Group Normalization

E4040.2023Fall.NNDL.report
Harinder Singh Mashiana hm3008, Yunhao Luo yl5444, Zongyu Li zl3372
Columbia University

Abstract

This project aims to reproduce the results of the paper "Group Normalization" by Yuxin Wu and Kaiming He [1]. The goal is to evaluate the effectiveness of Group Normalization (GN) in comparison to Batch Normalization (BN), Layer Normalization (LN) and Instance Normalization (IN) on the Tiny ImageNet-200 dataset and model architecture we created. We followed a structured approach to reproducing the results, including searching for relevant literature, understanding the methodology of the original paper, implementing the model architecture and training the model on the dataset, reproducing the results, and comparing the results with the original paper. Our results showed similar relative positions of curves when compared to the original paper, although they showed very different trends, which should be the result of using different models and datasets. Group normalization was shown to have the best performance in our results, which corresponds to the result in the paper. However, layer normalization is the one with the worst performance in our results but instance normalization is the worst performing one in the paper. This could also be caused by the difference in data and model architecture.

1. Introduction

The paper "Group Normalization" by Yuxin Wu and Kaiming He [1] presents a simple alternative to Batch Normalization (BN) called Group Normalization (GN) for normalizing activations in deep neural networks. BN is a milestone technique in the development of deep learning, enabling various networks to train. However, normalizing along the batch dimension introduces problems — BN's error increases rapidly when the batch size becomes smaller, caused by inaccurate batch statistics estimation. This limits BN's usage for training larger models and transferring features to computer vision tasks including detection, segmentation, and video, which require small batches constrained by memory consumption. In this paper, the authors present GN as a simple alternative to BN. GN divides the channels into groups and computes within each group the mean and variance for normalization. GN's computation is independent of batch sizes, and its accuracy is stable in a wide range of batch sizes. On ResNet-50 trained in ImageNet, GN has 10.6% lower error than its BN counterpart when using a batch size of 2; when using typical batch sizes, GN is comparably good with BN and outperforms other normalization variants. Moreover, GN can be naturally transferred from pre-training to fine-tuning. GN can outperform its BN-based counterparts for object detection and segmentation in COCO, and for video classification in Kinetics, showing that GN can effectively replace the powerful BN in a variety of tasks. GN can be easily implemented by a few lines of code in modern libraries.

The goal of our project is to reproduce the results of the paper "Group Normalization" [1]. The objective is to evaluate the effectiveness of GN in comparison to BN and other normalization techniques such as Layer Normalization (LN) and Instance Normalization (IN) on the Tiny ImageNet-200 dataset [4] and model architecture we created.

The technical challenges and difficulties of this project. To approach and solve these challenges, we will follow a structured approach to reproducing the results, including searching for relevant literature, understanding the methodology of the original paper, implementing the model architecture and training the model on the Tiny ImageNet-200 dataset, reproducing the results, and comparing the results with the original paper. By following these steps, we hope to gain a better understanding of the effectiveness of GN and its potential applications in various computer vision tasks.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

The paper [1] introduces Group Normalization (GN) as an alternative to Batch Normalization (BN) for normalizing activations in deep neural networks. The authors argue that BN's performance is limited by its dependence on batch size, which can lead to inaccurate statistics estimation when the batch size is small. GN divides the channels into groups and computes the mean and variance for normalization within each group. This computation is independent of batch size, making GN more stable and accurate than BN in a wide range of batch sizes. The authors demonstrate that GN outperforms BN and other normalization variants in various computer vision tasks, including object detection, segmentation, and video classification.

The authors first describe the limitations of BN and the motivation behind GN. They then present the details of GN, including the group division, normalization computation, and the implementation of GN in convolutional neural networks. The authors also compare GN with BN and other normalization methods in terms of accuracy and computational efficiency. Finally, the authors evaluate GN on various computer vision tasks, including object detection, segmentation, and video

classification, and demonstrate its superiority over BN and other normalization methods.

2.2 Key Results of the Original Paper

The authors demonstrate that GN outperforms BN and other normalization variants in various computer vision tasks [1]. Specifically, on ResNet-50 trained on ImageNet, GN has 10.6% lower error than BN when using a batch size of 2 [1]. When using typical batch sizes, GN is comparably good with BN and outperforms other normalization variants [1]. Moreover, GN can be naturally transferred from pre-training to fine-tuning [1]. In comparison to Layer Normalization (LN), GN is more suitable for convolutional neural networks (CNNs) because it exploits the channel dimension of feature maps. In contrast, LN normalizes the features across the spatial dimensions of each sample, which is more suitable for recurrent neural networks (RNNs) [1].

