

School of Informatics



Informatics Research Review


Methods of solving internal covariate shift and improving performance of deep convolutional neural networks for image classification



Abstract

Deep Convolutional Networks have provided astounding results in image classification tasks over the past decade. As the depth of these networks increase, optimization issues such as internal covariate shift become more apparent and lead to networks breaking down in training. Several optimization methods have been introduced to solve this issue and improve performance. These methods have allowed for the training of much deeper Convolutional Networks than previously possible and improved image classification performance. This paper reviews and compares these methods in terms of their ability to solve the problem of internal covariate shift and increase network performance in image classification.

Date: Friday 29th January, 2021

Supervisor: 

1 Introduction

Convolutional Neural Networks (CNNs) have been used with great success in a variety of different fields and applications. Perhaps the most notable of these is image classification, where deep CNNs have led to huge improvements in image recognition [1] [2]. However, optimization issues present a significant challenge when training such architectures. This research review will examine one of these problems: Internal covariate shift. Internal covariate shift and the issue of vanishing/exploding gradients which results from it lead to training error rates in deeper CNNs exceeding that of simpler, shallower CNNs. This is surprising as deeper, more complex networks generate more complex functions than their shallow counterparts with which to learn the training data [3]. Further, deep networks can offer more computational and statistical efficiency as argued by Bengio et Al. [4]. We would expect this to lead to lower training error.

CNNs differ from traditional Artificial Neural Network (ANN) architectures in their introduction of convolutional and pooling layers to the network. A CNN is a network which combines these two types of network layers with the traditional fully-connected layers found in ANNs [5]. Each unit in a convolutional layer is known as a filter and receives inputs from multiple units in the previous layer. These inputs are situated in the same local neighbourhood as each other in the previous layer [6]. This introduces the concept of "local receptive fields" [6] to the network and allows the network to identify shared information between the units in the local receptive field, thereby allowing the convolutional layer to extract features from the data. In terms of image classification a grouping of pixels may indicate the edge of an object and the convolutional units can extract this information from local receptive fields. One may think of each filter extracting "feature maps" from the inputted data, with a feature map outputted by every filter. Each convolutional layer is followed by a pooling or "sub-sampling" layer which performs a local averaging and sub-sampling of the outputs of the convolutional layer. The combination of convolutional, pooling and fully connected layers in a traditional CNN for image classification is shown in Figure 1.

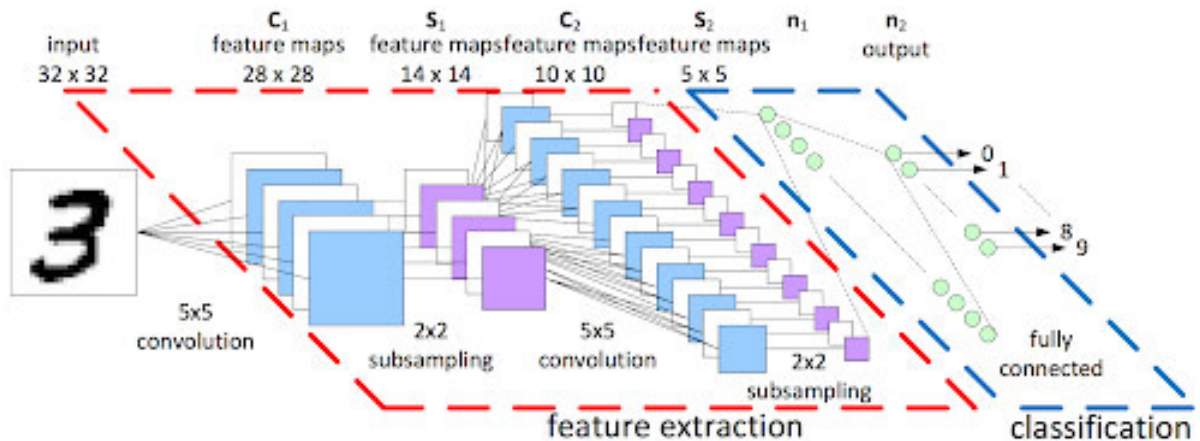


Figure 1: Example architecture of a CNN for hand written digit classification. From [7]

