MLP Coursework 2

s2015345

Abstract

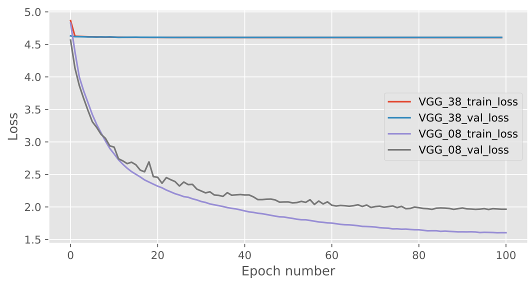
Deep neural networks have become the state-of-the-art in many standard computer vision problems thanks to their powerful representations and availability of large labeled datasets. While very deep networks allow for learning more levels of abstractions in their layers from the data, training these models successfully is a challenging task due to problematic gradient flow through the layers, known as vanishing/exploding gradient problem. 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.

1. Introduction

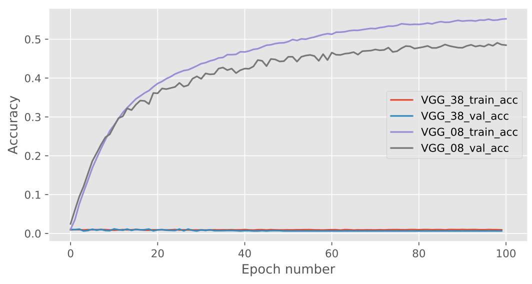
Despite the remarkable progress of modern convolutional neural networks (CNNs) in image classification problems (Simonyan & Zisserman, 2014; He et al., 2016), training very deep networks is a challenging procedure. One of the major problems is the Vanishing Gradient Problem (VGP), a phenomenon where the gradients of the error function with respect to network weights shrink to zero, as they backpropagate to earlier layers, hence preventing effective weight updates. This phenomenon is prevalent and has been extensively studied in various deep neural networks including feedforward 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 occurring in the VGG38 model[1] and addressing it by implementing two standard solutions. In particular, we first study a "broken" network in terms of its gradient flow, L1 norm of gradients with respect to its weights for each layer and contrast it to ones in the healthy and shallower 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 CIFAR100 (pronounced as 'see far 100') dataset and present the results. The results show that though separate use of BN and RC does mitigate the vanishing/exploding gradient problem, therefore enabling effective training of the VGG38 model, the best results are obtained by combining both BN and RC.

[1]VGG stands for the Visual Geometry Group in the University of Oxford.



(a) Cross entropy error per epoch



(b) Classification accuracy per epoch

Figure 1. Training curves for VGG08 and VGG38 in terms of (a) cross-entropy error and (b) classification accuracy

2. Identifying training problems of a deep CNN

[Question Figure 3]