Instance Normalization (IN) is another normalization technique that normalizes the features across the spatial dimensions of each channel. IN is commonly used in style transfer and generative models, where the spatial information is important. However, IN is not suitable for classification tasks, where the spatial information is less important [1].

3. Methodology (of the Students' Project)

In our reproduction, we focused only on classification problems. For classification, the original paper used Imagenet data and Resnet50 architecture. However, since training on Imagenet data was infeasible given resource constraints, we focused on a dataset called Tiny Imagenet [4]. We added the data augmentation technique of mirror as described in the paper [] except for changes in aspect ratio since we are working with images of size less than the images in ImageNet dataset.

We implemented the resnet50 architecture and Group normalization technique as described in the paper. Our initial analysis revealed that the model was underfitting despite training on 100 epochs because our dataset was too small and the model was too complex. So for the remainder of our efforts, we decided to simplify the model and focused on training it to highlight the differences between Batch Normalization and the proposed Group Normalization technique. We use the same initializer (Xavier Initialisation) for convolution layers as used in the original paper and the learning rate schedule where we reduce the learning by a factor of 10 after 30, 60, and 90 epochs. In addition, we use 1 to initialize all γ parameters, except for each residual block's last normalization layer where we initialize γ by 0 following [6]. We experiment with Batch, Layer, Instance, and Group Normalizations.

In the last part, we report the results at different batch sizes for BN and GN trained on 10 epochs each due to resource constraints. All error rates are reported as the median error of the last five epochs as it is done in the original paper to decrease random variation.

3.1. Objectives and Technical Challenges

The objective was to replicate the results of the original paper [1] by producing code that upon successful completion would report similar numbers as reported in the paper. Our main challenge was the size of the original Imagenet dataset that was used to train the Resnet50, then we had to create a separate model for object detection and use the COCO dataset. We had to train models on separate normalization layers, with different batch sizes to report results. All this was infeasible given our time and resource constraints.

3.2.Problem Formulation and Design Description

Details:

1. Focus on classification tasks.
2. Gather the right dataset that would be similar to the Imagenet dataset but smaller in size.
3. Train the model on the new dataset. Since Original model was underfitting, design a smaller model that would provide reasonable results to verify the claims in paper [1].
4. Implement Batch, Layer, Instance and Group normalizations.
5. Train model on all the normalization and with different batch sizes.

4. Implementation

In the implementation, we found the dataset that is suitable for our reproduction. Then design each model and realize the implementation of the key layers. Finally complete the whole structure by adding layers to training models, changing training parameters, and plotting the desired graphs.

4.1 Data

The dataset that was used by the original paper was ImageNet. ImageNet is a large visual database designed for use in visual object recognition software research [2]. It contains more than 14 million images that have been hand-annotated to indicate what objects are pictured, and in at least one million of the images, bounding boxes are also provided. The database contains more than 20,000 categories, with a typical category consisting of several hundred images. The annotations of third-party image URLs are freely available directly from ImageNet, though the actual images are not owned by ImageNet.

However, due to the time consuming and the limitation of the computing power by our hand, we decided to use the Tiny ImageNet-200 dataset [4]. Tiny ImageNet-200 is

a dataset that was introduced by Le et al [5]. It is a 64×64 color-downsized version of ImageNet with 200 classes and 100,000 images in total. Each class has 500 training images, 50 validation images, and 50 test images. The dataset is used for various computer vision tasks such as classification, image clustering, and image generation [4].

Below we show some sample images from this dataset.

Fig 1. Random images from Tiny Imagenet dataset

4.2 Deep Learning Network

We use the same network architecture for training for our models. The only layer and parameter that is changed among the models for the different norm methods is the normalization layer. used for training were identical beside the different methods of normalization. The first layers of the structure consist of a 2D Convolutional layer with 2x2 strides and 'valid' padding and a GlorotNormal kernel initializer followed by Normalization of a particular norm method and an Activation layer. Output shape kept at 29 x 29 x 64 in these layers. There is then a Max pooling layer with 2x2 strides and padding='same' shrinking the output shape to 15 x 15 x 64. This is then followed by another 2D Convolutional layer with 1x1 strides and a GlorotNormal kernel initializer, a Normalization layer of the particular norm method and an Activation layer that expands the output to 15 x 15 x 256. Data then passes through a Global average pooling layer that shrinks the output to 1 x 1 x 256 and a Dense layer with kernel_initializer='he_normal' that further shrinks the output to 1 x 1 x 128. The data then enters the output Softmax layer in the shape of 1 x 1 x 200, corresponding to the 200 classes in the image dataset.

