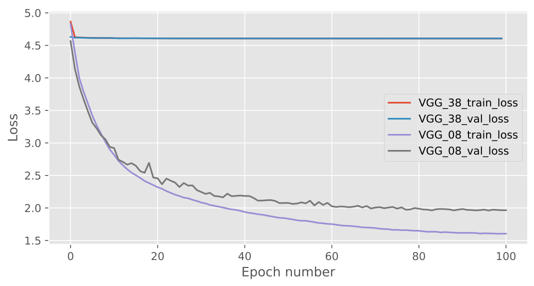
MLP Coursework 2

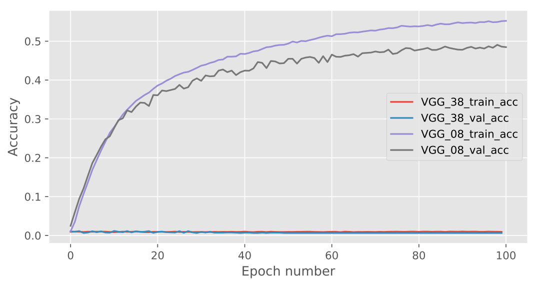
s1813106

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 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 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

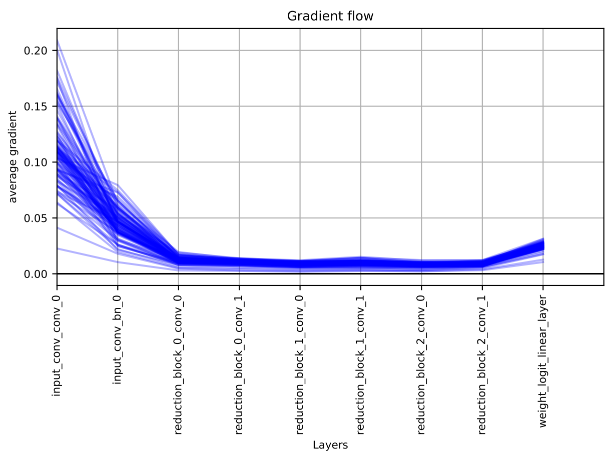


Figure 2. Gradient flow on VGG08

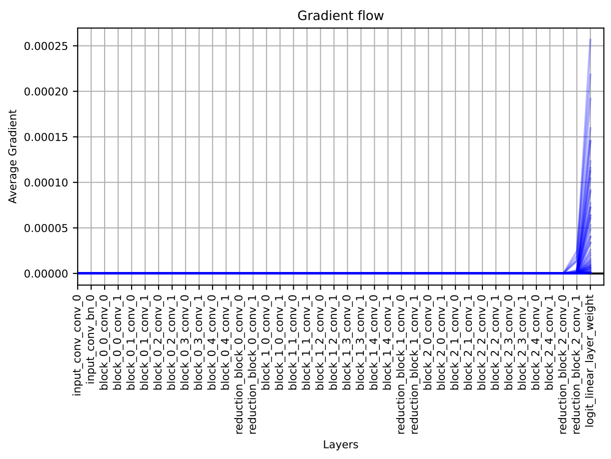


Figure 3. Gradient Flow on VGG38

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:

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

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. [

As we can see from the Figure 3, VGG 38 suffers from the vanishing gradient problem as the first layers have a average gradient value of 0 and only the last 3 layers have non-zero average gradient values with the last layer's average gradient value peaking at 0.00025. The effects of this are visible in Figure 1 and 2 where VGG8's accuracy keeps increasing with the number of epochs while VGG38's accuracy keeps the same as the model is not learning anything from the data. In each gradient update step, we are back-propagating and multiplying together the gradients of the weights. But as we go to the previous layers, we multiply by gradients that are near zero making the update step equal to zero. So, the weights do not get updated and hence the model is not learning(VGG38)] .