Concretely, training deep neural networks typically involves

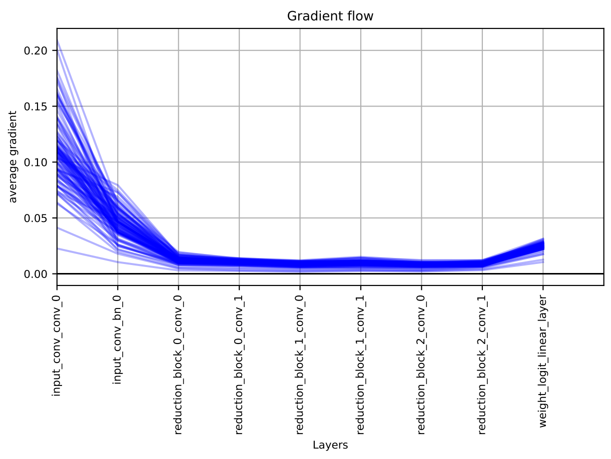


Figure 2. Gradient flow on VGG08

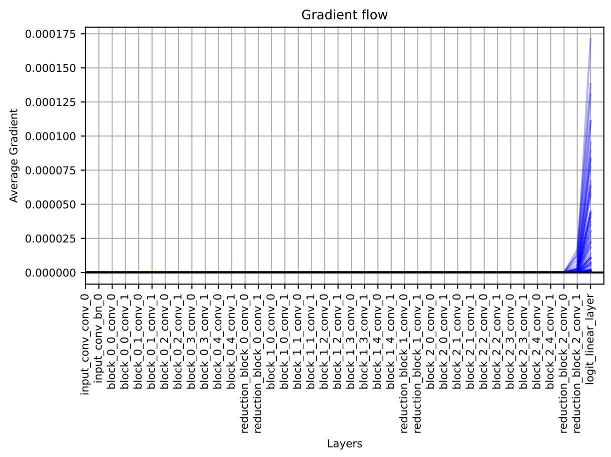


Figure 3. Gradient Flow on VGG38

three steps: forward pass, backward pass (or backpropagation algorithm (Rumelhart et al., 1986)) and weight update. The first step involves passing the input $\boldsymbol{x}^{(0)}$ 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 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:

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

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. **[Based on the results in Figures 2 and 3, we can conclude that the VGG38 network suffers from the Vanishing Gradient problem. This is because the gradients of the weights of most convolutional layers are 0, with the only exception being the final fully-connected layer, where the gradient peaks up to roughly 0.000175.**

In backpropagation, each gradient update comprises of several factors multiplied together, with the number of factors increasing as the backpropagation algorithm gets closer to the start of the network. These factors, which are the derivatives of the weights, biases, and non-linear activation functions, all tend to 0. Therefore, by multiplying a large number of these small factors, the result of this multiplication will tend to 0 as well, hence resulting in the vanishing gradient problem.

As a result of the vanishing gradient problem, the gradients of the weights of the convolutional layers tend to 0. This means that the weights cannot be updated after each iteration (because of the negligible gradients), and therefore the model is unable to learn. We see this in effect from the results in Table 1, where the training and validation accuracies are both 0.01, a direct consequence of the vanishing gradient problem. We further see this in Figure 1, where in Figure 1a the cross entropy error for both training and validation does not decrease over more epochs for VGG38, and we also see that in Figure 1b, the classification accuracy for both training and validation for VGG38 once again fails to improve over time.

Meanwhile, we see (from Table 1) that the VGG08 network achieves a training accuracy of 51.59% and a validation accuracy of 46.84%. We can also see from Figure 2 that the gradient flow for VGG08 looks quite standard (unlike the one in Figure 3), which indicates that the VGG08 network does not suffer from the vanishing gradient problem. This is further supported by the results for VGG08 in Figure 1, with Figure 1a showing that the cross entropy error for both training and validation decreases as the number of epochs increases, and in Figure 1b we see that the classification accuracy for both training and testing increases over time.] .

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) A well-known way of mitigating the VGP is proposed by He *et al.* in (He et al., 2016). In their paper, the authors depict the error curves of a 20 layer and a 56 layer network to motivate their method. Both training and testing error of the 56 layer network are significantly higher than of the shallower one.

[In Figure 1 from (He et al., 2016), we can see the training and testing errors for both the 20 layer and the 56 layer networks; this plot shows that both training and testing errors are higher for the deeper network (56 layer) compared to the shallower one (20 layer). These results are further supported by the left-hand-side plot in Figure 4 of (He et al., 2016), where we see a comparison between an 18 layer and a 34 layer network, with the shallower one once again outperforming the deeper model, this time on a different dataset (Figure 1 was based on CIFAR-10 whereas Figure 4 was based on ImageNet).

The authors note the following observation: when deeper networks begin to converge, a degradation problem is observed, where the accuracy of the model gets saturated and then sharply decreases. They also link this to the increasing depth of the network, suggesting that greater network depth (and therefore greater network capacity, as it would be able to fit a wider variety of functions) could result in worse performance over time. These conclusions are similar to those found by (Subramanian & Simon, 2013); however, Subramanian and Simon (2013) concluded that there is a direct relation between network capacity and overfitting, namely the greater the network capacity, the higher the chances of overfitting.

