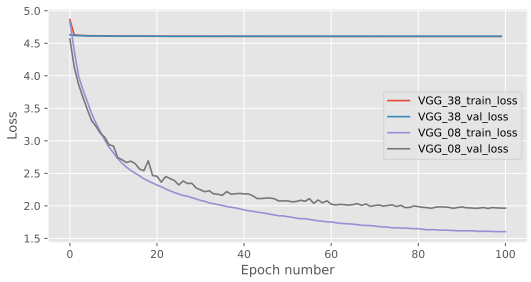
MLP Coursework 2

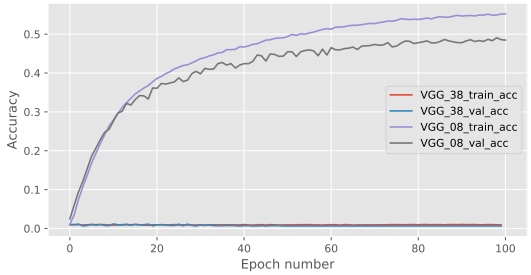
s1858933

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.



(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

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.

2. Identifying training problems of a deep CNN

[ ]

Concretely, training deep neural networks typically involves

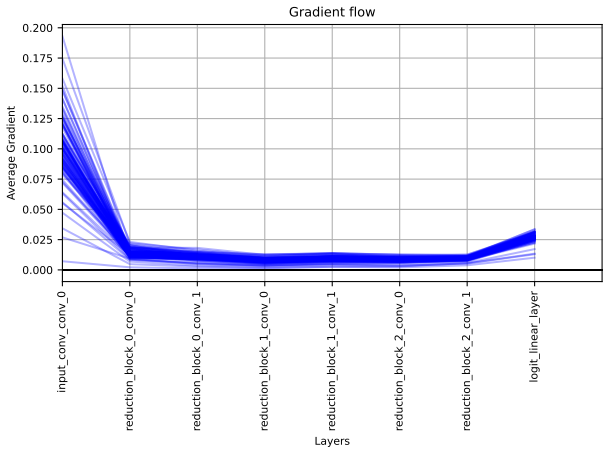


Figure 2. Gradient flow on VGG08

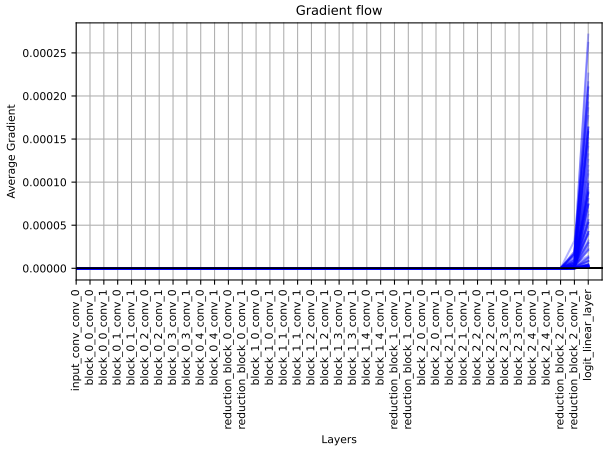


Figure 3. Gradient Flow on VGG38

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:

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

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. **[Initially, when comparing these models, we would expect to see the VGG38 model perform better than the VGG08 model as we are increasing the depth and therefore capacity of the model. However, looking at figure 1 we can see this is not the case. VGG08's performance follows roughly the pattern we would expect; with the models performance improving as training goes on in terms of loss and accuracy, achieving approximately 52% training and 49% validation accuracy by epoch 100. VGG38 on the other hand shows no performance improvement over training at all with approximately 1% test and validation accuracy throughout. Clearly something is wrong with the VGG38 model.**

Indeed, when we look at figures 2 and 3 we see a similar picture emerge. In figure 2, showing the gradient flow in each layer of the VGG08 model, we don't see anything out of the ordinary suggesting this model is healthy. In contrast, in figure 3 we see that the gradient flow for VGG38 is approximately 0.00025 at the output layers and then rapidly diminishes to 0 as we move back through the network towards the input layers. This is the classic symptom of the vanishing gradient problem.

In summary, by analysing figure 2 we can see that the VGG08 model has a normal gradient flow plot indicating a healthy training process, this manifests in the models performance in figure 1 which follows the type of pattern we would expect. In contrast figure 3 shows us that VGG38 suffers from the vanishing gradient problem leading to the poor performance and lack of improvement over training that we found in figure 1.] .

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.

[As we increase a model's depth, we increase it's capacity - it's ability to represent complex functions and patterns. This increased complexity generally makes a model more prone to overfitting; where the model learns the patterns in its training data too well leading to better performance in training accuracy, but worse generalisation and worse validation accuracy (Ying, 2019). This is a common issue in deep neural networks(Bejani & Ghatee, 2021)(Zhang et al., 2018).