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 achieves to reduce the internal covariate shift through normalization of the layers inputs as the training progresses by normalising the inputs of each layer in the neural network. Through this, it manages to do layer whitening by having the inputs have zero mean and unit variance. This normalisation is done by subtracting the mean of the inputs and dividing by their standard deviation. The mean and standard deviation are not extracted from the whole training set but rather from a minibatch making the algorithm a lot faster. Now let's go through how it achieves this.

Let's consider a minibatch B = [$x_{1..m}$] Let $\hat{x}_{1..m}$ be the normalised values and $y_{1..m}$ be their linear transformations. μ_β denotes the mini-batch B mean and σ_β^2 the mini-batch variance:

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

$$\hat{x}_i = \frac{(x_i - \mu_\beta)}{\sqrt{\sigma_\beta^2 + \epsilon}}$$

$$\sigma_\beta^2 = \frac{1}{m} * \sum_{i=1}^{m} (x_i - \mu_\beta)^2$$

$$\mu_\beta = \frac{1}{m} * \sum_{i=1}^{m} x_i$$

Then the batch transformation function is defined as :

$$BN_{\gamma,\beta} : x_{1..m} \mapsto y_{1..m}$$

During Training:

The algorithm of batch normalization used during the training is as below(taken from paper):

```
Input: Network $N$ with trainable parameters $\Theta$;
       subset of activations $\{x^{(k)}\}_{k=1}^{K}$
Output: Batch-normalized network for inference, $N_{\mathrm{BN}}^{\mathrm{inf}}$
 1: $N_{\mathrm{BN}}^{\mathrm{tr}} \leftarrow N$    // Training BN network
 2: for $k = 1 \ldots K$ do
 3:     Add transformation $y^{(k)} = \mathrm{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to
        $N_{\mathrm{BN}}^{\mathrm{tr}}$ (Alg. 1)
 4:     Modify each layer in $N_{\mathrm{BN}}^{\mathrm{tr}}$ with input $x^{(k)}$ to take
        $y^{(k)}$ instead
 5: end for
 6: Train $N_{\mathrm{BN}}^{\mathrm{tr}}$ to optimize the parameters
    $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
 7: $N_{\mathrm{BN}}^{\mathrm{inf}} \leftarrow N_{\mathrm{BN}}^{\mathrm{tr}}$    // Inference BN network with frozen
                          // parameters
 8: for $k = 1 \ldots K$ do
 9:     // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10:     Process multiple training mini-batches $\mathcal{B}$, each of
        size $m$, and average over them:
            $\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$
            $\mathrm{Var}[x] \leftarrow \frac{m}{m-1} \mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$
11:     In $N_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with
        $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$
12: end for
```

Algorithm 2: Training a Batch-Normalized Network

Batch normalization addresses the vanishing gradient problem since by normalizing activations throughout the network, it prevents small changes in layer parameters from amplyfying as the data propagates through a deep network. In addition to that, batch normalization makes training more resilient to the parameter scale meaning that back-propagation through a layer is unaffected by the scale of its parameters(which is a problem when vanishing gradient occurs).

] .

4.2. Residual connections

[

Despite Batch normalisation being a very good solution for the vanishing gradient problem, it is not a complete

one and it comes with a drawback. A common problem that was found when using batch normalization is that training accuracy often fails to improve and can increase as number of layers increases. But if we stack more layers and make the model more complex, it may fail to learn simpler models like the identity function. Also, as we stack more and more layers, we will see that accuracy will saturate at some point and then degrade(as the model may be overfitting). To cope with that, (He et al., 2016) introduced the residual connections. In the paper, he suggested that a deeper model constructed from a shallower one by adding new layers and mapping them using the identity function should have a training error less than or equal to the shallower model. This mapping was defined as H(X) (where X is the input and H the mapping) and this was to be learned by the non-linear(like convolution) layers. Then these stacked non-linear layers have to learn the mapping :

F(X) = H(X) - X

Therefore the mapping H(X) is recast into F(X) + X. As long as F(X) = 0, an identity mapping is formed:

H(X) = X

F(X) is called the residual.