Similar to traditional ANNs, CNNs self-optimize through learning. This learning is achieved through gradient-based learning methods, which is where our optimization problem arises. Internal covariate shift refers to changes in the distribution of network activations due to the

change in network parameters during training [8]. Each time the inputs of a network pass through a network layer they are perturbed by the layer’s activation functions. These perturbations occur at every hidden layer of the network, meaning any change in the input parameters can have a large effect on the distribution of network activations throughout the model [8]. Non-linear activation functions in network layers can lead to layer outputs becoming saturated, when the outputs of a unit asymptotically approach the bounds of the unit’s activation function [9]. Internal covariate shift makes it difficult to prevent network activations from becoming saturated in deep networks due to the amplification of the effect of parameter changes as inputs pass through the model. When activations become saturated, there is little to no change in the value of layer outputs as we go through the network. Saturation damages the network’s ability to learn by reducing the network to a binary state, where only values at the asymptotic bounds of the activation function are outputted [9]. In this state gradient-based learning methods struggle to optimize the weights of the network since adjustments to the weights will have little effect on the saturated units. This leads to stagnation in learning. In the worst case this results in vanishing/exploding gradients. In this case, the error gradient of the network becomes so prohibitively small/large (vanishing/exploding) that the model is prevented from learning the training data. This effectively stops the model from being able to train as set out by Hochreiter in [10].

Traditional methods of avoiding these issues involve careful initialization of model parameters [4], usage of very small learning rates and the usage of Rectified Linear Units as activation functions rather than logistic sigmoid activation functions [11]. However, the small learning rates used lead to slow training and the resulting models do not overcome internal covariate shift as networks deepen. Even with these precautions, prior to the introduction of modern methods such as Batch Normalization [8], the deepest state-of-the-art network architectures were based on the GoogleNet model of 22 layer depth [12]. Modern approaches both solve the optimization issues *and* improve performance, allowing for the faster training of deeper networks.

We examine a number of different approaches to solving internal covariate shift and avoiding the resulting vanishing/exploding gradients problem. These solutions can be divided into two broad categories: Layer-based solutions and connection-based solutions. Layer-based solutions refer to methods that solve our optimization problem through the addition of new layers to the network. Similarly, connection-based methods refer to methods which exploit novel connections between layers in order to optimize training. Importantly, layer-based and connection-based approaches can be combined to achieve optimal results as we will see. Aside from their ability to alleviate internal covariate shift these optimization methods are compared in terms of the performance increase in models. Performance is measured by classification accuracy on benchmark image classification tasks.

Each of the proposals examined are successful in alleviating internal covariate shift in the training of deep CNNs to varying degrees. While all allow for deeper networks to be trained successfully, some break down as network depth continues to increase. All approaches therefore allow for the successful training of deeper networks, which in turn results in improved performance in image classification benchmark tasks compared to traditional CNN architectures. Furthermore, all of the solutions examined allow for faster training speeds. However, it is approaches which combine both layer based and connection based methods that are most successful in mitigating the effects of internal covariate shift and improving performance.

2 Literature Review

2.1 Layer-based Methods

Ioffe and Szegedy seek to confront internal covariate shift through the introduction of a novel technique known as "Batch Normalization" (BN) [8]. Not only do they seek to solve this optimization issue by increasing network stability but also to accelerate model training. The BN technique reduces internal covariate shift by stabilizing the distribution of nonlinearity inputs through normalizing the mean and variance of layer inputs. It has the further benefit of reducing the dependency of gradients on the scale of model parameters and their initial values. This benefits gradient flow, allowing higher learning rates to be used without risking vanishing/exploding gradients.

