AI6103 - Individual Assignment Report on MobileNet Tuning

Chen Yuhao
G2406208G
CCDS, Nanyang Technological University
50 Nanyang Avenue, 639798 Singapore
ch0009ao@e.ntu.edu.sg

Abstract

This report investigates the effects of key hyperparameters on the training of MobileNet using the CIFAR-100 dataset. Four main tasks are explored: the selection of an optimal learning rate, the application of a learning rate scheduler (cosine annealing), the use of weight decay as a regularization method, and the comparison between ReLU and Sigmoid activation functions. Experimental results and analyses provide insights into improving neural network stability, convergence, and generalization performance.

Introduction

Description on MobileNet

In the field of Computer Vision, CNNs (Convolutional Neural Networks) are powerful models that achieved outstanding performance on different downstream tasks. On the one hand, convolution kernels can capture local features in the image, which extract effective features of the image. On the other hand, convolution neural networks benefit from the weight-sharing mechanism, which significantly reduces the amount of parameters than an MLP (Multi-Layer-Perceptron), lowering the difficulty of training and the risk of overfitting.

Back in 2014, researchers at Oxford came up with ideas to increase the CNN's depth to enhance the model performance and designed the first prototype of deep CNN, as known as the VGG net (Simonyan and Zisserman 2015). The design of VGG is quite simple, as its architecture consists of multiple convolutional layers, each with a convolution kernel size of 3x3 and a stride of 1, and a pooling layer with a pooling size of 2x2 and a stride of 2. This design makes it theoretically possible to expand very deep CNNs. However, the disadvantage of VGG is that it is computationally intensive and has a very large number of parameters, so it also requires a lot of computing resources and memory.

Based on this problem, how to design a lighter CNN has become an focus in the field of deep learning. One of the typical designs of lighter CNN is MobileNet. It replaces the normal convolution in VGG with the **Depthwise Separable Convolution** (Howard et al. 2017). More specifically,

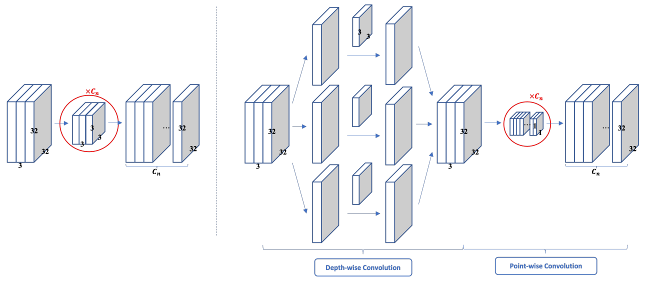


Figure 1: Design of Depthwise Separable Convolution adapted from the original paper. Left: Standard convolutional layer. Right: Depthwise Separable convolutions with Depth-wise and Point-wise convolution layer.

Depthwise Separable Convolution splits the convolution operation into two steps:

- **Depthwise Convolution**: Instead of computing across all input channels and all output channels as in normal convolution, a convolution kernel is applied independently to each input channel. This means that each convolution kernel is convolved with only one input channel, thus greatly reducing the amount of computation.
- **Pointwise Convolution**: This is a 1x1 convolution that combines the output of the depthwise convolution from the separate feature maps of each input channel to generate the final output channel. This part is used to integrate the output of the depthwise convolution between channels.

Through this decomposition, MobileNet significantly reduces the amount of computation and the number of parameters while maintaining good performance. Compared with normal convolution, depth-separable convolution requires less computing resources and is especially suitable for embedded devices and mobile devices. That is also the reason why it is called MobileNet.

Motivation & Report Organization

The main purpose of this report is to learn the effects of different optimization and regularization settings on training deep neural networks by tuning the MobileNet model.

In general, the report focuses and is organized in the following content.