By contrast, (He et al., 2016) suggest that their results are not a cause of overfitting, but rather an issue caused by model optimization, where not all models are equally easy to fine-tune in order to produce comparable training and testing accuracies. Their conclusions are further supported by the reesults in Figure 1, where we see that the deeper network also performs worse on the training data, and not just the testing data. Nonetheless, the optimization issue mentioned in their paper could be due to the Vanishing Gradient Problem, where the greater network capacity (and network depth) results in a longer distance for the backpropagation algorithm to travel through the network, therefore resulting in very small gradients that prevent the weights from updating and the network from being able to learn.

Other factors causing the difference in performance between the 20 layer and 56 layer network (besides the VGP and overfitting) could be the larger number of parameters present in the deeper network (making the model harder to optimize) as well as any regularization techniques being used.].

Residual networks, colloquially known as ResNets, aim to alleviate VGP 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

BN has been a standard component in the state-of-the-art convolutional neural networks (He et al., 2016; Huang et al., 2017). Concretely, BN is a layer transformation that is performed to whiten the activations originating from each layer. As computing full dataset statistics at each training iteration would be computationally expensive, BN computes batch statistics to approximate them. Given a minibatch of B training samples and their feature maps $X = (\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^B)$ at an arbitrary layer where $X \in \mathbb{R}^{B \times H \times W \times C}$, H, W are the height, width of the feature map and C is the number of channels, the batch normalization first computes the following statistics:

$$\mu_c = \frac{1}{BWH} \sum_{n=1}^{B} \sum_{i,j=1}^{H,W} \boldsymbol{x}_{cij}^n \tag{3}$$

$$\sigma_c^2 = \frac{1}{BWH} \sum_{n=1}^{B} \sum_{i,j=1}^{H,W} (\boldsymbol{x}_{cij}^n - \mu_c)^2 \tag{4}$$

where c, i, j denote the index values for y, x and channel coordinates of feature maps, and $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ are the mean and variance of the batch.

BN applies the following operation on each feature map in batch B for every c, i, j:

$$\mathrm{BN}(\boldsymbol{x}_{cij}) = \frac{\boldsymbol{x}_{cij} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}} * \gamma_c + \beta_c \tag{5}$$

where $\gamma \in \mathbb{R}^C$ and $\beta \in \mathbb{R}^C$ are learnable parameters and ϵ is a small constant introduced to ensure numerical stability.

At inference time, using batch statistics is a poor choice as it introduces noise in the evaluation and might not even be well defined. Therefore, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are replaced by running averages of the mean and variance computed during training, which is a better approximation of the full dataset statistics.

Recent work has shown that BatchNorm has a more fundamental benefit of smoothing the optimization landscape during training (Santurkar et al., 2018) thus enhancing the predictive power of gradients as our guide to the global minimum. Furthermore, a smoother optimization landscape should additionally enable the use of a wider range of learning rates and initialization schemes which is congruent with the findings of Ioffe and Szegedy in the original BatchNorm paper (Ioffe & Szegedy, 2015).

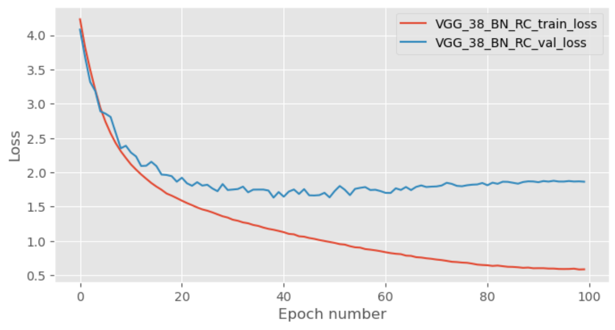
4.2. Residual connections

Residual connections are another approach used in the state-of-the-art Residual Networks (He et al., 2016) to tackle the vanishing gradient problem. Introduced by He et. al. (He et al., 2016), a residual block consists of a convolution (or group of convolutions) layer, "short-circuited" with an identity mapping. More precisely, given a mapping $F^{(b)}$ that denotes the transformation of the block b (multiple consecutive layers), $F^{(b)}$ is applied to its input feature map $\boldsymbol{x}^{(b-1)}$ as $\boldsymbol{x}^{(b)} = \boldsymbol{x}^{(b-1)} + F(\boldsymbol{x}^{(b-1)})$.

