optimization and efficiency. To do this, multiply a preconditioning matrix P = $(C_a C_a^T)^{-1/2}$. The end result gives a reparameterization of C_a. Reparamaterization typically uses computationally expensive methods such as matrix inversion. However, ConvNorm still works efficiently as it uses the FFT method. This significantly reduces the complexity from $O(n^3)$ to $O(nlogn)$. Expanding this idea to k channels, the output can be normalized by $P_k = (\sum_{j=1}^{C_I} C_a C_a^T)^{-1/2} = (A_k A_k^T)^{-1/2}$

4 Methods

This paper examines ResNet18 with BatchNorm, ConvNorm, and a combination of BatchNorm and ConvNorm. We trained the models on three different datasets and three levels of label noise (0%, 10%, and 20%). We wanted to see how each of these normalization methods would perform alone and if combining them allowed the other to enhance in any way. All experiments were run with 0.9 momentum, CrossEntropyLoss, 90-10 Test/Validation split, and Batch Size 128. The model architecture is comprised of a ReLU activation layer, convolutional layer, linear layer, and an optional ConvNorm later.

4.1 CIFAR10.

We used the builtin Torch CIFAR10 dataset to train these models. The dataset was then normalized according to CIFAR10 channel means and standard deviations. We further used a learning rate optimizer that steps at epochs 40 and 80, with a starting learning rate of 0.1. This dataset was trained for 100 epochs.

4.2 COCO.

Rather than using the complete dataset, our paper uses the tiny COCO dataset given in 10-417/10-617 Homework 2. The data was normalized according to the channel means and standard deviations of the tiny COCO dataset. Because the learning rate optimizer was not reaching convergence with a default starting value of 0.1, we tested different values of 0.05, 0.025, 0.01 for training 100 epochs. The best value of the three values we tested was 0.025, and we used this to train for 200 epochs with the optimizer stepping at epochs 50, 100, and 150.

5 Results

5.1 Overall Results

BatchNorm outperforms ConvNorm pretty much everywhere, but ConvNorm does have more stable training. It is possible that with more time or computational resources ConvNorm would also be able to perform as highly as BatchNorm. However, even in the cases where we do both BatchNorm and ConvNorm, ConvNorm seems to be bringing BatchNorm accuracies down.

Note: All train/test losses and accuracy plots can be found in Appendix I.

5.2 CIFAR 10: Label Noise 0%, 10%, 20%

BatchNorm outperformed both ConvNorm and BatchNorm + ConvNorm for all three label noise settings on the CIFAR10 dataset, with a high test accuracy of 94.5% with 0% label noise, 87.8% with 10% noise, and 81.5% with 20% noise. ConvNorm and BatchNorm + ConvNorm performed at around 73% accuracy for 0% label noise, 68% for 10% label noise, and at around 63% for 20% noise. [Figures 1, 2, 3]

However, regardless of actual accuracies, we found that ConvNorm, as well as the combination of BatchNorm and ConvNorm, seemed to have the most robustness to label noise; while the BatchNorm accuracy plot shows a dip when we compare the 20% noise training run to the 10% and 0% runs, the ConvNorm and BatchNorm + ConvNorm accuracy plots show similar accuracy levels and training patterns, regardless of noise.

It's also important to note that there are some irregularities, or what seem to be jumps, within the plots. These are due to our learning rate optimizer; at epochs 40 and 80, respectively, it reassessed learning rates, which changes our training patterns, and therefore our testing accuracy patterns.