- **Learning the effect of Learning Rate**: Compare how different learning rate will affect model performance by training few epochs (e.g. 15 epochs) using different initial learning rate settings, also to find a best setting for following study.
- **Learning the effect of Learning Rate Schedule**: Use the best initial learning rate obtained from the previous step, analyze the effect of Learning Rate Schedule (mainly the cosine annealing scheduler that slowly reduces the learning rate to 0 as training epoch grows) on neural network training.
- **Learning the effect of Weight Decay**: Use weight decay to regularize neural network training and compare the effects of different regularization strengths (set by the hyperparameter λ).
- **Learning the effect of Activation Function**: The activation functions of layers [4, 5, 6, 7, 8, 9, 10] in the MobileNet model are replaced from ReLU to Sigmoid, in order to compare the effects of different activation functions on model training and overall performance.

Basic Setting & Experiment Environment

Dataset This report uses the CIFAR-100 dataset to train the MobileNet on the task of image classification. The CIFAR-100 dataset contains classified natural images (divided into 100 subcategories), and the resolution of each image is **32 × 32**. After partitioning, **80% (4000 images)** of the original training dataset (5000 images) is used as the training set, **20% (1000 images)** is used as the validation set, and another 1000 images are used as the test set. In all experiments, DataLoader is used to load images and labels with batch size set to **128**. Additionally, **random cropping** and **random horizontal flip** are used on training sets to augment data. For optimization, all networks training are using **Stochastic Gradient Descent optimizer** with momentum set to **0.9**.

Experiment Environment All experiments were run on a PC with an i7-14700KF CPU and an NVIDIA GeForce RTX 4070 Ti Super GPU (16GiB memory). Unless otherwise stated, model training, validation, and testing are all performed on the GPU to speed up the overall calculation. The CUDA version is 12.6, Python version is 3.9 and Pytorch version is 2.6.0. Additionally, to supervise the training process, Weight & Bias[1] is used to log and record the experiment settings and results.

Learning Rate

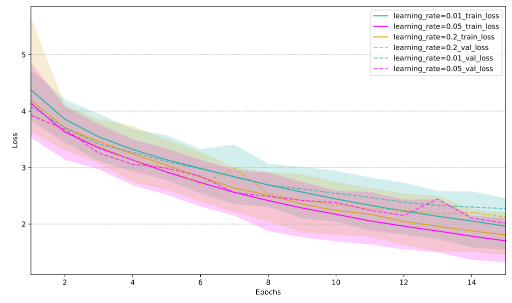
Brief on Learning Rate

The learning rate is a hyperparameter that controls the size of the steps a neural network takes during training in response to the gradients calculated from the loss function. It determines how quickly or slowly a model learns and converges to an optimal solution. Theoretically, a learning rate that is too high can cause the model to overshoot the minimum, while one that is too low can lead to slow convergence.

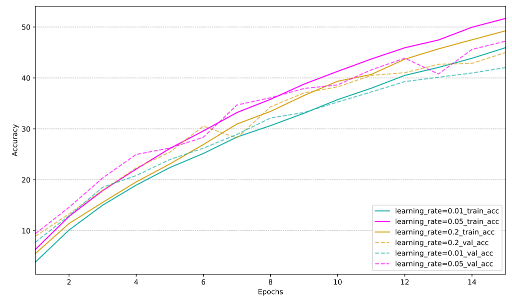
To validate this, this report runs 3 experiments on how learning rate effects networks training by setting learning rate to 0.01, 0.05 and 0.2 respectively. The other parameters and settings are kept consistently at the same time. Additionally, only 15 epochs are trained because the main idea is to identify the best best value for learning rate initialization.

Experiment Result

After the comparison experiments on 3 different learning rate settings, Figure **??** shows the loss curves of training and validation processes. The loss curves demonstrate that, whether in the training or verification process, with the increase of training epochs, when the learning rate is set to **0.05**, the network training and verification losses decrease the most, and the accuracy improves most significantly.



(a) Loss curves



(b) Accuracy curves

Figure 2: Loss Curves and Accuracy Curves on Different Learning Rate

Furthermore, the models with different learning rate settings were tested on the training set. The test results show that the model with the learning rate set to **0.05** still achieved the highest accuracy of **48.25%**, while the accuracy rates of the models with the learning rate set to 0.01 and 0.2 were **42.32%** and **45.33%** respectively.