Layer (type)	**Output Shape**
conv2d (Conv2D)	(None, 29, 29, 64)
batch_normalization/ group_normalization/ instance_normalization/ layer_normalization (Normalization Layer)	
activation (Activation)	(None, 29, 29, 64)
max_pooling2d (MaxPooling2D)	(None, 15, 15, 64)
conv2d_1 (Conv2D)	(None, 15, 15, 256)
batch_normalization_1/ group_normalization_1/ instance_normalization_1/ layer_normalization_1 (Normalization Layer)	(None, 15, 15, 256)
activation_1 (Activation)	(None, 15, 15, 256)
avg_pool (GlobalAveragePooling2D)	(None, 256)
dense (Dense)	(None, 128)
output(Softmax)	(None, 200)

Table 1. List of layers of the model

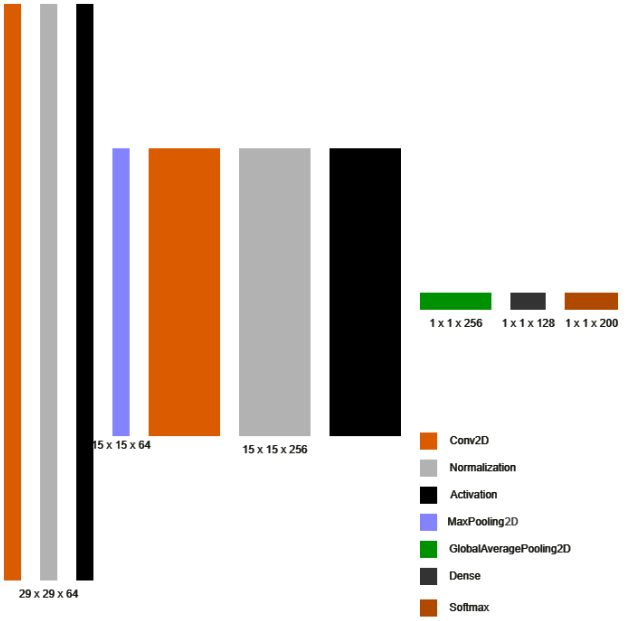

Fig 2. Architectural block diagram of the model

The 100000 training images with 500 images in each 200 classes are used as the training set for the models. The 10000 testing images with 50 images in each 200 classes are used as the validation set for the models. Data input into the model is kept at a batch size of 32 for the models used for comparison of error curves when using different

norm methods, which is the same as used in the paper. Batch sizes of 2, 4, 8, 16, and 32 are used to compare the sensitivity of the model to different batch sizes between batch normalization and group normalization. The training and validation data are passed through ImageDataGenerator to flip the images horizontally.

