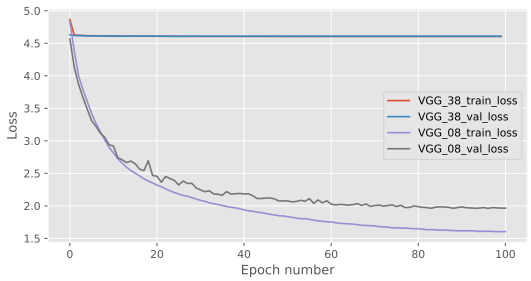
MLP Coursework 2

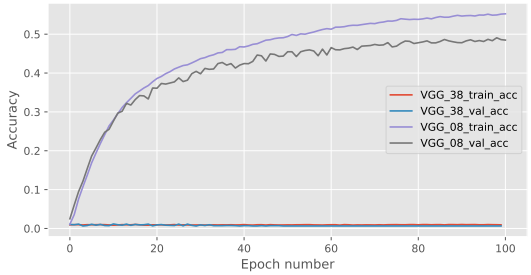
s1893731

Abstract

Deep neural networks have become the state-of-the-art in many standard computer vision problems thanks to more powerful neural networks and large labeled datasets. While very deep networks allow for better deciphering of the complex patterns in the data, training these models successfully is a challenging task due to problematic gradient flow through the layers, known as vanishing/exploding gradient problem (VGP and EGP respectively). In this report, we first analyze this problem in VGG models with 8 and 38 hidden layers on the CIFAR100 image dataset, by monitoring the gradient flow during training. We explore known solutions to this problem including batch normalization or residual connections, and explain their theory and implementation details. Our experiments show that batch normalization and residual connections effectively address the aforementioned problem and hence enable a deeper model to outperform shallower ones in the same experimental setup.



(a) Loss per epoch



(b) Accuracy per epoch

Figure 1. Training curves for VGG08 and VGG38

1. Introduction

Despite the remarkable progress of deep neural networks in image classification problems (Simonyan & Zisserman, 2014; He et al., 2016), training very deep networks is a challenging procedure. One of the major problems is the VGP, a phenomenon where gradients from the loss function shrink to zero as they backpropagate to earlier layers, hence preventing the network from updating its weights effectively. This phenomenon is prevalent and has been extensively studied in various deep network including feed-forward networks (Glorot & Bengio, 2010), RNNs (Bengio et al., 1993), and CNNs (He et al., 2016). Multiple solutions have been proposed to mitigate this problem by using weight initialization strategies (Glorot & Bengio, 2010), activation functions (Glorot & Bengio, 2010), input normalization (Bishop et al., 1995), batch normalization (Ioffe & Szegedy, 2015), and shortcut connections (He et al., 2016; Huang et al., 2017).

This report focuses on diagnosing the VGP occurred in the VGG38 model and addressing it by implementing two standard solutions. In particular, we first study the "broken" network in terms of its gradient flow, norm of gradients with respect to model weights for each layer and contrast it to ones in the healthy VGG08 to pinpoint the problem. Next, we review two standard solutions for this problem, batch normalization (BN) (Ioffe & Szegedy, 2015) and residual connections (RC) (He et al., 2016) in detail and discuss how they can address the gradient problem. We first incorporate batch normalization (denoted as VGG38+BN), residual connections (denoted as VGG38+RC), and their combination (denoted as VGG38+BN+RC) to the given VGG38 architecture. We train the resulting three configurations, and VGG08 and VGG38 models on CIFAR-100 dataset and present the results. The results show that though separate use of BN and RC does tackle the vanishing/exploding gradient problem, therefore enabling the training of the VGG38 model, the best results are obtained by combining both BN and RC.

2. Identifying training problems of a deep CNN

[ ] Concretely, training deep neural typically involves three steps, forward pass, backward pass (or backpropagation algorithm (Rumelhart et al., 1986)) and weight update. The first step involves passing the input x^0 to the network and producing the network prediction and also the error value. In detail, each layer takes in the output of the previous layer