Conclusion & Discussion

In conclusion, among the three tested initial learning rates (0.01, 0.05, and 0.2), the learning rate of 0.05 achieved the

[1]Weight & Bias is an advanced platform where AI developer can log and monitor their model training process with concise data panels as well as many visualization tools, https://wandb.ai/

best overall performance, yielding the lowest loss and highest accuracy on both training and validation sets. Specifically, the model with a learning rate of 0.05 achieved a validation accuracy of 46.97%, outperforming the models trained with learning rates of 0.01 (41.02%) and 0.2 (44.17%).

The experimental results suggest that setting the learning rate too low (e.g., 0.01) can lead to slow convergence, hindering the model from effectively escaping local minima within the limited epochs. Conversely, a high learning rate (e.g., 0.2) may cause unstable training, as indicated by relatively poorer performance compared to 0.05. The learning rate of 0.05 strikes a balance between convergence stability and training speed, enabling the network to efficiently minimize loss and improve accuracy. This observation aligns with theoretical expectations that an appropriate learning rate should neither be excessively high nor too low, facilitating stable yet sufficiently rapid convergence.

For future experiments, the learning rate of 0.05 will be adopted as the baseline to further investigate the effects of learning rate scheduling and weight decay, aiming for enhanced generalization and improved accuracy on the MobileNet architecture.

Learning Rate Schedule

Brief on Learning Rate Schedule

A learning rate schedule refers to a strategy that dynamically adjusts the learning rate during the training of neural networks. Such strategies help improve training efficiency and facilitate better convergence by systematically decreasing the learning rate as training progresses. Commonly used learning rate schedules include step decay, exponential decay, and cosine annealing. All these methods share the objective of enhancing the model's training stability and generalization capabilities.

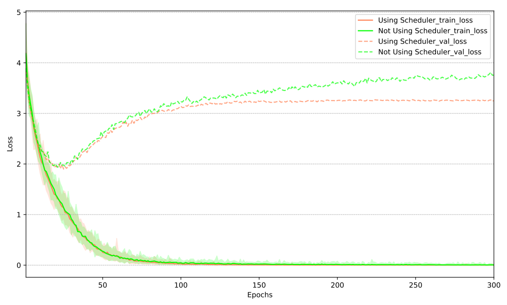
In this section, the cosine annealing scheduler is utilized to explore the influence of learning rate scheduling on the training of MobileNet. The cosine annealing approach gradually reduces the learning rate following a smooth cosine curve, allowing the network to make significant parameter adjustments initially and perform fine-tuning towards the end of the training process (Loshchilov and Hutter 2017).

For this experiment, the number of training epochs is increased to 300, providing the model sufficient time to fully exploit the potential advantages offered by the cosine annealing learning rate schedule.

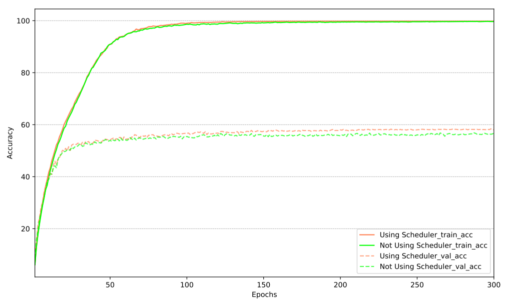
Experiment Result

Similarly, use the training and validation loss curves to compare the impact of using and not using the learning rate scheduler on model training. From the loss curves in Figure 3, it can be seen that during the 300 epochs of training, the training loss of the model is constantly decreasing, with a faster decline in the first 50 epochs and a slower decline thereafter. However, the validation loss of the model on the validation set increased significantly after 50 epochs, indicating overfitting.

At the same time, when comparing the results of using and not using the learning rate scheduler, it can be seen that the effects of the two on the training loss are not much different, as reflected in the fact that the training loss curves are almost identical, but in the validation loss curves, the effect of using the scheduler on the increase in the validation loss in the later stage is not as serious as not using the scheduler.