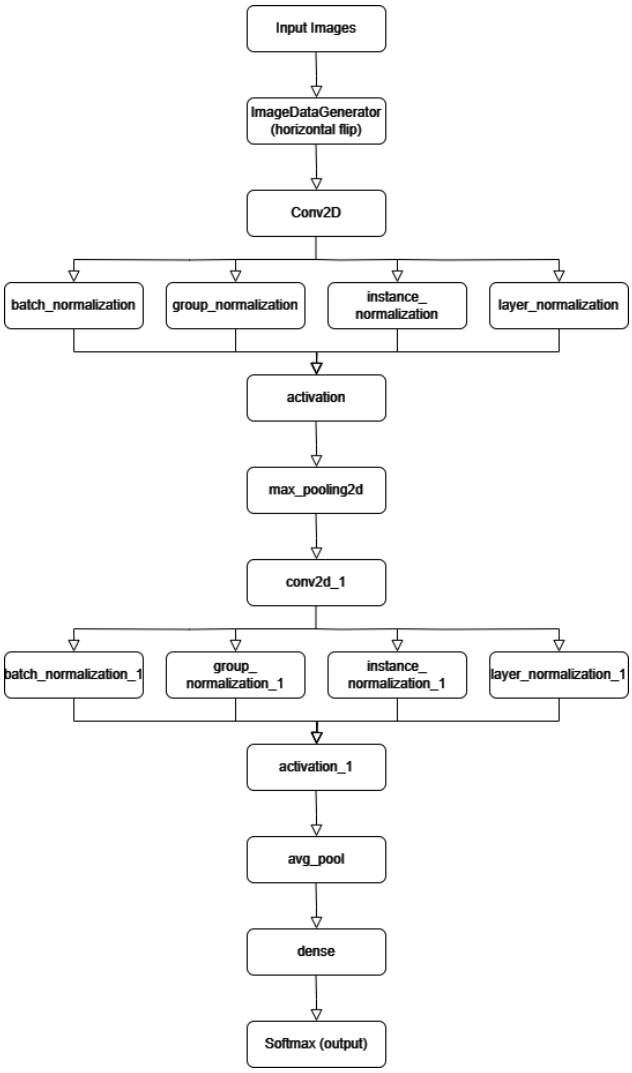

Fig 3. Flowchart of the model

4.3 Software Design

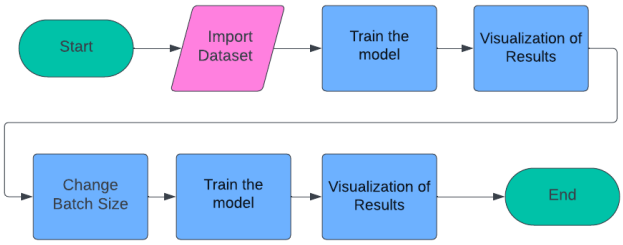

Fig 4. Top level flowchart of the software design

Our implementation followed the procedure shown in Fig 4. In the "Train the model" section, we used the architecture we created, and the details were discussed in the previous section and are shown in Fig 3.

When implementing the Batch Normalization layers and Layer Normalization layers, we used the existing models by importing them from tensorflow.keras.layers.

To realize Group Normalization layers, we adapted the code from Github [7] and changed the functions and initial parameters to fit our implementation [9]. For the realization of Instance Normalization layers, we did a similar procedure by changing the Github code [8] to fit our dataset and model [9].

5. Results

5.1 Project Results

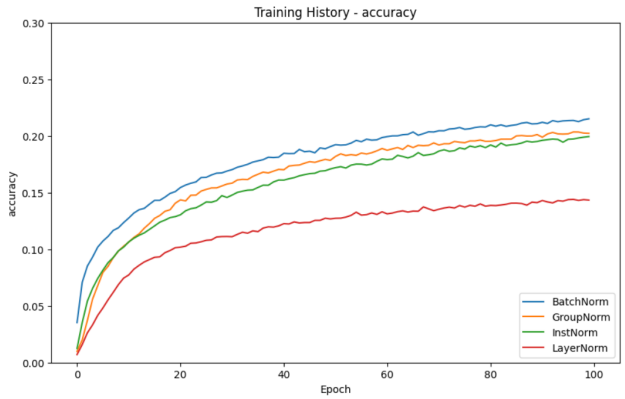

Fig 5. Training accuracy of different norm methods over 100 training epochs with a batch size of 32

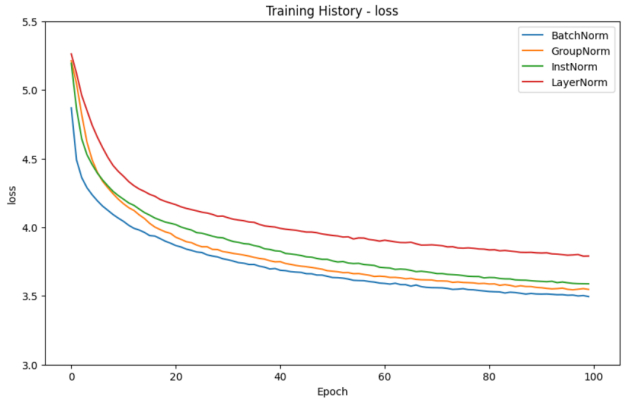

Fig 6. Training error of different norm methods over 100 training epochs with a batch size of 32

Fig 7. Validation of different norm methods over 100 training epochs with a batch size of 32

Fig 8. Validation error of different norm methods over 100 training epochs with a batch size of 32

	Batch Norm	**Group Norm**	**Inst Norm**	**Layer Norm**
val error	3.393	3.238	3.330	3.503
Δ (vs. BN)	-	-0.155	-0.063	0.11

Table 2. Average validation error from last 5 epochs of each normalization method

Training the model with an epoch of 100 with different normalization methods gives different trends of accuracy and error in both training and validation set.