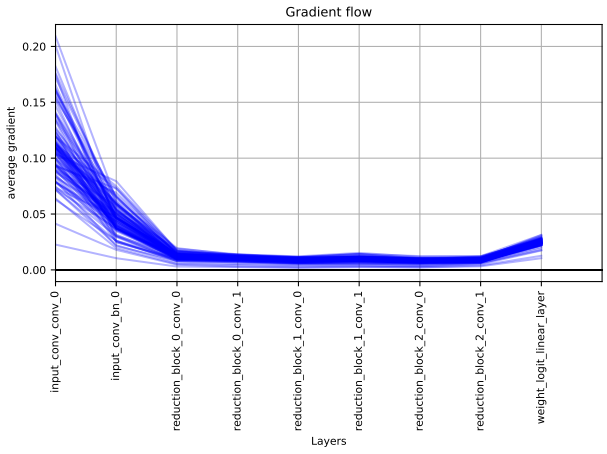


Figure 2. Gradient flow on VGG08

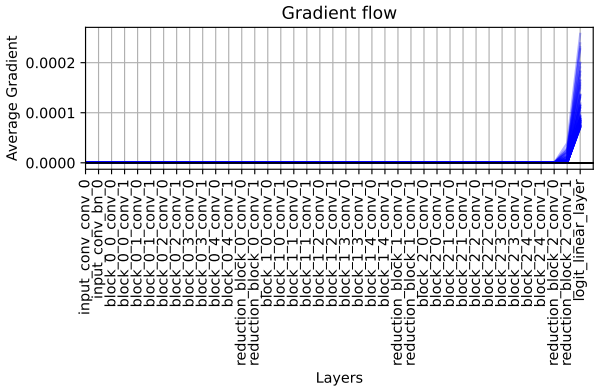


Figure 3. Gradient flow on VGG38

and applies a non-linear transformation:

$$\boldsymbol{x}^{(l)} = f^{(l)}(\boldsymbol{x}^{(l-1)}; W^{(l)}) \tag{1}$$

where (l) denotes the l-th layer in L layer deep network, $f^{(l)}(\cdot, W^{(l)})$ is a non-linear transformation for layer l, and $W^{(l)}$) are the weights of layer l. For instance, $f^{(l)}$ is typically a convolution operation followed by an activation function in convolutional neural networks. The second step involves the backpropagation algorithm, where we calculate the gradient of an error function E (e.g. cross-entropy) for each layer's weight as follows:

$$\frac{\partial E}{\partial W^{(l)}} = \frac{\partial E}{\partial \boldsymbol{x}^{(L)}} \frac{\partial \boldsymbol{x}^{(L)}}{\partial \boldsymbol{x}^{(L-1)}} \cdots \frac{\partial \boldsymbol{x}^{(l+1)}}{\partial \boldsymbol{x}^{(l)}} \frac{\partial \boldsymbol{x}^{(l)}}{\partial W^{(l)}}. \tag{2}$$

This step includes consecutive tensor multiplications between multiple partial derivative terms. The final step involves updating model weights by using the computed $\frac{\partial E}{\partial W^{(l)}}$ with an update rule. The exact update rule depends on the optimizer.

A notorious problem for training deep neural networks is the vanishing/exploding gradient problem (Bengio et al., 1993) that typically occurs in the backpropagation step when some of partial gradient terms in Eq. 2 includes values larger or smaller than 1. In this case, due to the multiple consecutive multiplications, the gradients w.r.t. weights can get exponentially very small (close to 0) or very large (close to infinity) and prevent effective learning of network weights.

Figures 2 and 3 depict the gradient flows through VGG architectures (Simonyan & Zisserman, 2014) with 8 and 38 layers respectively, trained and evaluated for a total of 100 epochs on the CIFAR100 dataset. **[The VGG38 suffers from the Vanishing Gradient Problem as can be seen in Figure 3, the gradients of the weights of the all the convolutional layers are 0 and only the final fully connected layer experiences a peak with gradients of up to 0.0002.**

Each gradient update in backpropagation consists of numerous multiplied factors which keep increasing in number the further to the start of the network the backpropagation goes. These factors are the derivatives of weights, biases and non-linear activation functions which tend to 0. The multiplication of a huge number of these small factors culminates in the vanishing gradient problem.

As a consequence of the vanishing gradient problem, the gradients of the weights of convolutional layers tend to zero meaning thereby, the model isn't able to learn since the weights cannot be updated after each iteration due to negligible gradients. This can be verified from Table 1 below and the VGG38 achieves a training and validation accuracy of 0.01 which is a direct consequence of the vanishing gradient problem.

While the VGG08 achieves a training accuracy of 51.59% and a validation accuracy of 46.84%. It can be seen from Figure 2 that the gradient flow for VGG08 is nothing out of the ordinary and in contrast with Figure 3, it doesn't suffer from the vanishing gradient problem.]

.

3. Background Literature

In this section we will highlight some of the most influential papers that have been central to overcoming the VGP in deep CNNs.

Batch Normalization (Ioffe & Szegedy, 2015) BN seeks to solve the problem of internal covariate shift (ICS), when distribution of each layer's inputs changes during training, as the parameters of the previous layers change. The authors argue that without batch normalization, the distribution of each layer's inputs can vary significantly due to the stochastic nature of randomly sampling mini-batches from your training set. Layers in the network hence must continuously adapt to these high variance distributions which hinders the rate of convergence gradient-based optimizers. This optimization problem is exacerbated further with network depth due to the updating of parameters at layer l

being dependent on the previous $l-1$ layers.