Residual connections are skip connections which allow the gradients to flow through directly during training without having to go through the non-linear layers(As we saw in VGG38 it is the non-linear convolution layers having a gradient near zero that cause the vanishing gradient problem). This addresses the vanishing gradient problem and complements batch normalization in dealing with this problem.] .

5. Experiment Setup

[ ]

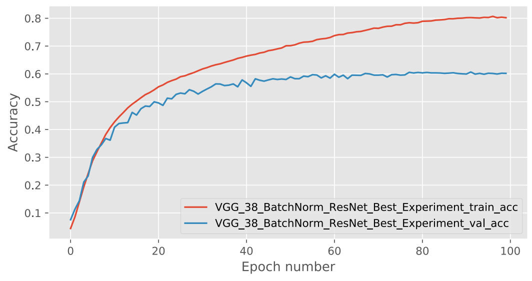
[ ]

[ ]

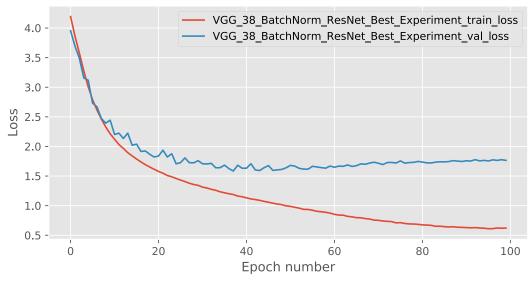
We conduct our experiment on the CIFAR-100 dataset (Krizhevsky et al., 2009), which consists of 60,000 32x32 colour images from 100 different classes. The number of samples per class is balanced, and the samples are split into training, validation, and test set while maintaining balanced class proportions. 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



(a) Loss per epoch



(b) Accuracy per epoch

Figure 4. Training curves for VGG38 with Batch Normalisation and Residual Connections, and a learning rate of 10^{-2}

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

[

If we look at the results from Table 1, we can clearly see that VGG38 with no batch normalisation and residual connections suffers from the vanishing gradient problem as the both training and validation accuracy are 0.01. As we add batch normalization and residual connections to the model, its accuracy dramatically increases to around 49 and 60 respectively. Please note that I have replaced the existing values in the table as I was getting different results when running my models. Nevertheless, the setup was the same with a batch size of 100 for 100 epochs(and no weight decay). We can see that now VGG38 BN and VGG38 BN + RC have a higher validation accuracy than VGG08(meaning that the two methods have been successful in dealing with the vanishing gradient problem). However, we can notice that the generalization gap for VGG38 BN + RC is much

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	1.25	63.11	1.25	49.73
VGG38 RC	1e-3	336 K	1.33	61.52	1.84	52.32
VGG38 BN + RC	1e-3	339 K	0.63	80.04	1.71	59.24
VGG38 BN	1e-2	339 K	1.59	55.08	1.96	48.68
VGG38 BN + RC	1e-2	339 K	0.621	80.21	1.76	60.20

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.

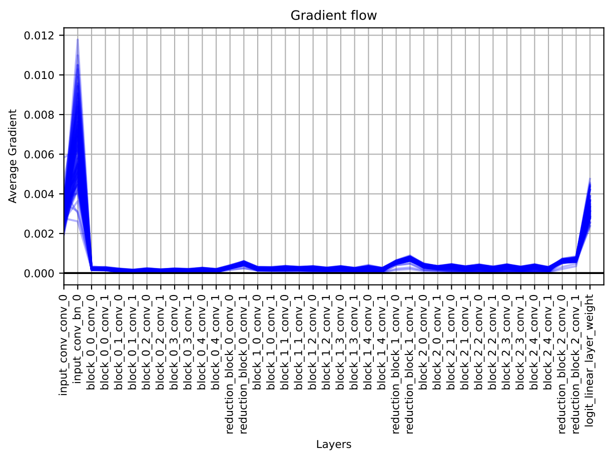


Figure 5. Gradient Flow on VGG38 with Batch Normalisation and Residual Connections, and a learning rate of 10^{-2}

higher than the one for VGG38 BN. We can see also see from Figure 4 that the validation accuracy keeps increasing until around 60th epoch and then mostly stays the same meaning that the network trained faster with both methods implemented. In this case, 60 epochs would have been enough to get satisfying results(lower generalization gap and high accuracy).