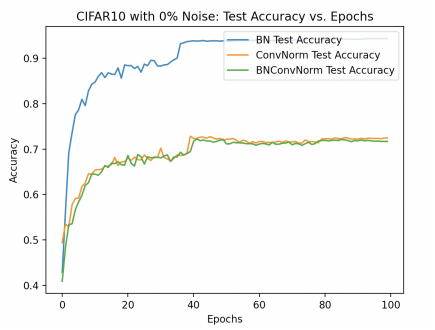


Figure 1

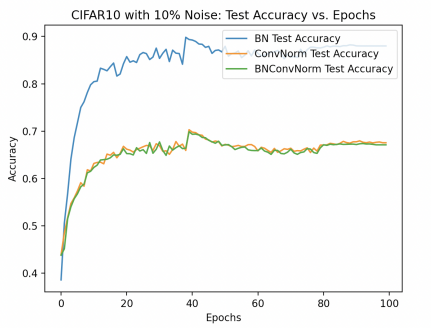


Figure 2

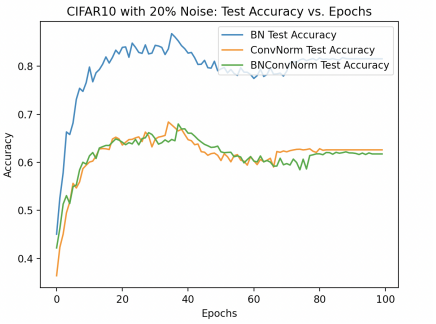


Figure 3

5.3 COCO: Label Noise 0%, 10%, 20%

We observe similar patterns between COCO and CIFAR10; once again, BatchNorm outperformed both ConvNorm and BatchNorm + ConvNorm for all three label noise settings, with a high test accuracy of 79.9% with 0% label noise, 70.6% with 10% noise, and 58.6% with 20% noise. On the other hand, ConvNorm and BatchNorm + ConvNorm performed at around 45% accuracy for both 0% and 10% label noise, and at around 36% for 20% noise. [Figures 4, 5, 6]

In general, however, testing accuracy was lower, even after running our models for 200 epochs, on the tiny COCO dataset. This can likely be attributed to the fact that this dataset was smaller than CIFAR10, and therefore had less material for the model to train on and learn from.

Regardless of actual accuracies, however, we found once again that ConvNorm, as well as the combination of BatchNorm and ConvNorm, seemed to have the most robustness to label noise. Furthermore, we also see much lower noise in testing accuracy levels between epochs with BatchNorm and ConvNorm. [Figures 4, 5, 6].

The jumps related to our learning rate optimizer, which steps at epochs 50, 100, and 150, are much more pronounced with BatchNorm in the COCO dataset than ConvNorm, or BatchNorm + ConvNorm. This may indicate that gradients normalized by ConvNorm are more resistant to learning rate optimization. This can be useful to avoid issues with gradient explosion and erasure. However, in this case, we just see a slower training trend.

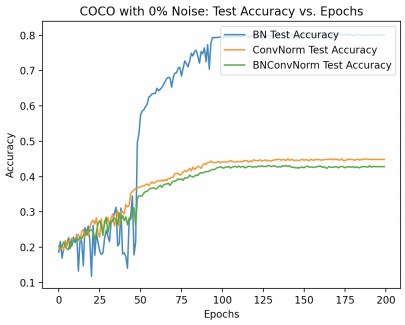


Figure 4

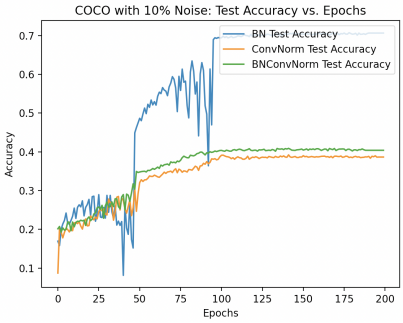


Figure 5

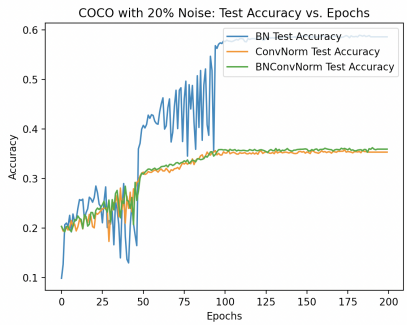


Figure 6

6 Discussion and Analysis

Overall, BatchNorm performed better than ConvNorm for most experiments, though ConvNorm has a much more stable test accuracy, which is more robust to label noise on average. Though ConvNorm has potential to perform better than BatchNorm, our experiments were not able to show these results; it is possible that we can achieve similar results to BatchNorm given more training time, or perhaps given larger training datasets.

Training time and computational resources, as well as dataset size, as such posed significant limitations to our work. Further work could explore different variations of the baseline model, ResNet, or compare against other normalization techniques. For example, examining results on ResNet50 or comparing ConvNorm against LayerNorm.

7 Resources

Google Drive: ConvNorm *
Google Drive: ConvNorm-Results
PyTorch: CIFAR10 Dataset
Google Drive: CIFAR10 Dataset
Google Drive: Small COCO Dataset Pickle File
Google Drive: Small ExDark Dataset Pickle File

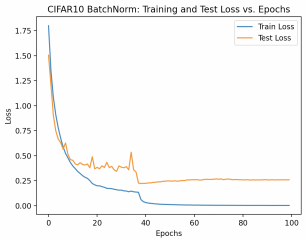
Note: Notes on ConvNorm, as well as how to execute a given model on a given dataset, are in appendix II.