It is hence beneficial to embed the normalization of training data into the network architecture after work from LeCun *et al.* showed that training converges faster with this addition (LeCun et al., 2012). Through standardizing the inputs to each layer, we take a step towards achieving the fixed distributions of inputs that remove the ill effects of ICS. Ioffe and Szegedy demonstrate the effectiveness of their technique through training an ensemble of BN networks which achieve an accuracy on the ImageNet classification task exceeding that of humans in 14 times fewer training steps than the state-of-the-art of the time. It should be noted, however, that the exact reason for BN's effectiveness is still not completely understood and it is an open research question (Santurkar et al., 2018).

Residual networks (ResNet) (He et al., 2016) One interpretation of how the VGP arises is that stacking non-linear layers between the input and output of networks makes the connection between these variables increasingly complex. This results in the gradients becoming increasingly scrambled as they are propagated back through the network and the desired mapping between input and output being lost. He *et al.* observed this on a deep 56-layer neural network counter-intuitively achieving a higher training error than a shallower 20- layer network despite higher theoretical power. Residual networks, colloquially known as ResNets, aim to alleviate this through the incorporation of skip connections that bypass the linear transformations into the network architecture. The authors argue that this new mapping is significantly easier to optimize since if an identity mapping were optimal, the network could comfortably learn to push the residual to zero rather than attempting to fit an identity mapping via a stack of nonlinear layers. They bolster their argument by successfully training ResNets with depths exceeding 1000 layers on the CIFAR10 dataset. Prior to their work, training even a 100-layer was accepted as a great challenge within the deep learning community. The addition of skip connections solves the VGP through enabling information to flow more freely throughout the network architecture without the addition of neither extra parameters, nor computational complexity.

4. Solution overview

4.1. Batch normalization

[Batch Normalization aims to reduce the internal covariate shift of a neural network which is defined as the change in the distribution of network activations due to the change in network parameters during training. Batch Normalization achieves this by fixing the distribution of the layer inputs as the training progresses, it aims to carry this out via layer whitening which is the process of carrying out linear transformation to have zero mean and unit variances. Therefore, for each iteration, batch normalization normalizes the inputs by subtracting their mean and dividing by their standard deviation, where the mean and variance of the activations are estimated based on statistics of the minibatch (which is much more practical than using the entire training set for carrying out normalization). After this a scale coefficient and a scale offset is applied.

Here, $\mathcal{B} = [x_{1...m}]$ is the minibatch

$\mu_{\mathcal{B}} = \frac{1}{|\mathcal{B}|}\sum_{i=1}^{m} x_i$ is the minibatch mean

$\sigma_{\mathcal{B}}{}^2 = \frac{1}{|\mathcal{B}|}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$ is the minibatch variance

Normalization is carried out $\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}{}^2 + \epsilon}}$

After which scaling and shifting is carried out with parameters γ for scaling and β for shifting

$y_i = \gamma\hat{x}_i + \beta$

Application on training and inference of the model -

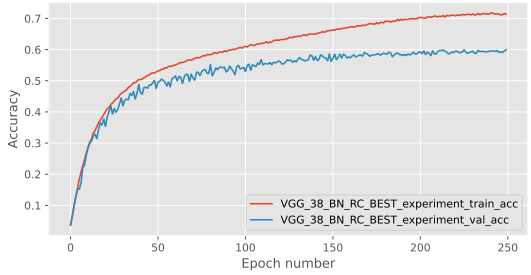
A training BN network N_{BN}^{tr} with trainable parameters Θ and activations $[x^{(k)}]_{k=1}^{K}$ is taken and for each set of activations, the transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ is added to N_{BN}^{tr} and each layer of N_{BN}^{tr} is modified to take $y^{(k)}$ as input instead. After this N_{BN}^{tr} is trained to optimize the parameters $\Theta \cup [\gamma^{(k)}, \beta^{(k)}]_{k=1}^{K}$ and the inference BN network N_{BN}^{inf} is provided with these frozen parameters. Multiple training minibatches $\mathcal{B}$ are then processed and the expectation and variance are calculated as $E[x] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[x] = \frac{m}{m-1}E_{\mathcal{B}}[\sigma_{\mathcal{B}}{}^2]$. In N_{BN}^{inf}, the transform $y = BN_{\gamma,\beta}(x)$ is replaced with $y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}}x + (\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}})$ and we use the entire dataset to estimate the expectation and variance at inference time. This is done since, the noise in the mean and variance estimated from minibatches is no longer required once the model has been trained. Additionally, the model might be required for inference one prediction at a time.