(a) Loss curves



(b) Accuracy curves

Figure 3: Loss Curves and Accuracy Curves on using and not using Scheduler

Conclusion & Discussion

Based on the experimental results, the cosine annealing learning rate scheduler effectively mitigates the problem of overfitting observed in the MobileNet training process. Although both models with and without the scheduler show similar patterns of decreasing training loss over 300 epochs, the validation loss curves highlight the clear advantage of incorporating the scheduler. Specifically, the model utilizing the cosine annealing scheduler exhibits a noticeably reduced increase in validation loss in later epochs compared to the model without the scheduler. This indicates improved generalization performance, likely due to the gradual reduction in the learning rate of the scheduler, allowing more precise parameter updates as training progresses.

The underlying reason for this improved performance can be attributed to the capacity of the cosine annealing schedule to adjust the learning rate adaptively, ensuring larger parameter updates at the early stage when the network is far from optimal and finer adjustments in later stages when the network approaches convergence. By dynamically controlling the learning rate, the cosine annealing method effectively balances exploration and exploitation during training, leading to better overall optimization outcomes.

Thus, it can be concluded that adopting a cosine annealing learning rate scheduler is beneficial for training deep neural networks like MobileNet, significantly improving their stability and generalization capabilities.

Weight Decay

Brief on Weight Decay

The previous experiments also showed two potential conclusions. First, a smaller epoch may not give the network enough time or rounds to fully learn the patterns and characteristics of the data, resulting in under-fitting of the model. Second, a larger epoch may make the model's decisions rely too much on the training data, resulting in over-fitting of the model, that is, an increase in validation loss in the later stages.

To address this problem, weight decay is a good method which allows the model to be trained for more rounds and can reduce the risk of over-fitting of the model caused by increasing training epochs. In short, weight decay is a regularization technique used to prevent overfitting by penalizing large weights in the model. It adds a term to the loss function that discourages the model from assigning excessively high values to the weights. Specifically, the modified loss function $L'(w)$ is defined as follows:

$$L'(w) = L(w) + \frac{1}{2}\lambda|w|^2, \tag{1}$$

where $L'(w)$ represents the original loss function without regularization, w is the vector containing model parameters (weights), and λ is the weight decay coefficient controlling the strength of regularization.

During optimization with gradient descent, the update rule for the parameters at iteration t becomes:

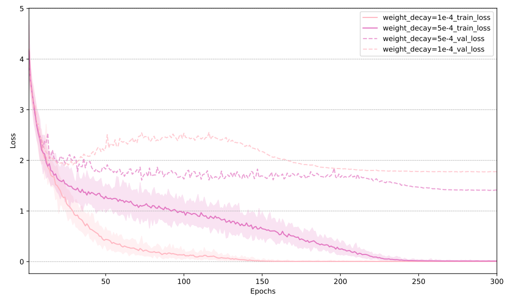
$$w_{t+1} = w_t - \eta\frac{\partial L(w_t)}{\partial w_t} - \eta\lambda w_t, \tag{2}$$

where η represents the learning rate. From the above equation, the effect of weight decay is equivalent to reducing each weight by an amount proportional to its current magnitude. Thus, weight decay prevents the model parameters from growing excessively large, thereby improving the generalization ability of the neural network and achieving better performance on unseen data (Zhang et al. 2018).

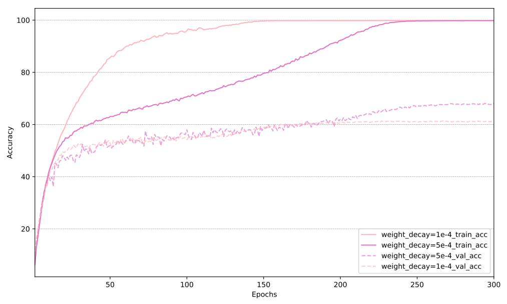
Experiment Result

The experimental results after using weight decay are in sharp contrast with the previous results are shown in Figure 4. Similarly, the model training loss shows a continuous downward trend, but in terms of validation loss, compared with the model without any weight decay, the upward phenomenon of validation loss curves in the later stage of training disappears, and instead, validation loss can continue to decline in a relatively stable trend.