Batch normalization resulted in the highest training accuracy and the lowest training error throughout the training process, but its resultant validation accuracy and error are not as good as in the training set. The validation accuracy and error fluctuate a lot during the training process, reaching the worst of all four methods in some instances and never reaching the level that is as good as group normalization.

Group normalization, on the other hand, resulted in the second-lowest training accuracy and the second-highest training error among the four methods. However, its resultant validation accuracy and error kept being the highest among all during the training process.

Instance normalization resulted in training accuracy and training error that is similar and just a little worse than group normalization. However, its resultant validation accuracy and error are worse than that of group normalization, similar to the validation accuracy and error of batch normalization.

Layer Normalization gives the worst performance in both training and validation accuracy and error.

Overall, the group normalization methods deliver the best performance as it results in the best and most stable validation accuracy and error.

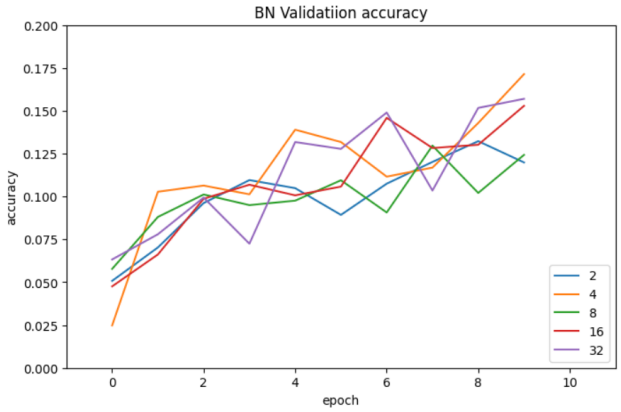

Fig 9. Validation accuracy of Batch Normalization over 10 training epochs with different batch sizes

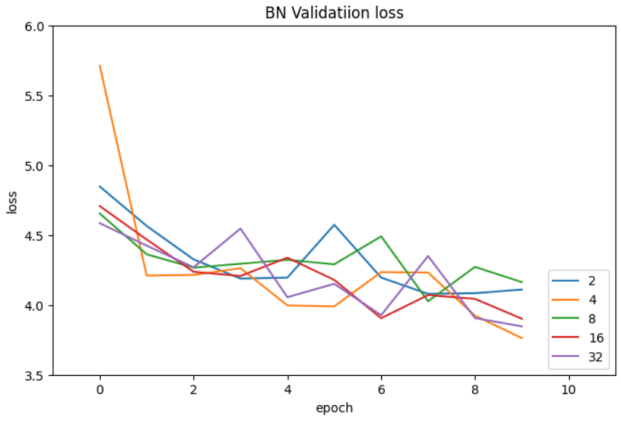

Fig 10. Validation error of Batch Normalization over 10 training epochs with different batch sizes

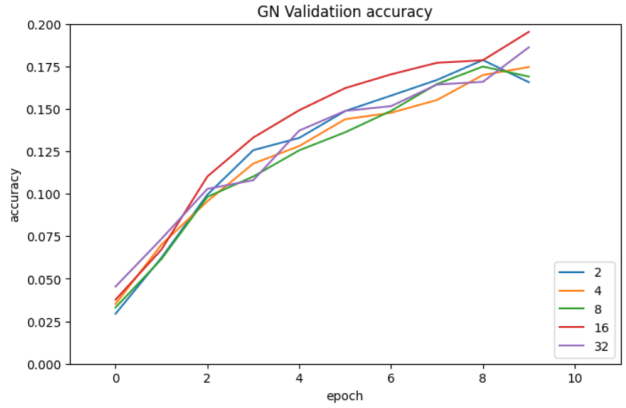

Fig 11. Validation accuracy of Group Normalization over 10 training epochs with different batch sizes