Batch Normalization reduces the probability of vanishing and exploding gradient problems since by normalizing the activations throughout the networks prevents small changes in the parameters to amplify into larger and suboptimal changes in the activations in gradients.

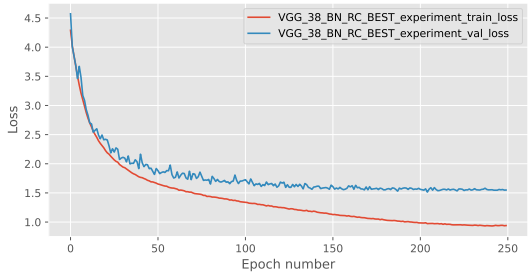
Moreover, with Batch Normalization, backpropagation is unaffected by the scale of its parameters, large weights and large scalars in backpropagation would lead to smaller gradients and therefore, batch normalization stabilises parameter growth and makes training more resilient to the parameter scale. (Ioffe & Szegedy, 2015)] .

4.2. Residual connections

[For deep neural networks, if newly added layers can be trained into an identity function $f(x) = x$, the new model will be at least as effective as the original model and should have a training error no greater than the original model since it would follow a nested function class and would also be able to find a better solution



(a) Accuracy by epoch for VGG38 with Batch Normalisation and Residual Connections



(b) Loss by epoch for VGG38 with Batch Normalisation and Residual Connections

Figure 4. Training curves for VGG38 with Batch Normalisation and Residual Connections with learning rate = 1e-2

to fit the training dataset and reduce training errors due to the increased complexity. Therefore, the identity mapping is the desired underlying mapping to be taken into consideration when adding more layers to the deep neural network. He et al. defined this underlying mapping as $\mathcal{H}(x)$ which was to be learned by the stacked non-linear layers. They hypothesized a residual function $\mathcal{F}(x) := \mathcal{H}(x) - x$ which was the new underlined mapping that the stacked non-linear layers had to learn, the original function therefore becomes $\mathcal{F}(x) + x$. Since the identity mapping $f(x) = x$ was established as the ideal underlying mapping to be learnt, the residual mapping $\mathcal{F}(x) := \mathcal{H}(x) - x$ became easier to learn since the weights and biases of the stacked layers can be lowered to 0 to learn the identity mapping. (He et al., 2016)

Residual connections are skip connections which allow gradients to flow through directly during backpropagation without passing through the stacked non-linear layers and activations, since it is the non-linear activations that cause the vanishing gradient problem due to the multiplication of multiple small gradient values, therefore, residual connections address the vanishing gradient problem and hence facilitate deeper model architectures.

] .

5. Experiment Setup

[ ]

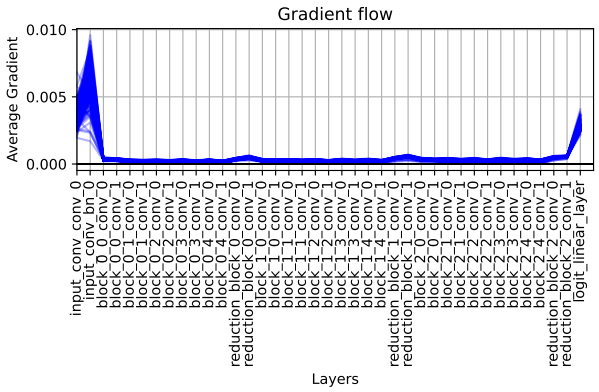


Figure 5. Gradient Flow on VGG38 with Batch Normalisation and Residual Connections with learning rate = 1e-2

[ ]

[Question Table 1 - Fill in Table 1 with the results from your experiments on

1. ***VGG38 BN (LR 1e-3)*, and**
2. ***VGG38 BN + RC (LR 1e-2).***

]

We conduct our experiment on the CIFAR-100 dataset (Krizhevsky et al., 2009), which consists of 60,000 32x32 colour images from 100 different classes. The number of samples per class is balanced, and the samples are split into training, validation, and test set while maintaining balanced class proportions. In total, there are 47,500; 2,500; and 10,000 instances in the training, validation, and test set, respectively. Moreover, we apply data augmentation strategies (cropping, horizontal flipping) to improve the generalization of the model.

With the goal of understanding whether BN or skip connections help fighting vanishing gradients, we first test these methods independently, before combining them in an attempt to fully exploit the depth of the VGG38 model.

All experiments are conducted using the Adam optimizer with the default learning rate (1e-3) – unless otherwise specified, cosine annealing and a batch size of 100 for 100 epochs. Additionally, training images are augmented with random cropping and horizontal flipping. Note that we do not use data augmentation at test time. These hyperparameters along with the augmentation strategy are used to produce the results shown in Figure 1.