Comparing different weight decay settings, the experiment mainly controls the strength of regularization by setting the weight decay coefficient λ. It can be found that when



(a) Loss curves



(b) Accuracy curves

Figure 4: Loss Curves and Accuracy Curves on different Weight Decay settings

the regularization strength is large (λ is set to 0.05), although the model slows down the decline of training loss in the middle stage of training compared with the small regularization strength (λ is set to 0.01), it can still approach the loss reduction degree of small regularization in the later stage. And the larger regularization strength can ensure the stable decline of validation loss and better control the problem of overfitting.

Finally, comparing the performance of the two weight decay setting models on the test set, it can be seen that the model trained with a larger regularization strength ($\lambda = 0.05$) achieved an accuracy of **68.05%**, which is significantly better than the model trained with a smaller regularization strength ($\lambda = 0.01$) with accuracy of **61.97%**.

Conclusion & Discussion

The experimental results clearly demonstrate the effectiveness of weight decay in alleviating the overfitting problem observed in deep neural networks such as MobileNet. Specifically, weight decay significantly stabilizes the validation loss curves, allowing continuous improvement in model performance throughout the entire training period of 300 epochs.

Comparing different weight decay strengths, a larger regularization coefficient ($\lambda = 0.05$) proves to be more effective in controlling overfitting, despite slightly slower training convergence during the intermediate training stages. Importantly, this larger regularization strength ultimately results in superior generalization performance, as evidenced by achieving a higher accuracy (**68.41%**) on the test set, compared to the smaller regularization strength ($\lambda = 0.01$, accuracy **62.33%**).

This improvement can be explained by the underlying principle of weight decay: penalizing excessively large parameters prevents the model from overly fitting specific features within the training data, thus enhancing its generalization ability. Additionally, the moderate slowdown in the intermediate stages caused by stronger regularization is beneficial, allowing the network to explore a more robust optimization path rather than quickly converging to potentially suboptimal solutions.

In conclusion, applying an appropriate weight decay regularization effectively balances training convergence and model generalization. Therefore, choosing a suitable regularization strength is critical for achieving optimal performance and preventing overfitting when training deep neural networks.

Activation Function

Brief on Activation Function

Activation functions introduce non-linearity into neural networks, enabling them to model complex and nonlinear relationships between inputs and outputs. Common activation functions include ReLU, sigmoid, and Tanh. The selection of activation functions is critical as it affects both the training dynamics and the performance of neural networks.

The Rectified Linear Unit (ReLU) activation function has become one of the most widely used activation functions due to its simplicity, computational efficiency, and ability to mitigate issues such as gradient vanishing. Mathematically, ReLU is defined as:

$$\mathrm{ReLU}(x) = \max(0, x). \tag{3}$$

ReLU effectively alleviates gradient vanishing by maintaining a constant gradient (equal to 1) for positive input values, which significantly accelerates training convergence.

In contrast, the sigmoid activation function outputs values between 0 and 1, providing a smooth, differentiable nonlinear mapping. It is typically used in output layers for binary classification tasks or in situations requiring probability interpretations. The sigmoid function is mathematically expressed as follows:

$$\mathrm{Sigmoid}(x) = \frac{1}{1 + e^{-x}}. \tag{4}$$

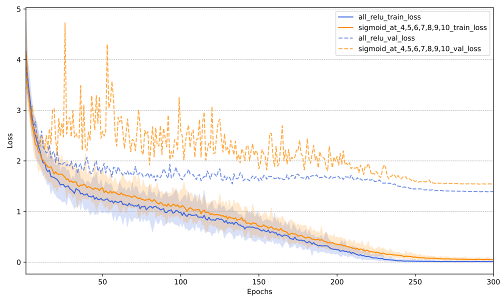
Despite its smoothness and interpretability, sigmoid often faces issues such as gradient saturation or vanishing, particularly when inputs are extremely large or small. This can negatively affect the training efficiency of deep neural networks.