The aim is to ensure that the network always produces activations with the desired distribution, regardless of the parameter values while also preserving the network's information. To this end, scalar input features are independently normalized to have distribution $N(0, 1)$. Layers with d -dimensional inputs $(x_1 \dots x_d)$ are normalized for each dimension by:

$$\hat{x}_k = \frac{x_k - \mathbf{E}[x_k]}{\sqrt{x_k}} \quad (1)$$

This normalization could change what a layer represents (e.g. constraining the inputs of a sigmoid activation function to a linear regime of the nonlinearity). This is avoided by introducing learnable parameters γ_k, β_k for each x_k such that:

$$y_k = \gamma_k \hat{x}_k + \beta_k \quad (2)$$

These parameters are learned along with the original model parameters. The transformation from x_k to y_k is known as the BN transformation and is performed over mini-batches of inputs. For experimentation on CNNs BN was applied to the input of every nonlinearity activation function in the network. Testing on multiple different network architectures found BN to allow for both increased learning rate (meaning higher training speed) and the removal of dropout layers. The removal of dropout layers, an effective regularization technique in neural networks points to the BN having the additional benefit of regularization. In practice, BN achieves its aims of not only alleviating internal covariate shift but also allowing faster network convergence and greater learning rates [12].

Despite its clear advantages, BN is not without its drawbacks. Since the estimation of the mean and standard deviation of inputs to hidden layers depends on batch statistics, these estimates can be inaccurate due to shifting parameter values, especially in the initial stages of training. Also, BN is not applicable to a batch size of one in which case it leads to overfitting of the training data and poor generalization performance [13]. These issues are addressed with the introduction of Normalization Propagation (NP) [14].

NP seeks to avoid the issue of inaccurate batch statistic estimates using the knowledge that the distribution of pre-activation data is Gaussian. This is an assumption which is generally true of real-world data and is also assumed by the BN creators [8]. Based on this assumption of a Gaussian distribution parametric normalization can be applied to the pre-activation data,

removing the need to use batch statistics to estimate its distribution. This normalization is then propagated through each layer of the network, alleviating internal covariate shift and preventing vanishing/exploding gradients. Experiments on the CIFAR-10 image classification benchmark dataset show that NP avoids internal covariate shift to a greater degree than BN. Figure 2 illustrates the evolution of the input distribution to hidden layers across two otherwise identical networks with NP or BN applied, along with a third non-normalized network. The experiment is carried out on the validation set of the CIFAR-10 data. The NP network is clearly more successful in normalizing the input mean to hidden layers to zero, with much less variability than the BN network. Figure 2 also demonstrates the success of both methods in reducing internal covariate shift relative to traditional network architectures with the huge difference in hidden layer input means across layers for the non-normalized network.

Further, NP can be applied to a batch size of just one, with performance appearing to increase as batch size is lowered [14]. The speed of training of NP networks is also shown to be approximately 12% faster than BN networks [14] with convergence being achieved with less training epochs, shown in Figure 3. Hence, NP manages to address the existing issues of BN and improve upon its successes in alleviating internal covariate shift and accelerating training.

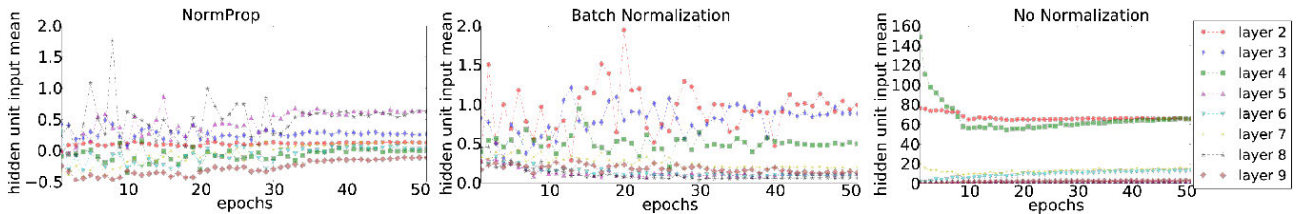


Figure 2: Evolution of hidden layer input distribution mean of a randomly chosen unit for all layers of 9 layer CNNs trained on CIFAR-10. From [14]

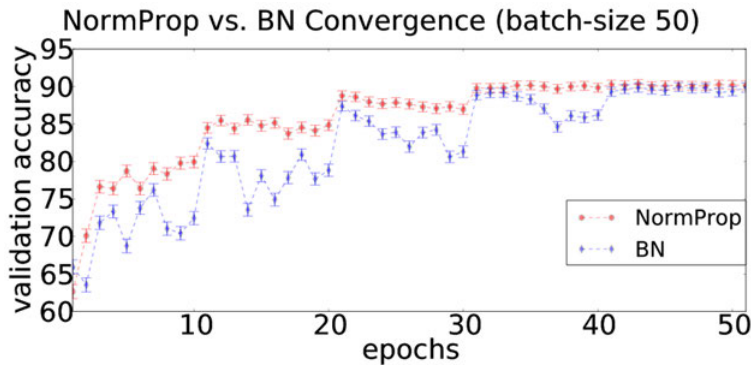


Figure 3: NP vs. BN convergence stability. NP alleviates internal covariate shift more than BN, especially in early training. From [14]