When used, BN is applied after each convolutional layer, before the Leaky ReLU non-linearity. Similarly, the skip connections are applied from before the convolution layer to before the final activation function of the block as per Figure 2 of (He et al., 2016)

6. Results and Discussion

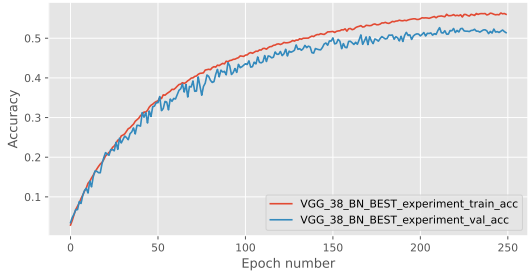
Model	LR	# Params	Train loss	Train acc	Val loss	Val acc
VGG08	1e-3	60 K	1.74	51.59	1.95	46.84
VGG38	1e-3	336 K	4.61	00.01	4.61	00.01
VGG38 BN	1e-3	339 K	1.53	56.02	1.75	51.39
VGG38 RC	1e-3	336 K	1.33	61.52	1.84	52.32
VGG38 BN + RC	1e-3	339 K	1.26	62.99	1.73	53.76
VGG38 BN	1e-2	339 K	1.70	52.28	1.99	46.72
VGG38 BN + RC	1e-2	339 K	0.93	71.33	1.55	59.95

Table 1. Experiment results (number of model parameters, Training and Validation loss and accuracy) for different combinations of VGG08, VGG38, Batch Normalisation (BN), and Residual Connections (RC), LR is learning rate.

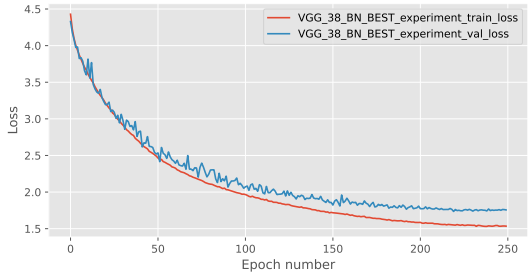
[The experiments were conducted on the VGG model architecture with VGG08 consisting of 7 convolutional layers and 1 fully connected layer and VGG38 consisting of 37 convolutional layers and 1 fully connected layer constructed using two basic building blocks of which a certain combination is used one of these blocks is called the Convolutional Processing Block and the other the Convolutional Dimensionality Reduction Block. Both of these blocks consist of 2 convolutional layers, with the dimensionality reduction block implementing average pooling (with a reduction factor of 2 which is essentially an average pooling conducted with a 2x2 kernel with a stride of 2 in contrast with the paper in which max pooling was implemented instead) after the first convolutional layer. Following the original VGG architecture, a padding of 1 was utilised with 3x3 convolutions. Since the models were to be used on the CIFAR-100 dataset which consists of 32x32 images, 32 filters were utilized for each convolutional layer in contrast with the paper (Simonyan & Zisserman, 2014) where the VGG models trained on the ImageNet dataset and a varying number of filters ranging from 64 to 512 were utilised for the convolutional layers. In the architecture for VGG08, 3 convolutional dimensionality reduction blocks were utilized in addition to an input convolutional layer and in the architecture for VGG38, 15 convolutional processing blocks are utilized and 3 convolutional dimensionality reduction blocks are utilized. For the first two experiments, batch normalization and residual connections are not implemented, batch normalization is only implemented for the input convolutional layer and the leaky ReLU activation function is utilised for all the models in the experiments. For VGG38 with BN and with BN + RC, batch normalization was applied after each convolutional layer before the Leaky ReLU activation.