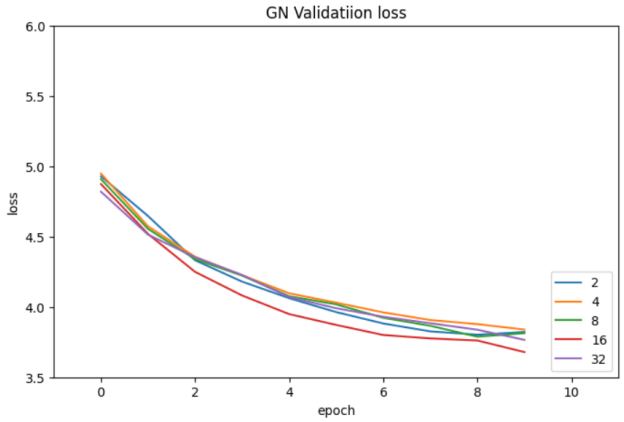

Fig 12. Validation accuracy of Group Normalization over 10 training epochs with different batch sizes

batch size	**32**	**16**	**8**	**4**	**2**
BN	4.126	4.024	4.062	4.147	4.239
GN	3.882	3.904	3.868	3.776	3.831
Δ	0.244	0.120	0.194	0.371	0.408

Table 3. Average validation error from last 5 epochs of each normalization methods

Batch Normalization resulted in curves that varied a lot when trained with different batch sizes, as shown in Fig 9 and 10. Group normalization, on the other hand, resulted in relatively consistent curves when trained with different batch sizes, as shown in Fig 11 and 12.

5.2 Comparison of the Results Between the Original Paper and Students' Project

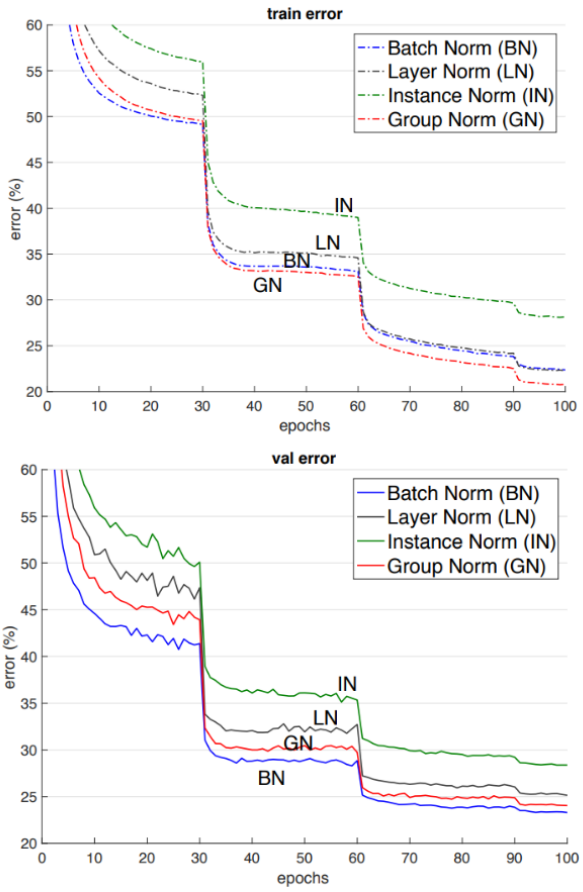

Fig 13. Comparison of error curves with a batch size of 32 (Figure 4.) from the paper

	BN	LN	IN	GN
val error (%)	**23.6**	25.3	28.4	24.1
△ (*vs.* BN)	-	*1.7*	*4.8*	***0.5***

Table 1. Comparison of error rates with a batch size of **32 images**/GPU, on ResNet-50 in the ImageNet validation set. The error curves are in Figure 4.