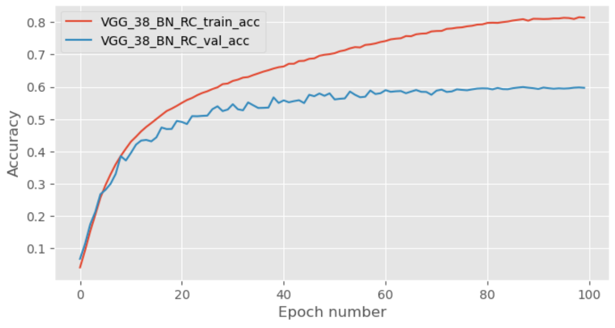
Intuitively, stacking residual blocks creates an architecture where inputs of each blocks are given two paths : passing through the convolution or skipping to the next layer. A residual network can therefore be seen as an ensemble model averaging every sub-network created by choosing one of the two paths. The skip connections allow gradients to flow easily into early layers, since

$$\frac{\partial \boldsymbol{x}^{(b)}}{\partial \boldsymbol{x}^{(b-1)}} = \mathbb{1} + \frac{\partial F(\boldsymbol{x}^{(b-1)})}{\partial \boldsymbol{x}^{(b-1)}} \tag{6}$$

where $\boldsymbol{x}^{(b-1)} \in \mathbb{R}^{C \times H \times W}$ and $\mathbb{1}$ is a $\mathbb{R}^{C \times H \times W}$-dimensional tensor with entries 1 where C, H and W denote the number of feature maps, its height and width respectively. Importantly, $\mathbb{1}$ prevents the zero gradient flow.



(a) Cross entropy error per epoch



(b) Classification accuracy per epoch

Figure 4. Training curves for VGG38 with Batch Normalisation and Residual Connections and learning rate 1e-2 in terms of (a) cross-entropy error and (b) classification accuracy

5. Experiment Setup

[Question Figure 4]

[Question Figure 5]

[Question Table 1]

We conduct our experiment on the CIFAR100 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

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	339K	1.59	55.17	1.95	47.04
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	339K	0.58	81.39	1.86	59.68

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.

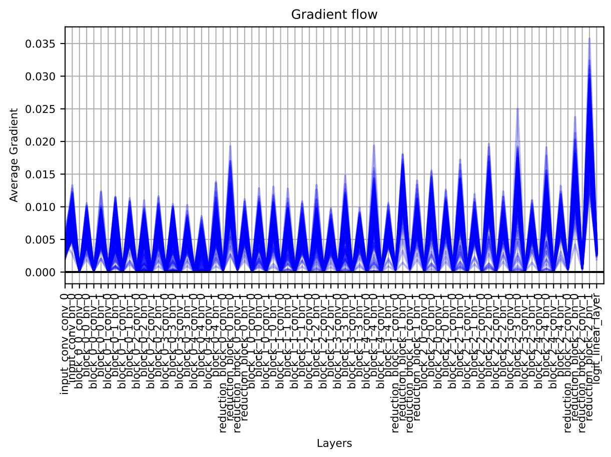


Figure 5. Gradient Flow on VGG38 with Batch Normalisation and Residual Connections and learning rate 1e-2

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 Fig. 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 Fig. 2 of (He et al., 2016). Note that adding residual connections between the feature maps before and after downsampling requires special treatment, as there is a dimension mismatch between them. Therefore in the coursework, we do not use residual connections in the down-sampling blocks. However, please note that batch normalization should still be implemented for these blocks.