As can be seen from the table, the vanilla VGG38 with no batch normalization or residual connections implemented despite being more complex than the VGG08, performs much worse than the VGG08 due to the vanishing gradient problem as can be seen from the Gradient Flow plot in Figure 3 achieving a training and validation accuracy of only 0.001, practically learning nothing while the VGG08 achieved a training accuracy of 51.59% and a validation accuracy of 46.84% after a 100 epochs of training. In order to address the vanishing gradient problem, batch normalization was implemented in the VGG38 BN which was trained for 250 epochs with a batch size of 264, a learning rate of 1e-3, with a weight decay coefficient of 1e-6, the VGG38 BN managed to achieve a much higher validation accuracy of 51.39% and a higher training accuracy of 56.02% than the VGG08, while achieving an accuracy of 51.42% on the test set. The VGG38 BN being a much more complex model (with 339K parameters) than the VGG08 took much longer to converge, although it achieved a much lower generalizaton gap than the VGG08 showing the advantages of utilizing more complex models. Another VGG38 BN was trained using a learning rate of 1e-2, a batch size of 100, for 100 epochs achieved results worse than both the VGG08 and VGG38 BN (trained with a learning rate of 1e-3), even though it managed to reduce the training loss more than the VGG08, it achieved a performance roughly similar to the VGG08 supposedly due to a higher learning rate. From these 4 experiments it can be observed that the implementation of batch normalization significantly improved the learning capacity of the VGG38 model by sufficiently addressing the problem of vanishing gradients and achieving a performance comparable to and even better than the VGG08 on the CIFAR-100 dataset. In Figure 6, the training curves for the VGG38 BN trained with a learning rate of 1e-3 for 250 epochs with a batch size of 264 can be observed, the model sufficiently reduces the training error in addition to which it has an optimally low generalization gap

The VGG38 RC implemented residual connections in the Convolutional Processing Block of the VGG38 and for a learning rate of 1e-3 and batch size of 100, trained for 100 epochs managed to reduce the training loss more than either of the VGG38 BN models while the VGG38 RC and VGG38 BN with a learning rate of 1e-3 achieved a better validation accuracy than both the VGG38 BN models and reduced the training loss better than both the VGG38 BN models. It can be concluded that residual connections reduced the probability of the vanishing gradients problem more optimally than batch normalization. This could be explained due to the fact the batch normalization primarily addresses the internal covariate shift of a neural network, reducing the probability of the dying ReLU problem and allowing for the model to be trained at higher learning rates in addition to which



(a) Accuracy by epoch for VGG38 with Batch Normalisation



(b) Loss by epoch for VGG38 with Batch Normalisation

Figure 6. Training curves for VGG38 with Batch Normalisation with learning rate = 1e-3

batch normalization stabilises parameter growth and makes training and backpropagation more resilient to parameter scale indirectly addressing the problem of vanishing gradients whereas residual connections directly address the problem of backpropagation through non-linear activations in a deep neural network (which is the primary cause of the vanishing gradient problem) with the help of skip connections which help in carrying out backpropagation without passing the gradients through non linear activations. Therefore, it can be concluded that residual connections are more useful for reducing the probability of vanishing gradient problem in deep neural networks.

The two best performing models out of all the experiments in the table were implementations of VGG38 with batch normalization and residual connections. The implementation of these models consisted of a batch normalization after each convolutional layer and residual connections in the convolutional processing blocks of the VGG38. One of the VGG38 BN + RC models was trained with a learning rate of 1e-3 with a batch size of 100, for 100 epochs with a weight decay coefficient of 0 while the other was trained with a learning rate of 1e-2 with a batch size of 264 for 250 epochs with a weight decay coefficient of 1e-6. It was observed that the model trained on a learning rate of 1e-3 achieved a training accuracy of 62.99% and a validation accuracy of 53.76% while the one trained on a learning rate of 1e-2 achieved a training accuracy of 71.33% , a validation accuracy of 59.95% and a test accuracy of 61.4%, the latter probably outperforms the former due to the fact that it has been trained for more epochs although the model trained on the lower learning rate would probably manage to outperform the other if trained for longer.

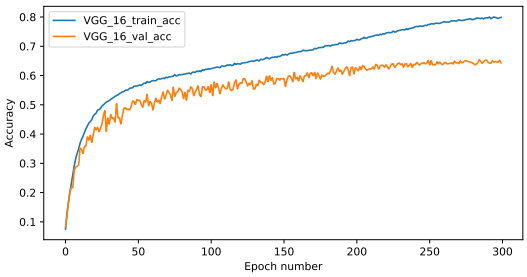
It can be concluded from the above experiments that batch normalization and residual connections not only sufficiently addressed the problem of vanishing gradients, but facilitated the training of deeper models even at higher learning rates of 1e-2 which allowed us to carry out effective depth scaling.

Additional Experiments -

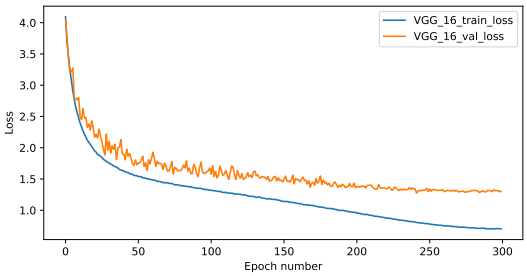
Certain additional experiments were conducted, varying the number of layers of the model and implementing residual connections in both the convolutional processing block and the convolutional dimensionality reduction block in contrast with the previous experiments analysed above in which residual connections were implemented only in the convolutional processing block. Since the convolutional dimensionality reduction block utilizes average pooling, in order to make the dimensions equal, a convolutional layer with batch normalization was applied in the shortcut in order to implement the skip connection. Additionally, both the convolutional processing block and the convolutional dimensionality reduction block were made similar to the residual blocks of the ResNet-50 model, with 3, 3x3 convolutional layers in each block instead of 2, 3x3 convolutions per block (which was the setup for the previous above). Therefore, for the new setup, the convolutional processing block consisted of 3 convolutional layers while the convolutional dimensionality reduction block consisted of 4 convolutional layers (1 additional convolutional layer in the shortcut).