However, looking at figure 1 in He et al this is not the pattern that we see. Instead, both training and validation error are higher in the deeper model. The fact that training error is higher in the deeper model suggests that overfitting is not taking place in this case. Instead, the paper proposes that the degradation of performance in the deeper model is caused by increased difficulty in optimising deeper networks.

Taking a shallow model and a deeper counterpart model; the paper argues that the added layers in the deeper model can be constructed as identity mappings from the shallow model, and other layers copied over meaning that the deep model should have training accuracy that is a least no worse than the shallow model. However, this doesn't hold out in the results shown in figure 1 of the paper, we see the training error degrading in the deeper model implying that the model's solvers may struggle to approximate identity mappings using multiple nonlinear layers.] .

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:

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

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

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.

5. Experiment Setup

[ ]

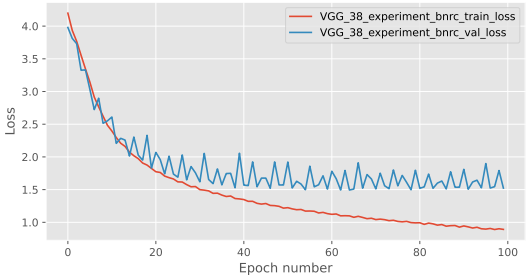
[ ]

[

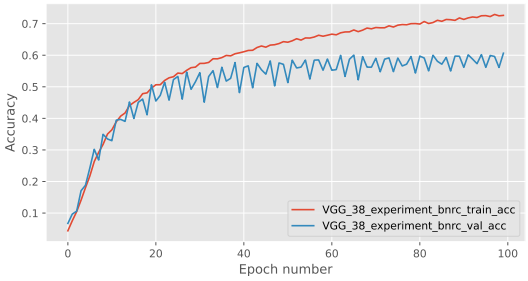
]

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



(a) Cross entropy error per epoch



(b) Classification accuracy per epoch

Figure 4. Training curves for best performing model on CIFAR100 dataset; VGG38 BN+RC with learning rate = 1e-2, in terms of (a) cross-entropy error and (b) classification accuracy

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. 4.

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

[Downsampling layers reduce the spatial dimensions of inputs in terms of width and height in order to increase a models receptive field and improve efficiency. Residual connections aim to alleviate VGP by enabling the addition of a layer's input to its output, therefore incorporating them to downsampling layers may result

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	1.63	53.44	1.89	48.84
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	0.89	72.62	1.52	60.64

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.

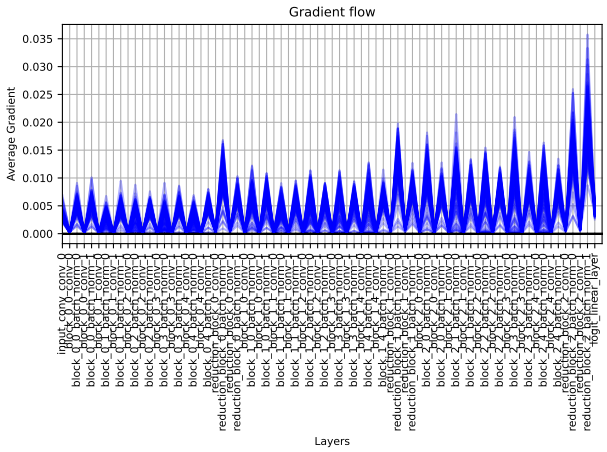


Figure 5. Gradient Flow on VGG38 BN+RC with learning rate = 1e-2

in a mismatch in dimensions between the input and output. In order to incorporate residual connections to downsampling layers we would need to address this dimensional mismatch.

One approach to resolving this would be to use a convolutional layer with a stride equal to the downsampling factor on the input, therefore matching the dimensionality with the downsampling layers output. This method would guarantee exact dimension matching between the inputs and outputs allowing for direct element-wise addition. Additionally having an extra convolutional layer would increase the network capacity of the model. However, this additional layer introduces more parameters to the model and so may impact the training complexity and increase the chances of overfitting.

Alternatively, we could apply pooling, such as average pooling, to the input to match the reduced spatial dimensions and use padding, such as zero padding to align the number of channels with the output of the downsampling layer. This approach would not introduce learnable parameters so it would'nt affect the model complexity and potential overfitting, however, it may lead to increased channel counts and computational demands in later layers.] .

6. Results and Discussion

[Experiments were conducted on the VGG model architecture. The VGG08 model consisted of 7 convolutional layers and 1 fully connected layer and the VGG38 model consisted of 37 convolutional layers and 1 fully connected layer.