8 References

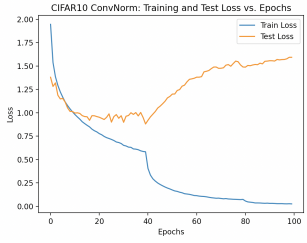
[1] ConvNorm Paper: arXiv:2103.00673
[2] Batch Norm Original Paper: arXiv:1502.03167
[3] ResNet50 Original Paper: arXiv:1512.03385
[4] Github Repository of "Awesome Normalization Techniques"

9 Appendix I: Plots

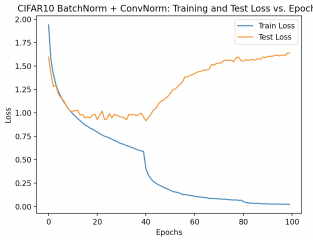
9.1 Table 1: CIFAR10 Label Noise 0%



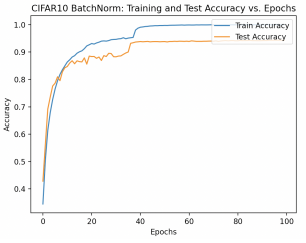
BatchNorm Losses



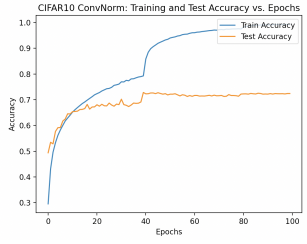
ConvNorm Losses



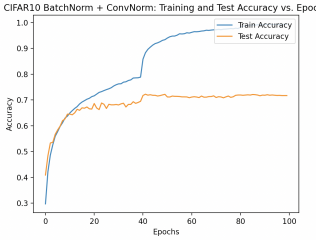
Batch/ConvNorm Losses



BatchNorm Accuracies

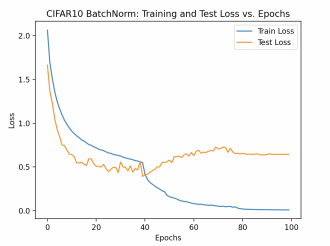


ConvNorm Accuracies

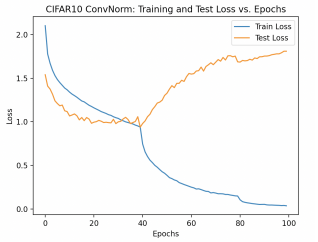


Batch/ConvNorm Accuracies

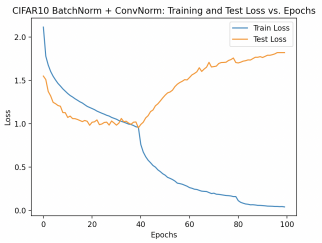
9.2 Table 2: CIFAR10 Label Noise 10%



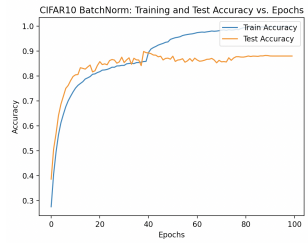
BatchNorm Losses



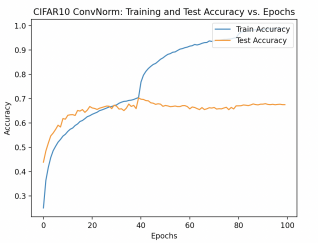
ConvNorm Losses



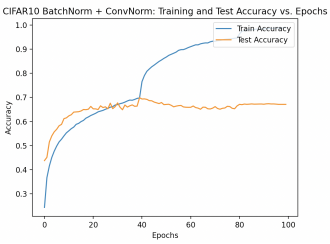
Batch/ConvNorm Losses



BatchNorm Accuracies

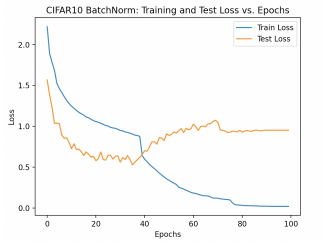


ConvNorm Accuracies

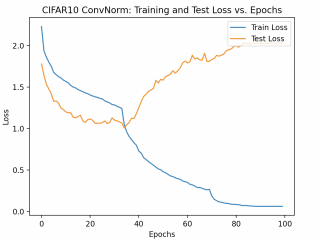


Batch/ConvNorm Accuracies

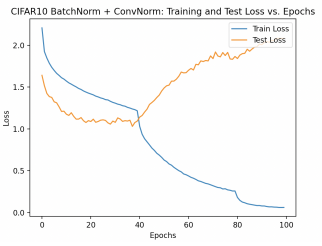
9.3 Table 3: CIFAR10 Label Noise 20%



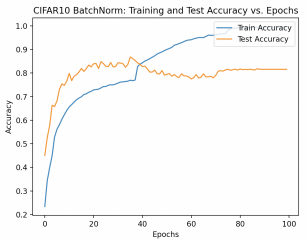
BatchNorm Losses



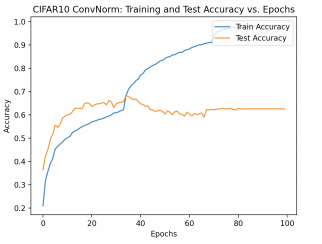
ConvNorm Losses



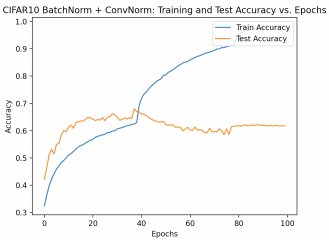
Batch/ConvNorm Losses



BatchNorm Accuracies

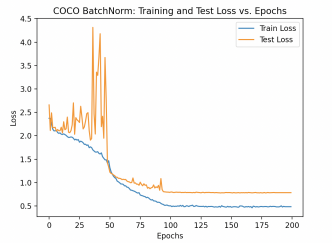


ConvNorm Accuracies

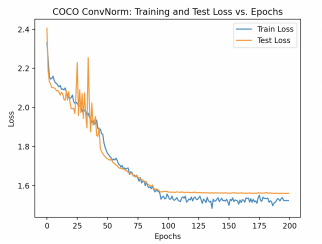


Batch/ConvNorm Accuracies

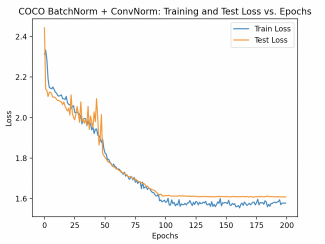
9.4 Table 4: COCO Label Noise 0%



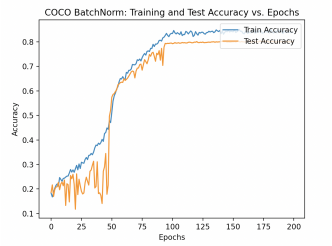
BatchNorm Losses



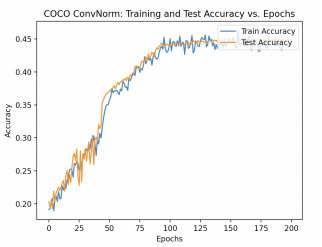
ConvNorm Losses



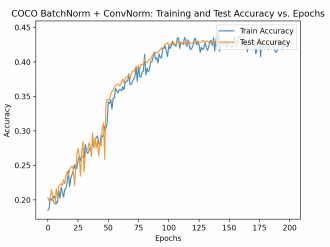
Batch/ConvNorm Losses



BatchNorm Accuracies

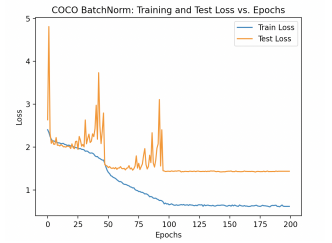