Another novel approach named deeply-supervised nets (DSN) is adopted in [15]. Rather than the standard supervision of only the output layer and then back propagating this supervision to earlier layers, DSNs provide integrated, direct supervision of all hidden layers in a network. Direct supervision is carried out by introducing “companion objective functions” for every hidden layer. These companion functions provide a classification output at each hidden layer, analogous to the output of a truncated network at that point. Empirical results show DSNs reduce

internal covariate shift leading to improved convergence behaviour while giving a reduction in test error but not necessarily training error (effectively acting as a kind of feature regularization).

This supervision acts as a strong method of regularization for classification accuracy and learned features for smaller training data and relatively shallow networks. In deeper networks on larger training data it improves performance by alleviating the problem of vanishing gradients. The DSN framework was tested against BN and NP methods on the CIFAR-10, CIFAR-100, and SVHN benchmark datasets. For the purposes of these experiments, CNNs with the same architectures and hyperparameters were used, with the only differences coming from the implementation of the methods in question [14]. These results are presented in Table 1, with NP achieving the best results for all datasets used.

Method	CIFAR-10	CIFAR-100	SVHN
Normalization Propagation	9.11	32.19	1.88
Batch Normalization	9.41	35.32	2.25
Deeply Supervised Nets	9.69	34.57	1.92

Table 1: Comparison of optimization methods on benchmark image classification datasets. Figures represent % classification error on each benchmark’s test set. From [14]

2.2 Connection-based Methods

Layer-based solutions are outperformed in terms of their ability to train deep CNNs without experiencing vanishing/exploding gradients in [16] through the implementation of a ”deep residual learning” or ”ResNet” framework. Rather than hoping that the stacked layers of a deep net assume a desired underlying mapping the authors fit these layers to a residual mapping. They consider $\mathcal{H}(\mathbf{x})$ being the underlying mapping of some stacked layers in a network with \mathbf{x} being the inputs to these layers. The key hypothesis is that if multiple nonlinear layers can approximate complex functions like $\mathcal{H}(\mathbf{x})$ then one may assume that they can asymptotically approximate the residual function $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$. It is reasoned that this residual function may be easier to learn. However the authors themselves admit that this hypothesis of asymptotic approximation is an open debate [3].

The residual learning is applied to blocks of stacked layers throughout the network. The output vector \mathbf{y} from a building block is defined as follows:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x} \quad (3)$$

where $\mathcal{F}(\mathbf{x}, \{W_i\})$ represents the residual mapping to be learned. A standard building block according to Equation 3 is shown in Figure 4. The addition of the input \mathbf{x} to the residual mapping is implemented by the insertion of shortcut connections and element-wise addition. Shortcut connections are connections which skip one or more layers in the block [17], [18]. Since these shortcut connections are applied to arbitrary blocks of layers, it is possible to apply deep residual learning to stacks of layers which include optimizing layers such as BN layers.

For testing the efficacy of ResNets, two ”plain” CNNs with 19 and 34 layers are applied to the ImageNet dataset as well as ResNets of the same depth. BN is applied in the same way as [8] to all models. To ensure no unfair advantage is afforded to ResNets it is ensured that