Fig 14. Comparison of error rates (Table 1) from the paper

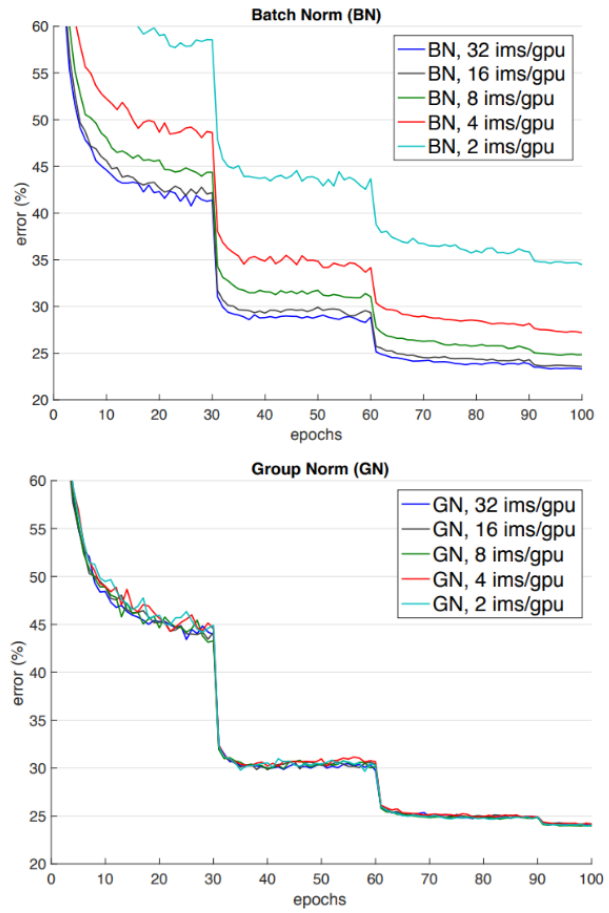

Fig 15. Comparison of error curves of Batch normalization and Group normalization with different batch sizes (Figure 5.) from the paper

batch size	32	16	8	4	2
BN	**23.6**	**23.7**	24.8	27.3	34.7
GN	24.1	24.2	**24.0**	**24.2**	**24.1**
△	*0.5*	*0.5*	*-0.8*	*-3.1*	*-10.6*

Table 2. Sensitivity to batch sizes. We show ResNet-50's validation error (%) in ImageNet. The last row shows the differences between BN and GN. The error curves are in Figure 5. This table is visualized in Figure 1.

Fig 16. Comparison of error rates of Batch normalization and Group normalization with different batch sizes (Table 2) from the paper

Our comparison of training and validation error of the four different normalization methods (Fig 6 and 8) resulted in similar relative positions of curves when compared to Figure 4 from the paper although they show very different trends, which should be the result of using different models and dataset. Group normalization is shown to have the best performance in our result, which corresponds to the result in the paper. However, Layer normalization is the one with the worst performance in our results but Instance normalization is the worst-performing one in the paper. This could also be caused by the difference in data and model architecture.

Our comparison of error of batch normalization and group normalization with different values of batch size gave similar results to the results plotted in Figure 5 in the paper. Batch normalization in the result of the paper showed variation in positions of curves with different batch sizes and group normalization showed relatively close positioning of curves. Our results on batch normalization (Fig. 10) showed similar variation in positions despite not matching in trend which should also be caused by the difference in dataset and model used. Our results on group normalization (Fig. 12) are also similarly closely positioned.

5.3 Discussion / Insights Gained

It is not uncommon for the results of a reproduction study to differ from the original paper due to differences in the dataset, model architecture, hyperparameters, or other factors. In our case, the results differed from the original paper in terms of the trends shown in the training and validation error curves for the four different normalization methods. However, we noted that the relative positions of the curves were similar to those in Fig 13 (Figure 4 of the original paper), and that Group Normalization had the best performance in our results, which corresponds to the result in the paper.

We also noticed that Layer Normalization had the worst performance in our results, while Instance Normalization was the worst-performing one in the paper. This could be due to differences in the dataset and model architecture used in our study compared to those in the original paper.

In addition, we noted that our results on batch normalization and group normalization with different values of batch size gave similar results to those plotted in Fig 15 (Figure 5 of the original paper). However, our results on batch normalization showed variation in the positions of curves with different batch sizes, while group normalization showed relatively close positioning of curves. This could also be due to differences in the dataset and model used in our study compared to those in the original paper.

Overall, it is important to carefully consider the differences between our study and the original paper when interpreting the results. It is also important to report any differences in the results and discuss the potential reasons for these differences. This can help to identify areas for further investigation and improve the reproducibility of research findings.

6. Future Work

Based on our implementation, there are several possible directions for future work. Investigating the impact of GN on different architectures: We used a specific model architecture in our implementation. It would be interesting to investigate the impact of GN on

other architectures and datasets.Exploring the effectiveness of GN in other domains: We used GN in image classification. It would be interesting to explore its effectiveness in other domains such as natural language processing, speech recognition, and reinforcement learning. Improving the efficiency of GN: While GN is more stable and accurate than BN in a wide range of batch sizes, it is still computationally expensive. It would be interesting to investigate ways to improve the efficiency of GN without sacrificing its accuracy. Combining GN with other normalization techniques: It would be interesting to investigate the effectiveness of combining GN with other normalization techniques such as weight normalization, layer normalization, and instance normalization.Investigating the impact of GN on transfer learning: We demonstrated that GN can be naturally transferred from pre-training to fine-tuning. It would be interesting to investigate the impact of GN on transfer learning in various computer vision tasks.