ConvNorm Accuracies

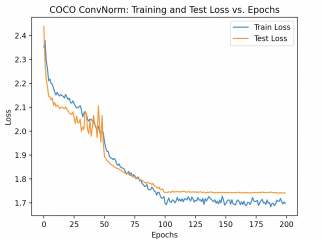


Batch/ConvNorm Accuracies

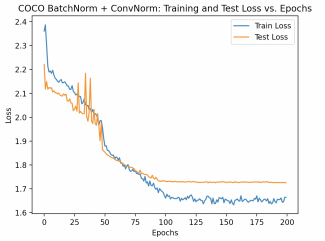
9.5 Table 5: COCO Label Noise 10%



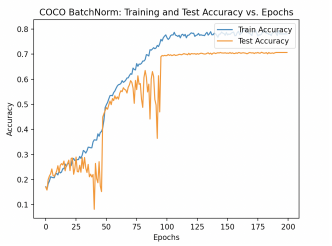
BatchNorm Losses



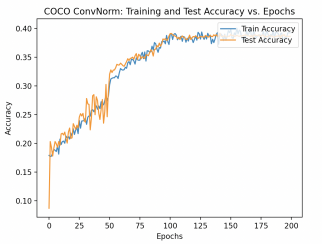
ConvNorm Losses



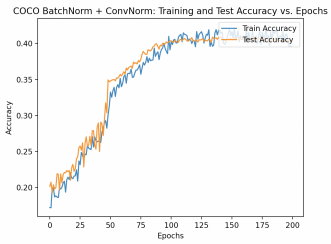
Batch/ConvNorm Losses



BatchNorm Accuracies

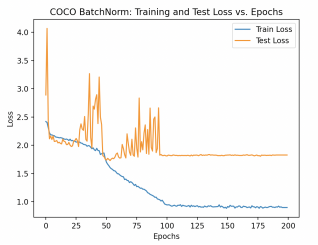


ConvNorm Accuracies

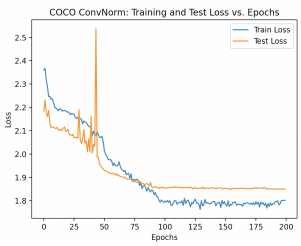


Batch/ConvNorm Accuracies

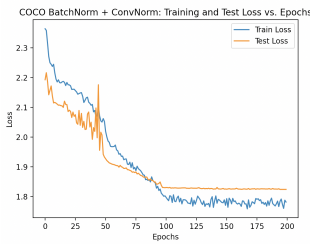
9.6 Table 6: COCO Label Noise 20%



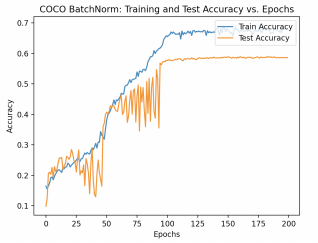
BatchNorm Losses



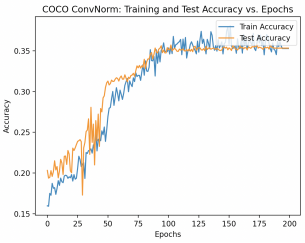
ConvNorm Losses



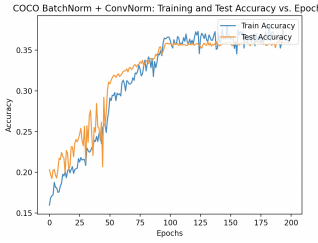
Batch/ConvNorm Losses



BatchNorm Accuracies



ConvNorm Accuracies



Batch/ConvNorm Accuracies

10 Appendix II: Code Base

10.1 ConvNorm

1. base: This folder contains all super classes (data loader, model, traimer)

2. cifar-10-batches-py: This is the directory the CIFAR10 dataset is downloaded to, or stored in. In the case of our Google Drive folder, the dataset is already downloaded and stored.

3. colab_files: This folder contains all colab files we used to train each model. Please refer to the section *Training a Model on a Given Dataset* for more information on how to run models.

4. config_*: These folders contain the configuration files for each dataset, model, label noise combination. In order to train a given model on some dataset, we must pass in the correct configuration file. However, this is abstracted away by our colab_files folder; please refer to section *Training a Model on a Given Dataset* for more information.

5. data_loader: This folder contains the code that parses each of our datasets, and creates dataloaders for training purposes.