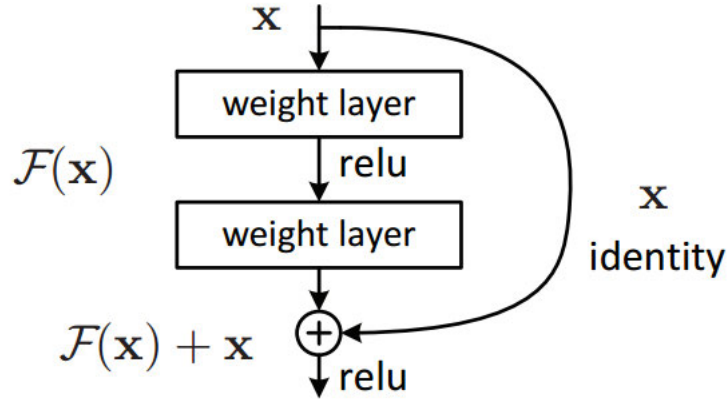


Figure 4: Building Block of a ResNet. From [16]

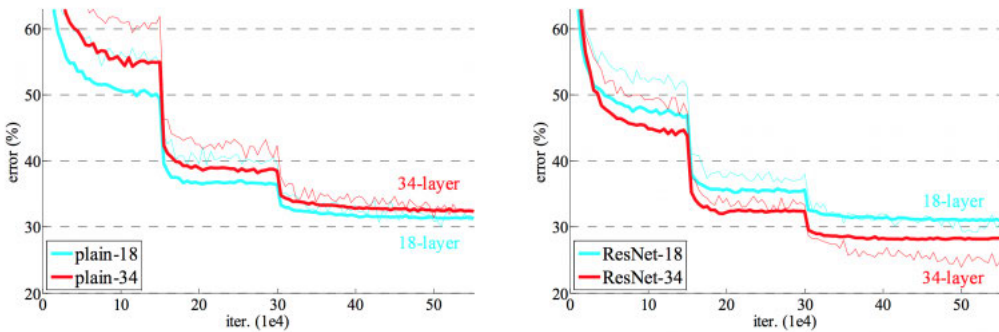


Figure 5: Comparison of training between plain networks and ResNets. From [16]

corresponding plain networks and ResNets used have the same number of parameters. Despite the inclusion of BN it is found that the deeper plain network has a higher validation error than the shallower network, indicating that BN does not fully solve internal covariate shift. However, this is not the case for the ResNet models, with both models outperforming their counterparts and the deeper ResNet network improving on the shallower Resnet in terms of both training and validation error. Results of these experiments are shown in Table 2. The consistent improvement in model performance indicates that residual learning addresses the optimization issues of internal covariate shift and vanishing gradients to an even greater degree than BN. In fact, ResNets allow for the training of extremely deep networks with a 110 layer ResNet achieving a test error of 6.43% on the CIFAR-10 test set, out-performing all previously examined methods [16]. In addition to the allowance of deeper networks, ResNets also exhibit faster convergence as illustrated in Figure 5.

It is worth noting that the ResNets do make use of BN in their architecture, signifying that the residual learning method adds to the already successful BN method, rather than superseding it. Further, the identity mapping of ResNets invalidates the theoretical premise of NP and as such, applying NP to ResNets results in a significant degradation in performance [12]. NP can be adjusted in order to improve the performance of ResNet models but nonetheless these models are still outperformed by ResNets using BN. This does lead to ResNets suffering from BN’s sensitivity to batch-size, with generalization performance decreasing as batch size decreases [13].

Depth	Plain	ResNet
18	27.94	27.88
34	28.54	25.03

Table 2: Top-1 error (%) on ImageNet validation data for plain and Resnet networks. From [16]

Huang et Al. succeed in solving the optimization problems by implementing a new network architecture, rather than simply modifying the existing structure of CNNs [19]. They propose a network architecture ensuring maximum information flow between layers by connecting all layers directly with each other. For a network with L layers, this adds $\frac{L(L+1)}{2}$ connections. This architecture is known as a ‘‘Dense Convolutional Network’’ (‘‘DenseNet’’). Similarly to ResNet, DenseNet architectures can be simply combined with layer-based methods since the only change in the network is the addition of these new layer connections. In a network with L layers, with each layer $l, 1 \leq l \leq L$ applying a non-linear transformation $H_l(\cdot)$, the densely connected architecture is achieved by passing as input to the l^{th} layer the feature-maps of all preceding layers x_0, \dots, x_{l-1} where x_0 is the original input image to the network. This results in an input $x_l = H_l([x_0, x_1, \dots, x_{l-1}])$ with $[x_0, x_1, \dots, x_{l-1}]$ being the concatenation of the feature maps in the layers prior to layer l . This concatenation is not viable when feature map sizes change. To allow for this the network is divided into multiple densely connected blocks, with convolution and pooling layers acting as transition layers between the blocks.

