# Understanding Layers of Batch Normalization through Convolutional Neural Networks

Haohua Tang and Julian Fong

**Abstract**

Convolutional Neural Networks have continually been the foundation for producing well trained deep learning models with the goal of analyzing images of all shapes and forms. There is no doubt that many tools were implemented to fine-tune and optimize the learning process so that the model could achieve the best possible results. In this paper we re-implement similar CNNs employed in prior papers in order to train a CNN that can perform facial recognition. We will also incorporate tools such as Batch Normalization to showcase that normalizing the training process can further help us improve our model's performance. Next, we will be performing analysis on models that contain Batch Normalization and compare them with models that do not have them. We will also do comparisons between models that have varying sets of batch normalization layers to see the effect on the model's performance and perform sensitivity analysis on hyperparameters.

## 1 Introduction and Related Works

Since the inception of Convolutional Neural Networks, various academic papers have been written describing ways CNNs can be constructed to obtain high level performance on image datasets. These include Alexnet created by Krizhevsky et al [2], achieving a error rate of 18.9 on the ImageNet dataset in 2012, or more recently Google's GoogLeNet [3]. Many different CNNs have been created to perform all sorts of image classification tasks. A more difficult task requires a more complex neural network and proper fine tuning, otherwise the model could run into potential problems during training. Batch Normalization is a key tool that can be used to fine-tune the model in order to improve performance. It is applicable in almost every type of hidden layer as done in Ioffe et al 2015 [1], and it functions by normalizing parts of the training process to ensure the weight parameters are not too volatile. Early research on Batch Normalization allows for a higher learning rate as described by Bjorck et al [5], and Ioeffe et al [1]. Without Batch Normalization, the model can encounter a problem known to us as 'internal covariate shift', which occurs when the distribution of a certain layer's input parameters change during the training process. The fluctuation of these input parameters can slow down the training time for our weights and can

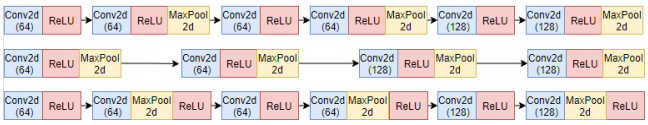


Figure 1: Algorithm Box for Model Architectures

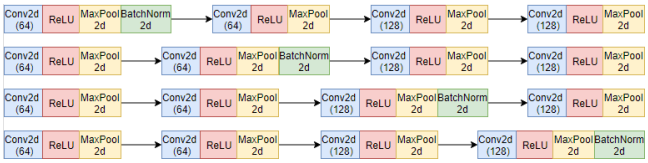


Figure 2: Model with BN Layers

only get worse when dealing with higher complex nonlinear functions. Batch Normalization continually encourages the activations and parameter weights to have zero mean and a standard deviation of one and by applying it, we significantly reduce the volatility that deeper layers in our model encounter. Otherwise, one slight change in the inputs in the shallower layers could have a drastic impact on the parameters in the deeper layers. For example, Bjorck et al [5] demonstrated that using Batch Normalization reduces the chance of explosions that gradient updates could take on.

## 2 Training Details

We begin by constructing three different CNN model architectures that can be visualized through Figure 1. The difference between the models can be described by the inclusion and exclusion of activations, layers, and adjustments of the layer's order in the model. We constructed these three models in order to perform hyperparameter analysis. After choosing the best model, we then incorporate Batch Normalization layers to see their impact on the model performance. The architecture can be then shown on Figure 2.

The three models were trained on a simple team-generated Face dataset. The model utilizes a categorical cross entropy loss function and Stochastic Gradient Descent with a learning rate of 0.01, batch size of 32, decay of 1e-6, and momentum of 0.9. After training the models, we performed hyperparameter sensitivity analysis, and tested which Batch Normalization layers had the most effect. In order to validate our initial results, we run model again on a new dataset to see if the results were reproducible.

The first CNN that we implemented contained 6 Convolutional Layers with input channels of 64 and kernel sizes of 3. Then we applied ReLU and Max Pooling where needed. The second model removes layering of convolutions and activations, to test if the model could perform just as well on the simple dataset. The third model is similar to the first model but instead, we switch the Max

Table 1: 3 Model Comparison

| Model | Accuracy | Stability | Learning Rate | With BN |
|---|---|---|---|---|
| Model 1 | 99.67% | Yes | 0.01 | No |
| Model 1 | 33.94% | No | 0.02 | No |
| Model 1 | 99.50% | Yes | 0.02 | Yes |
| Model 1 | 98.50% | No | 0.04 | Yes |
| Model 2 | 99.33% | Yes | 0.01 | No |
| Model 2 | 99.67% | Yes | 0.02 | Yes |
| Model 2 | 99.33% | Yes | 0.02 | Yes |
| Model 2 | 99.83% | Yes | 0.04 | Yes |
| Model 2 | 99.67% | No | 0.04 | No |
| Model 3 | 33.3% | No | 0.01 | No |

Pooling and ReLU activations in order to test the significance of their ordering. Finally, all three datasets, include a linear layer and an output layer with a softmax activation to produce probabilities for our desired classes.