Experiments were conducted on three models - VGG16 BN+RC, VGG50 BN+RC and the VGG150 BN+RC consisting of 15, 49 and 149 convolutional layers respectively. VGG16 BN+RC was built using 2 convolutional processing blocks and 2 convolutional dimensionality reduction blocks while the VGG50 BN+RC was built using 12 convolutional processing blocks and 3 convolutional dimensionality reduction blocks and the VGG150 BN+RC was built using 44 convolutional processing blocks and 4 convolutional dimensionality reduction blocks. For all 3 experiments, a batch size of 264, a learning rate of 1e-3 and a weight decay of 1e-3 was utilized using the Adam optimizer. The VGG16 BN+RC was trained for 300 epochs while the VGG50 BN+RC and VGG150 BN+RC were trained for 250 epochs.

Results - After 300 epochs, the VGG16 BN+RC achieved a training accuracy of 79.86% and a valdation accuracy of 64.32% and a accuracy of 64.22% on the test set. The VGG50 BN+RC achieved a training accuracy of 71.55%, a validation accuracy of 61.89% and an accuracy of 62.7% on the test set after 250 epochs, and the VGG150 BN+RC achieved a training accuracy of 74.19% a validation accuracy of 64.37% and an accuracy of 63.6% on the test set. The training plots for all 3 of these experiments can be seen in Figure 7, Figure 8

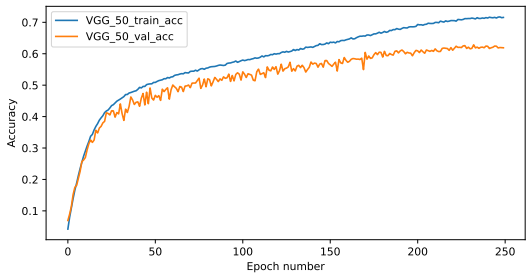


(a) Accuracy by epoch for VGG16 with Batch Normalisation and Residual Connection

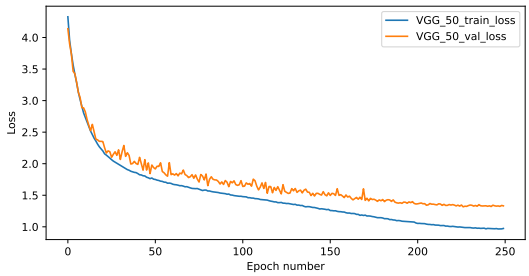


(b) Loss by epoch for VGG16 with Batch Normalisation and Residual Connection

Figure 7. Additional Experiment - Training curves for VGG16 with Batch Normalisation and Residual Connection



(a) Accuracy by epoch for VGG50 with Batch Normalisation and Residual Connection



(b) Loss by epoch for VGG50 with Batch Normalisation and Residual Connection

Figure 8. Additional Experiment - Training curves for VGG50 with Batch Normalisation and Residual Connection

and Figure 9.

Conclusions - It can be concluded from the additonal experiments that the implementation of BN and RC allowed for models as deep as the VGG150 to be trained. This experiment was inspired by the paper on ResNets (**He et al., 2016**) in which the authors trained models as deep as the ResNet-101, ResNet-152 and ResNet-1000 thanks to residual connections sufficiently addressing the problem of vanishing gradients.

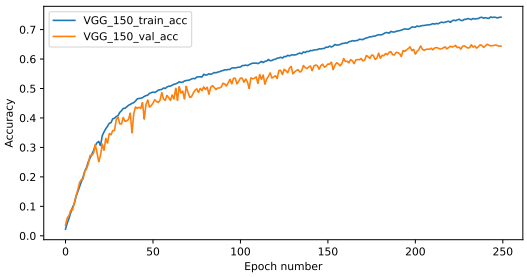
The implementation of BN and RC to less complex models like the VGG16 allowed the model to achieve results comparable to and even better than more complex models like the VGG38, VGG50 and the VGG150.

The application of residual connections to all the blocks yielded better results than in the previous setup for the VGG38 experiments where residual connections were utilised only in the convolutional processing block and not in the convolutional dimensionality reduction block.