If each function H_l produces k feature maps then the l^{th} layer has $k_0 + k \times (l - 1)$ feature maps where k_0 is the number of feature maps in the input layer. The authors refer to k as the ‘‘growth rate’’ of the network, the number of feature maps that are added with each layer. State-of-the-art results can be achieved with relatively small growth rates since each layer has access to all preceding feature maps. This access to the collective knowledge of the network removes the need to replicate the network’s feature maps at every layer unlike traditional CNNs. This allows for quicker training since less computation is needed to replicate previous feature maps. Similarly to ResNets, Dense Nets also use BN, with the BN transform being applied before every activation layer in a dense block. An example of a dense block is given in Figure 6.

The fully connected nature of DenseNets allows for feature reuse, yielding highly parameter efficient and easy to train modules. Improved flow of information allows for improved gradient flow, limiting the effects of internal covariate shift, making them easier to train than traditional CNNs. Each layer has direct access to the gradients from the loss function and the original input signal, leading to an implicit deep supervision. This deep supervision allows training of deeper network architectures, as explained by the findings of [15]. This may explain the improved accuracy of DenseNets on benchmark tasks. Moreover, the dense connections have a regularizing effect, reducing overfitting. DenseNets are evaluated on three benchmark datasets (CIFAR-10, CIFAR-100, SVHN), outperforming ResNet models even when compared with ResNets of greater depth. Comparison of DenseNet performance against ResNets on the four datasets are shown in Table 3. Their parameter efficiency allows for quicker training than existing methods while giving comparable accuracy. Additionally, DenseNets yield consistent improvements in accuracy as parameters increase, without performance degradation or overfitting occurring. Overall, DenseNets appear to be the optimal solution the problem of internal covariate shift while also achieving state-of-the-art results with less computation than other methods.

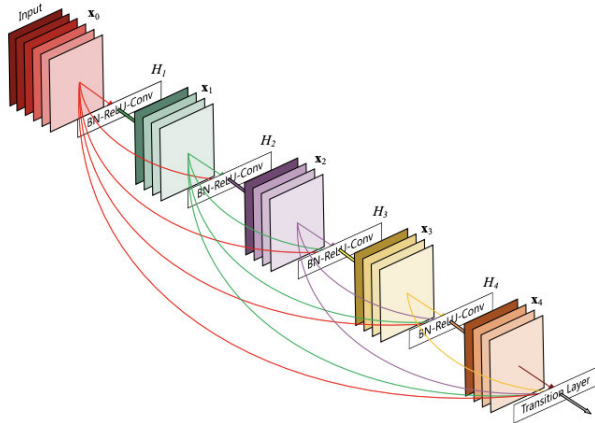


Figure 6: A 5-layer dense block. From [19].

Method	Depth	Parameters	C10	C10+	C100	C100+	SVHN
ResNet	110	1.7M	13.63	6.41	44.74	27.22	2.01
DenseNet (k = 12)	40	1.0M	7.00	5.24	27.55	24.42	1.79
DenseNet (k = 12)	100	7.0M	5.77	4.10	23.79	20.20	1.67
DenseNet (k = 24)	100	27.2M	5.83	3.74	23.42	19.25	1.59

Table 3: Comparison of ResNet model and DenseNet models of different architectures. Error rates (%) on CIFAR and SVHN datasets. k denotes growth rate of DenseNets. “+” indicates standard data augmentation (translation and/or mirroring). From [19]