# 3 Experiments

The two datasets used to test our model are both image datasets. The first dataset consists of 3,000 64 by 64 images with three classes, one for each individual's face. Each image was reduced to have dimension 64x64 and was properly normalized and given their own class. We split off the dataset into three parts, the training set, a validation set, and the test set. The training set consists of 1800 images, and 25% of that is used for the validation set. The remaining 1200 images are used for the test set. We run 6 accuracy tests for each Batch Normalization layers and average the results. The second dataset is the CIFAR-10 dataset, which includes 60,000 32x32 images and 10 classes. Afterwards, we picked a best model to include Batch Normalization onto every Convolutional layer, and ran tests to see which Batch Normalization layer had the most impact on model performance. We claim as a conjecture that applying Batch Normalization on the first layer will give the best results.

# 4 Results and Discussions

We apply the first model onto the Face dataset and see how the model would perform. We reach around a 99.67% final accuracy, which convergence around the ninth epoch. Afterwards, the accuracy remains stable between high 98% and mid 99%. Next, we change the architecture to match the right model as shown in Figure 1. When running tests, we noted that during training, the model would become unstable, while only occasionally obtaining high performance. In the remaining tests, the model would become stuck and finish training with

Table 2: Batch Normalization Performance

| Name | Accuracy on Face Dataset | Accuracy on CIFAR-10 |
|---|---|---|
| Batch Layer 1 | 99.83% | 79.98% |
| Batch Layer 2 | 99.53% | 78.5% |
| Batch Layer 3 | 97.79% | 79.1% |
| Batch Layer 4 | 87.40% | 79.31% |

extremely low accuracy. Afterwards, the model was then changed and the middle layer was removed to match the model in the center. The model obtains a final accuracy of 99.33%, and it converges at around epoch 14.

We opt to choose the center model so there is an easier interpretation when comparing models with different Batch Normalization layers. Since there are four convolutional layers, we have four potential layers to include Batch Normalization on, as shown in Figure 2. We now use both the Face dataset and the CIFAR-10 dataset to do testing.

On the Face Dataset as shown in Table 2, we have better accuracy performance when the Batch Normalization layer is applied onto either the first layer. It is possible that the early normalization of the weight parameters lead to better accuracy during the other parts of the training process. When applying the layers into the Convolutional layers in the middle, we get a slightly reduced accuracy of around 0.4% to 1.1%. The accuracy dropped heavily when applying Batch Normalization at layer 4, as the training process became slightly unstable. We also note that applying a Batch Normalization layer allows the model to converge faster and allows for a higher learning, achieving around 99% accuracy by around epoch 7 with a learning rate of 0.04.

# 5 Conclusion

In this paper we have described a method to test various effects of Batch Normalization layers as well as some different model architectures to compare model performance. We admit as a weakness that our initial three model architectures we designed performed too well on the Face dataset (the dataset may have been too simple). The simplicity of the face dataset made it harder to recognize differences between the implementation of the Batch Normalization layers since it would perform well regardless of their inclusion. Our model also performs slightly poorly on the CIFAR-10 dataset, only obtaining around 78-80% accuracy during our test runs. In this work, the instability of the model also made it more difficult to deduce performance of the Batch Normalization layers. As a strength, we demonstrated using four sets of Batch Normalization layers that applying one on the first could lead to better results than applying them during the middle convolutional layers or the last. We see a slight but substantial difference between the accuracies on the datasets. Also, from model 1 and 2 we have showed that batch normalization allows a larger learning rate.

# References

All of the relevant code can be found here:

`https://github.com/Kyurem1001/CSC413-Project`

[1] Ioeffe, Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv preprint arXiv:1502.03167* 2015.

[2] Krizhevsky, Sutskever, Hinton, ImageNet Classification with Deep Convolutional Neural Networks. 2012.

[3] Szegedy, Liu, Jia, Sermanet, Reed, Anguelov, Erhan, Vanhoucke, Rabinovich. Going deeper with convolutions *arXiv, preprint arXiv: 1409.4842*, 2015.

[4] Luo, Li, Urtasun, Zemel, Understanding the Effective Receptive Field in Deep Convolutional Neural Networks. *arXiv, preprint arXiv: 1701.04128*, 2017.

[5] Bjorck, Gomes, Selman, Weinberger, Understanding Batch Normalization, 2018

# Contributions

Julian Fong: Majority of Report Writing, part of code implementation, part of testing, part of design and model architecture, part of research.

Haohua Tang: Part of Report Writing, majority of code implementation, part of testing, part of design and model architecture, part of research.

## Appendix

Two codebases were used: One for the generation of the Face dataset, and another to write evaluate and train functions. We also had an original model architecture which we then altered nearly completely to match our goals and needs for the project. We also slightly changed the load data function to meet dataset needs.

"data.tar.gz" is the Face dataset for convenience.