When comparing VGG38 BN for 2 different learning rates(10^{-2} and 10^{-3}) we can see that they have comparable validation accuracy and comparable test accuracy. However, it seems that the BN with higher learning rate seems to be having a better/lower generalization gap. This is most likely connected to the model training faster on a lower learning rate. If we look at VGG38 with only RC implemented, we can notice that the accuracy is better than both VGG38 BN with 2 different learning rates. However, it achieves higher generalization gap than both of them.

Now we compare VGG38 BN+RC for learning rate of 10^{-2} and 10^{-3}. We can see that the model that used a learning rate of 10^{-2} achieved slightly higher validation and test set accuracy while having a slightly lower generalization gap. As we discussed for batch normalization, lower learning rates may cause the model to train faster. But we can see that both of these models have significantly higher generalization gaps than the other models. This may attributed to having 2 methods (BN and RC) which both have a high generalization gap and the results add up(meaning that training the models on fewer epochs would be beneficial as the models seem to train faster than the other models(batch normalization or residual connections).

I also tested the models with different weight decays(I have the results in the submission as well and I have a text file to state the setup that was used). I will discuss results for VGG38 BN + RC when using different weight decays. When using a weight decay of 10^{-4} the model seemed to have a lower generalization gap and have similar validation and test set accuracy(About 61). However, this model was trained on 200 epochs meaning that a higher weight decay would train the model slower than a lower weight decay. These conclusions are also confirmed by my other setup where the model was trained on a weight decay of 10^{-6}. In this case after 100 epochs, the model achieved an accuracy of 60.42 but the generalization gap was much higher.

Additional Experiments that could be run:

Some addition experiments that could have been run would be to try batch normalisation and residual connections architectures on simpler and much complex models than VGG38 in order to come up with more concise conclusions. Also, we could have run VGG38 with only residual connections trying different learning rates and weight decays to see how the model would react to these different parameters. Also, we could have played with learning rate and weight decay more to come up with more definite conclusions on how these hyperparameters affect training and validation accuracy. Dropout does not need to be considered here as (Ioffe & Szegedy, 2015) stated that dropout is not needed when we have batch normalisation. But, we can always test Maxout (Goodfellow et al., 2013) to analyze how it combines with batch normalisation or residual connections.

] .

7. Conclusion

[It can be concluded from different experiments that the implementation of batch normalisation and residual connections helps solve the vanishing gradient problem in a 38 layer network. We have seen that even used seperately, both of the aforementioned methods help solve the vanishing gradient problem. But they seem to complement each other, given that when used together the accuracy increased by a bit more 10%. Furthermore, (Ioffe & Szegedy, 2015) showed that batch normalization works better on computer vision tasks while (He et al., 2016) suggested that residual connections seems to be working better on deeper networks.

There are several future works that can be conducted. One direction it could go is trying to apply Group Normalization(Wu & He, 2018) on architectures that are already using batch normalisation and see if it achieves better results(in ImageNet, Group normalization has 10.6% lower error than its Batch normalization counterpart when using a batch size of 2). In addition to that, batch normalization could be conducted on architectures of different widths instead of depth to try and understand how it would perform. Also, we could see how scaling both in depth and width can affect the performance of a model using batch normalization.] .

References

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.

Goodfellow, Ian, Warde-Farley, David, Mirza, Mehdi, Courville, Aaron, and Bengio, Yoshua. Maxout networks. In *International conference on machine learning*, pp. 1319–1327. PMLR, 2013.

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.

Wu, Yuxin and He, Kaiming. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.