2.3 Future Improvement

As previously mentioned both DenseNets and ResNets suffer from the issues inherent in BN. As recently as mid-2020 a new method challenging BN was introduced, Filter Response Normalization (FRN) [20]. FRN follows the conventional approach of CNNs with convolutional layers producing feature maps as normal. Each convolutional layer uses learned filters to extract feature maps from inputs and the convolution outputs a 4-dimensional tensor \mathbf{X} with dimensions $[B, W, H, C]$. Here B denotes batch size, W and H are the dimensions of the map space and C is the number of filters in the convolutional layer. FRN is achieved using the following formula:

$$\hat{\mathbf{x}} = \frac{\mathbf{x}}{\sqrt{v^2 + \epsilon}} \quad (4)$$

In this case \mathbf{x} is defined as the vector of filter responses for the b^{th} batch point on the c^{th} filter, with v^2 being the mean squared norm of \mathbf{x} and ϵ a low magnitude positive constant to prevent division by zero. Similarly, to BN this removes scaling. However, FRN does not perform mean subtraction like BN. This would lead to activations having a bias away from zero. This is rectified through the introduction of a Threshold Linear Unit (TLU), an adapted ReLU with activation function with a learned threshold τ and activation output z defined as:

$$z = \max(\mathbf{y}, \tau) \quad (5)$$

The authors hypothesize that TLU activation functions are superior to ReLu for optimization in this case. The FRN method is tested on the ImageNet dataset on two 50 layer ResNets, one with FRN and one with BN. The FRN model outperforms the BN model for all batch-sizes tested showing that ResNets can indeed be further improved with this novel technique. Further research into the combination of DenseNets and FRN layers would be worthwhile.

3 Summary & Conclusion

This review has covered multiple methods of solving the challenge of internal covariate shift when training deep CNNs. While all methods achieve this aim to a greater or lesser degree, there are clearly preferable methods for optimizing the training of CNNs. Layer-based methods manage to reduce internal covariate shift and allow for the training of deeper networks with NP proving to give the greatest performance in image classification tasks. However, BN remains the more important development due to its use in connection-based methods. The more recent development of the FRN method gives a superior alternative to BN for use in ResNet architectures.

We have seen that connection based methods offer the most successful means of optimization, allowing for the training of extremely deep networks and state-of-the-art performance. However, it is important to note that these methods lean on existing innovations with their inclusion of BN in their networks. Hence, BN may be construed as the most important of the breakthroughs discussed in this paper, due to its use in subsequent state-of-the-art network models with certain researchers describing it as a "cornerstone of current high performing deep neural network models" [20]. It remains to be seen whether FRN can have a similar impact on the field to BN.

DenseNets present the state-of-the-art model for the training of deep CNNs. Not only do they solve the issue of internal covariate shift but they offer unbeaten performance on competitive benchmark datasets at the time of publication. The introduction of FRN to the field provides an interesting avenue of further research into further and improving performance in DenseNets.

References

- [1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. Curran Associates, Inc., 2012.
- [2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. Curran Associates, Inc., 2012.
- [3] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27, pages 2924–2932. Curran Associates, Inc., 2014.
- [4] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterton, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. JMLR Workshop and Conference Proceedings.
- [5] K. O’Shea and R. Nash. An introduction to convolutional neural networks. *ArXiv*, abs/1511.08458, 2015.
- [6] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [7] A toy convolutional neural network for image classification with keras. <https://www.kernix.com/article/a-toy-convolutional-neural-network-for-image-classification-with-keras/>, Jul 2020. Accessed: 23/01/2021.
- [8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.
- [9] A. Rakitianskaia and A. Engelbrecht. Measuring saturation in neural networks. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 1423–1430, 2015.
- [10] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.
- [11] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML’10, page 807–814, Madison, WI, USA, 2010. Omnipress.
- [12] Wenling Shang, Justin Chiu, and Kihyuk Sohn. Exploring normalization in deep residual networks with concatenated rectified linear units. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), Feb. 2017.
- [13] Xiangru Lian and Ji Liu. Revisit batch normalization: New understanding from an optimization view and a refinement via composition optimization, 2018. Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS) 2019, Naha, Okinawa, Japan. PMLR: Volume 89.
- [14] Devansh Arpit, Yingbo Zhou, Bhargava Kota, and Venu Govindaraju. Normalization propagation: A parametric technique for removing internal covariate shift in deep networks. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1168–1176, New York, New York, USA, 20–22 Jun 2016. PMLR.

- [15] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets, 2014.
- [16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.
- [17] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., USA, 1995.
- [18] Brian D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.
- [19] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.
- [20] Saurabh Singh and Shankar Krishnan. Filter response normalization layer: Eliminating batch dependence in the training of deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.