6. logger: This folder is untouched from the original version; this contains all our logging utilities.

7. models: All activation layers, loss and accuracy metrics, and models are in this folder, in addition to the ConvNorm layer* definition
   *Note that the original version of ConvNorm was implemented with PyTorch 1.6.0,

so we had to reimplement chunks of preconv.py in order to support newer versions of PyTorch and Torchvision.

8. trainer: We kept the normal trainer, but not the advanced trainer, from the original code.

9. utils: This folder is untouched from the original version; this contains additional utilities

10. 10417-coco.pkl: TinyCOCO dataset, given to us in Homework 2.

11. exdark-small.pkl: TinyExDark dataset, generated by sampling across 10 classes from the original ExDark dataset.

12. parse_config.py: Parses our configuration json to get the specified training utilities

13. train.py: Main file that we execute in order to train a model

You can find the original ConvNorm Github repository, which implements ConvNorm and BatchNorm for CIFAR10, at this link.

10.2 Training a Model on a Given Dataset

Step 1. Access this folder on Google Drive
Step 2. There are 3 components to file names:

a. Dataset Name: Select a subfolder based on the dataset you'd like

b. Model Name: BatchNorm is denoted as "bn", ConvNorm as "conv", and models that combine both normalization techniques are denoted with "bncn".

c. Label Noise Level: Noise levels are denoted as "[percentage]noise"

For example, if you wanted to access the COCO dataset, trained with label noise 10%, on a ResNet18 architecture that uses both BatchNorm and ConvNorm, you would run "coco/coco_bncn_10noise.ipynb"

In general, the file you would run to train some given dataset with some noise percentage on a specific normalization technique is: "[dataset]/[dataset]_[norm reference]_[noise percentage]noise.ipynb"