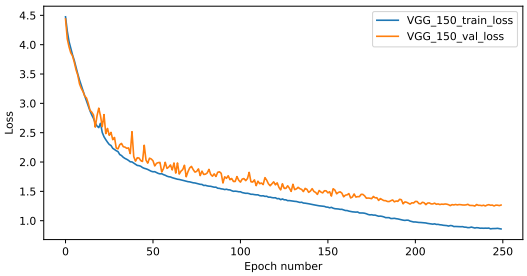
] .

7. Conclusion

[It can be concluded from multiple experiments conducted that the implementation of batch normalization and residual connections to the VGG architecture (**Simonyan & Zisserman, 2014**) facilitated the training of deeper models by addressing the vanishing gradient problem experienced by deeper models, allowing effective depth scaling while at the same time improving the performance of even the shallower architectures on the CIFAR-100 dataset. For instance, the effectiveness of batch normalization (**Ioffe & Szegedy, 2015**) is evident from the fact that in all of the current state of the art image recognition models, convolutional layers go hand in hand with batch normalization in a "conv block" consisting of (Conv + BN + ReLU). However, one of the major drawbacks of batch normalization is that it requires a sufficiently large batch size for accurate computation of minibatch statistics. New strategies such as Group Normalization (**Wu & He, 2018**) have been suggested to address this problem which divide the channels into groups and computes the mean and variance of each group for the purpose of normalization, thereby making the computation independent of batch size. Additionally, residual connections (**He et al., 2016**) have to be recognised as a revolutionary breakthrough in the field of computer vision facilitating massive depth scaling inspiring deeper convnets with the ability to capture richer and more complex features and generalize well on new tasks. One of the models inspired from residual connections was the DenseNet model architecture (**Huang et al., 2017**) which utilized the concatenation of feature maps from previous layers onto the inputs of each and every future layer to achieve a deep model. Additionally, there has been research conducted to increase the speed and accuracy of residual networks by making use of multi-residual networks which implement multiple skip connections resulting in a range of ensembles contributing towards gradient update (**Abdi & Nahavandi, 2016**). While the primary purpose residual connections



(a) Accuracy by epoch for VGG150 with Batch Normalisation and Residual Connection



(b) Loss by epoch for VGG150 with Batch Normalisation and Residual Connection

Figure 9. Additional Experiment - Training curves for VGG150 with Batch Normalisation and Residual Connection

have been utilised for is depth scaling to build deeper and deeper models, we must take into consideration the findings of (He et al., 2016) regarding the ResNet-1000 performing only as good as the ResNet-101 while being significantly deeper, in addition to the VGG150 BN+RC achieving a marginally lower test accuracy than the VGG16 BN+RC in the additional experiments conducted above, despite being significantly deeper and much more complex. Additionally, the findings of (Tan & Le, 2020) regarding scaling up the depth of a baseline have to be taken into account, when scaling up along a single dimension like depth, the accuracy improves up until a certain point after which it saturates. Therefore, the focus of future works should not only be on depth scaling, but width scaling and resolution scaling also have to be taken into account in order to carry out compound scaling. Additionally, arbitrary scaling is something that needs to be avoided and we need to utilize Neural Architecture Search as implemented in the current state of the art EfficientNet architecture (Tan & Le, 2020).] .

References

Abdi, Masoud and Nahavandi, Saeid. Multi-residual networks. *CoRR*, abs/1609.05672, 2016. URL http://arxiv.org/abs/1609.05672.

Bengio, Yoshua, Frasconi, Paolo, and Simard, Patrice. The problem of learning long-term dependencies in recurrent networks. In *IEEE international conference on neural networks*, pp. 1183–1188. IEEE, 1993.

Bishop, Christopher M et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Huang, Gao, Liu, Zhuang, Van Der Maaten, Laurens, and Weinberger, Kilian Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Krizhevsky, Alex, Hinton, Geoffrey, et al. Learning multiple layers of features from tiny images. 2009.

LeCun, Yann A, Bottou, Léon, Orr, Genevieve B, and Müller, Klaus-Robert. Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–48. Springer, 2012.

Rumelhart, David E, Hinton, Geoffrey E, and Williams, Ronald J. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Santurkar, Shibani, Tsipras, Dimitris, Ilyas, Andrew, and Mądry, Aleksander. How does batch normalization help optimization? In *Proceedings of the 32nd international conference on neural information processing systems*, pp. 2488–2498, 2018.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Tan, Mingxing and Le, Quoc V. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

Wu, Yuxin and He, Kaiming. Group normalization. *CoRR*, abs/1803.08494, 2018. URL http://arxiv.org/abs/1803.08494.