5.1. Residual Connections to Downsampling Layers

[Incorporating residual connections into the downsampling layers of a neural network requires specific changes to the network's architecture. One key aspect of these changes is the need for dimension matching, ensuring the input and output of a downsampling layer are aligned in terms of spatial dimensions and the number of channels. This input-output alignment could be achieved through the use of strided convolutions for spatial reduction and 1x1 convolutions for channel adjustment. Moreover, it is worth noting that such residual connections often bypass non-linear activations (such as ReLu) in order to preserve information and ensure a smooth gradient flow. These implementation considerations is particularly relevant for downsampling layers, as the shortcut connections are strategically placed to connect the inputs and outputs directly (He et al., 2016).

One implementation approach suggested by (He et al., 2016) is through the use of direct shortcut connections. This approach involves creating a direct path from the input to the output of a downsampling layer, where the input often undergoes downsampling via strided convolution or pooling to ensure dimension compatibility. The main advantage of this approach is its simplicity and directness, by creating an efficient gradient flow through the network and hence mitigating the vanishing gradient problem. However, the drawbacks of this implementation are the challenges in aligning input and output dimensions, as well as the increased computational demands created by the additional downsampling step.

Another approach discussed by (He et al., 2016) is that of bottleneck residual blocks. In this approach, the input is first passed through a convolutional layer with fewer filters before downsampling, and the output of the downsampling is then passed through another convolutional operation before being added to the output. A benefit of this implementation is the reduced number of parameters, which helps improve computational efficiency. Another benefit is that this approach enables a more refined transformation of features before downsampling. However, the main drawback of this method is the added complexity and potential risk of information loss due to the initial convolution with fewer filters, as this could have an impact on the network's overall performance.

Ultimately, the choice between these two approaches

boils down to the nature of the task at hand, with the direct shortcut approach being preferred for simpler tasks and scenarios with high computational constraints, whereas the bottleneck residual blocks approach would be preferred for more complex tasks where advanced feature transformation is necessary.] .

6. Results and Discussion

[Based on the results in Table 1, we can clearly see that VGG38 with both Batch Normalization (BN) or Residual Connections (RC) shows much better results than the original VGG38: the validation accuracy has dramatically improved form only 0.01% to above 53.76% (in both cases using a learning rate of 1e-3). This implies that both BN and RC are effective ways to mitigate the Vanishing Gradient Problem (VGP). Moreover, we note that all the modified VGG38 models achieve a comparable (if not better) performance to the VGG08 model, which agrees with the theoretical conclusions from literature that deeper networks possess a greater generalization ability on unseen data.

By comparing the results of VGG38 BN with learning rate 1e-3 and VGG38 BN with learning rate 1e-2, we observe that the validation accruacy does not decrease significantly (47.04% and 46.72% respectively). We observer the opposite effect between VGG38 BN + RC with learning rates 1e-3 and 1e-2, where the validation accuracy slightly increases from 53.76% to 59.68%. However, since both changes are not highly significant, this shows that BN more stable for large step training and allows for a larger learning rate.

As for the comparison between VGG38 BN and VGG38 RC (both with learnin rate 1e-3), we see that the BN method leads to a validation accuracy of 47.04%, whereas RC achieves a validation accuracy of 52.32%, higher than that of BN. By also considering the number of parameters of each model, with BN having 339K and 336K, we can conclude that RC is a more powerful tool, achieving a higher validation accuracy whilst also not introducing additional parameters.

Amongst all results, however, we see that the best performance was achieved when applying both BN and RC to the VGG38 model. This shows that these two techniques complement each other nicely, and result in much greater improvement than just through using one of them. We observed the best model to be VGG38 BN + RC with learning rate of 1e-2, resulting in both the highest training accuracy and the highest validation accuracy (81.39% and 59.68% respectively). We also see from Figure 5 that this implementation most definitely addresses the Vanishing Gradient Problem; this is because, when the gradient flows across the BN layers, it gets enlarged before further backpropagation, thus alleviating the exponential decrease effect seen in Figure 3. We also see the training and validation curves (for both cross entropy and accuracy) for this new network in Figure 4, where we observe that both cross entropy curves are now downward sloping, and both accuracy curves are upwards sloping, suggesting that the broken network has indeed been fixed. However, one thing we could note from the results in Figure 4 is that the model may be suffering from overfitting, as the training accuracy is much higher than the validation accuracy, and similarly the cross entropy error is much lower for training than for validation.