Initially, as shown in figure 1, the broken VGG38 model performed very poorly with validation loss of 4.61 and validation accuracy of 0.01%. In comparison; the VGG08, despite having a lower capacity with fewer layers, was shown to perform better achieving validation loss of 1.95 and accuracy of 46.84%. Using the results in figures 1 and 3 it was concluded that the VGG38 model was suffering from the vanishing gradient problem.

To rectify the VGP, batch normalisation (BN) and residual connections (RC) were implemented for the VGG38 model and the model was ran with various combinations of these solutions to investigate their effectiveness. When used independently both BN and RC lead to a dramatic improvement in performance. With a learning rate of 1e-3, BN achieved validation loss of 1.89 and accuracy of 48.84%, while RC achieved validation loss of 1.84 and accuracy of 52.32%. Bringing the VGG38 model performance more in line with what we would expect. The biggest improvement was found when using both BN and RC with a learning rate of 1e-2. In this model we saw the best scores of any model for validation loss and accuracy, of 1.52 and 60.64% respectively.

Clearly the BN and RC techniques were effective in resolving the issue of VGP as both independently lead to improved performance better than the VGG08 baseline model and when combined lead to the best performing model VGG38 BN+RC with learning rate of 1e-2. This model's error and accuracy are plotted in figure 4 and we can see these plots follow the pattern we would expect from a healthy model, with performance improving over training. Additionally, looking at figure 5 we can see that the gradient flow follows a normal pattern indicating that BN and RC have been successful in resolving the vanishing gradient problem.

However, while implementing BN and RC have improved the model performance, it is still far from optimal. In order to improve our performance I would run

a series of experiments on the VGG model with BN and RC implemented, as this was the best performing model for both learning rate values. The first experiment I would run would be to vary the number of layers in the model. So far we have only used 8 and 38 layer models and, while this did a good job of showcasing the VGP, it leaves the question of the optimal number of layers wide open. As proposed in (**Simonyan & Zisserman, 2014**) and commonly used in the literature CITE CITE CITE I would experiment with VGG16 and VGG19 models. I would then use the best performing model from these two and the 8 and 38 layer models in the rest of my experiments.

Then, the difference in performance for the VGG38 BN+RC model with learning rates of 1e-3 and 1e-2 shows the impact that hyperparameters can have on performance. To this end my next experiments would be in hyperparameter tuning to find the best optimal values for learning rate, batch size, number of epochs, weight initialisation, number of filters, and number of blocks per stage. As our model is expensive to run and an exhaustive hyperparameter optimisation would involve testing each combination of different hyperparameter values, I would propose using Gaussian Processes (GP) for hyperparameter tuning of the network through Bayesian Optimisation. Rather than a scatter-gun or exhaustive approach like grid or random search this technique takes into account the relationship between parameters and outputs and would intelligently choose which parameter spaces to explore reducing the number of times we have to run the model while still optimising the parameters.

Next I would run experiments to understand the behaviour of BN and RC. For BN I would experiment with different placements of the BN layers in relation to the activation function as wekk as varyin the activation function type, this would help me gain an appreciation of the effect BN is having throughout the model. For RC I would experiment with how different skip sizes would impact the models performance giving me insight into it's effects.] .

7. Conclusion

[In conclusion, while it may seem intuitive that increasing a models complexity inherently improves it's perfomance the whole picture isn't as simple as this. More complex models are more difficult to optimise and therefore require greater attention to detail in implementation to ensure they perform as well as possible. This was seen in the VGG38 model, where teh vanishing gradient problem meant that this model performed very poorly. Through the implementation of batch normalisation and residual connections this problem was addressed and the resulting model, the VGG38 BN+RC model performed the best out of all the models we tested.

While the implementation of these features did address the vanishing gradient problem, further work is required to improve the performance of the models we used. I propose that future work should aim to build on the findings in this coursework, using techniques such as parameter optimisation and varying model depth further to find the model architecture that performs optimally.

The work in this coursework raises the question as to whether BN and RC are effective in other deep learning architectures, work in this area would help to understand the effect of these techniques more generally rather than in the specific domain of this coursework].

References

Bejani, Mohammad Mahdi and Ghatee, Mehdi. A systematic review on overfitting control in shallow and deep neural networks. *Artificial Intelligence Review*, 54(8): 6391–6438, 2021. ISSN 1573-7462. doi: 10.1007/s10462-021-09975-1. URL https://doi.org/10.1007/s10462-021-09975-1.

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.

Ying, Xue. An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, 1168(2):022022, feb 2019. doi: 10.1088/1742-6596/1168/2/022022. URL https://dx.doi.org/10.1088/1742-6596/1168/2/022022.

Zhang, Chiyuan, Vinyals, Oriol, Munos, Remi, and Bengio, Samy. A study on overfitting in deep reinforcement learning, 2018.