Overall, the effectiveness of GN has opened up new possibilities for research in the field of deep learning.

7. Conclusion

In conclusion, our project sought to replicate the findings of the paper [1], focusing on the comparison of Group Normalization (GN) with Batch Normalization (BN), Layer Normalization (LN), and Instance Normalization (IN) using the Tiny ImageNet-200 dataset and model architecture. While our results exhibited differences in trends compared to the original paper, such variations are not uncommon in reproduction studies and can be attributed to disparities in dataset characteristics, model architectures, and other experimental factors. Despite the observed differences in trends, it is noteworthy that the relative positions of the curves in our study closely resembled those presented in the original paper. Group Normalization consistently demonstrated superior performance, aligning with the outcomes reported by Wu and He. However, we observed a discrepancy in the performance of Layer Normalization and Instance Normalization compared to the original paper, indicating the influence of dataset and model architecture variations on the normalization methods' effectiveness.

Furthermore, our exploration of different batch sizes in Batch Normalization and Group Normalization revealed similarities in results to those depicted in the original paper. Notably, Group Normalization displayed more stable positioning of curves across varying batch sizes, underscoring its robustness in our experimental setup.

In interpreting these findings, it is crucial to acknowledge the inherent differences between our study and the original paper. These distinctions encompass dataset characteristics, model architectures, and potentially other experimental parameters. Our study reinforces the importance of transparently reporting such disparities, facilitating a comprehensive understanding of the reproducibility of research findings. In essence, while our results may diverge in specific trends from the original paper, the consistent relative performance of Group Normalization suggests its robustness across different experimental setups.

8. Acknowledgement

We would like to extend our gratitude to the entire team of the ECBM4040 course for researching and providing this incredible paper [1] to work on and to Google Cloud Platform for providing free credits to us for academic research.

9. References

[1] Y. Wu and K. He. Group normalization. *arXiv:1803.08494*, 2018.

[2] “ImageNet.” Wikipedia. https://en.wikipedia.org/wiki/ImageNet (accessed Dec. 17, 2023).

[3] “Image-net.” ImageNet. https://image-net.org/ (accessed Dec. 17, 2023).

[4] Wu, J., Zhang, Q., & Xu, G Tiny ImageNet Challenge.

[5] “Tiny ImageNet.” Papers With Code. https://paperswithcode.com/dataset/tiny-imagenet (accessed Dec. 17, 2023).

[6] P. Goyal, P. Dolla´r, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch SGD: Training ImageNet in 1 hour. arXiv:1706.02677, 2017.

[7] D. David. (2019). face_recognition_TF2. GitHub. https://github.com/dmonterom/face_recognition_TF2/blob/master/resnet_groupNorm.py

[8] keras-team. (2019). keras-contrib. Github. https://github.com/keras-team/keras-contrib/blob/master/keras_contrib/layers/normalization/instancenormalization.py

[9] H. S. Mashiana, Y. Luo, Z. Li. (2023). e4040-2023fall-Project-NNDL-hm3008-yl5444-zl3372. Github. https://github.com/ecbme4040/e4040-2023fall-Project-NNDL-hm3008-yl5444-zl3372/tree/main/utils

10. Appendix

10.1 Individual Student Contributions in Fractions

	hm3008	yl5444	zl3372
Last Name	Mashiana	Luo	Li

Fraction of (useful) total contribution	1/3	1/3	1/3
What I did 1	Conduct initial experiments with the original model and dataset.	Implemented and tested Layer normalization	Constructed the models based on the dataset and the original ResNet
What I did 2	Gather a smaller dataset similar to the original dataset mentioned in the paper.	Primarily wrote the report.	Adjusted and trained the models, generated plots
What I did 3	Implement training details, like initializers, schedules, data augmentation as mentioned in the paper and work on the report.	Organized repository, other requirements to be met for final submission.	Create architectu-ral block diagram and flowchart in part 4.2 and wrote part 4.2, 5.1 and 5.2 of the report