To improve model performance, further experiments might consider training with deeper VGG models. From an engineering point of view, we can use neural architecture search (NAS) to search for the optimal network architecture across different VGG models. For instance, (Liu et al., 2018) proposed a progressive Neural Architecture Search method which uses reinforcement learning and evolutionary algorithms to search the structure space of convolutional neural networks (CNNs). In this way, we might be able to explore the full potential of the better generalization capabilities of deeper models.

Another potential direction for future experiments would be to consider integrating the proposed solutions in section 5.1 of this report. By incorporating these changes into the network architecture, we can investigate the effect of RC on downsampling and also compare the strength of these two methods proposed by (He et al., 2016).

The two avenues for future experiments proposed only consider the scope of BN and RC. However, as mentioned before, we saw that the best model (VGG38 BN + RC with learning rate 1e-2) showed potential signs of overfitting. Another avenue for further experimentation could be to shift the focus away from BN and RC, and consider integrating these tools with regularization techniques such as L1 and L2 regularization. This would allow us to explore if these techniques can be combined effectively to produce powerful models that generalize even better to unseen data than by simply using one approach on its own.] .

7. Conclusion

[Given the results of our experiments, we can conclude that the addition of batch normalization and residual connections to the current VGG acrchitecture (Simonyan & Zisserman, 2014) facilitated the training of deeper models, such as the VGG38, by addressing the vanishing gradient problem and allowing effective depth scaling on the CIFAR-100 dataset.

Our findings in regards to batch normalization (Ioffe & Szegedy, 2015) are not surprising: many state-of-the-art image recognition models make use of batch normalization, by integrating it withing the architecture of the convolutional neural network in the form Convolutional layer + Batch Normalization layer + ReLu). However, a major drawback of batch normalization is

that it requires a sufficiently large batch size for accurate computation of minibatch statistics. One proposed solution to address this was through Group Normalization (Wu & He, 2018), where the channels are divided into groups and we compute the mean and variance of each group for the purpose of normalization, thus making the computation independent of batch size.

Moreover, our results regarding residual connections (He et al., 2016) also comply with current literature, as they show how, by facilitating depth scaling, deeper convolutional neural networks are able to capture richer and more complex features and generalize well on new tasks. A modern example inspired by residual networks is that of 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, extensive research has been conducted to increase the speed and accuracy of residual networks through the use of multi-residual networks, which implement multiple skip connections, resulting in a range of ensembles contributing towards gradient update (Abdi & Nahavandi, 2016).

However, when considering these findings, we should also account for the findings of (Tan & Le, 2019). In their paper, the authors discuss scaling up the depth of a baseline, suggesting that when scaling up along a single dimension like depth, the accuracy of the model improves only up until a certain point, after which it saturates. Therefore, future works should not only focus on depth scaling, but also width scaling and resolution scaling in order to carry out compound scaling. Additionally, one thing that needs to be avoided is arbitrary scaling, with a better alternative to use being Neural Architecture Search as implemented in the EfficientNet architecture (Tan & Le, 2019).] .

References

Abdi, Masoud and Nahavandi, Saeid. Multi-residual networks: Improving the speed and accuracy of residual networks. *arXiv preprint arXiv:1609.05672*, 2016.

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.

Liu, Chenxi, Zoph, Barret, Neumann, Maxim, Shlens, Jonathon, Hua, Wei, Li, Li-Jia, Fei-Fei, Li, Yuille, Alan, Huang, Jonathan, and Murphy, Kevin. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 19–34, 2018.

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.

Subramanian, Jyothi and Simon, Richard. Overfitting in prediction models–is it a problem only in high dimensions? *Contemporary clinical trials*, 36(2):636–641, 2013.

Tan, Mingxing and Le, Quoc. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pp. 6105–6114. PMLR, 2019.

Wu, Yuxin and He, Kaiming. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.