In this section, experiments are conducted to investigate the impact of replacing **ReLU** with **sigmoid** activation in blocks **4–10** of the MobileNet architecture. By comparing their performance, this section aims to analyze how different activation functions influence model convergence behavior and overall performance.

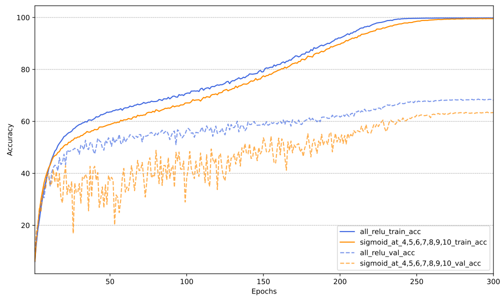
Experiment Result

The experimental results shows that model with ReLU for all activation functions performs better than replacing part of them to Sigmoid, based on the result of test accuracy of **68.39%** for the former and **64.37%** for the latter. Additionally, comparison on the ReLU and Sigmoid activation functions in blocks 4–10 of MobileNet is illustrated by the training and validation loss curves in Figure 5.

From the training loss curves (Figure 5(a)), it can be observed that both activation functions lead to rapid convergence and similar final training losses, suggesting that both ReLU and Sigmoid activations are capable of fitting the training data effectively within the given epochs.



(a) Loss curves



(b) Accuracy curves

Figure 5: Loss Curves and Accuracy Curves on different Activation Function settings

However, the validation loss curves (Figure 5(b)) reveal significant differences in training stability and generalization performance. While the validation loss for the ReLU-based network decreases steadily and consistently, the model employing the Sigmoid activation exhibits substantial instability and fluctuations, particularly evident in the intermediate training stages (approximately epochs 50–200). This instability suggests gradient saturation and vanishing gradient issues associated with the Sigmoid activation, adversely affecting the network's optimization trajectory and its ability to generalize effectively.

These results indicate that although both activation functions achieve comparable convergence on the training set, the ReLU activation significantly outperforms Sigmoid in terms of training stability and generalization capability, making it the preferred choice for intermediate layers in deep neural networks like MobileNet.

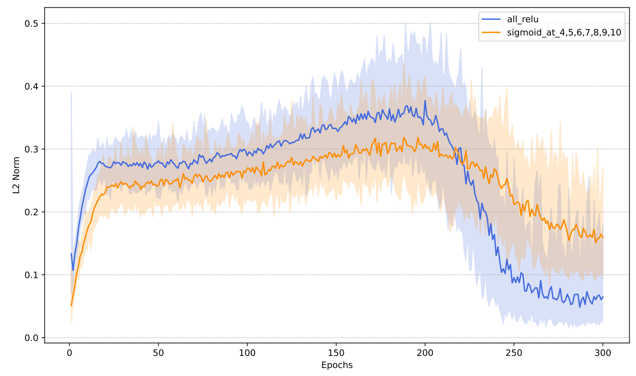


Figure 6: Gradient 2-norm of Layer 8 Convolution

Conclusion & Discussion

Based on the experimental results, there is clear evidence that the choice of activation function significantly impacts the training stability and generalization performance of MobileNet. Although both ReLU and Sigmoid activation functions allow the model to achieve similar rapid convergence on the training set, their effects differ dramatically in terms of validation performance and training stability. Specifically, ReLU provides a stable and consistently decreasing validation loss, whereas the Sigmoid activation function leads to pronounced instability and significant fluctuations in validation loss, particularly during the intermediate training stages.

Further insights can be obtained from the 2-norm evolution curves of weights gradient vector at layer `model.layers[8].conv1.weight` over the 300 epochs, as illustrated in Figure 6. The curves for both activation function settings share similar overall trends: the gradient norm initially increases, then maintains a relatively high magnitude during the first 200 epochs, followed by a noticeable decline beyond epoch 200. However, their behaviors differ after epoch 200. While the network partially adopting Sigmoid activation shows a gentle decrease with only moderate reduction in gradient norm, the network that uses all ReLU activation experiences a more significant and faster decline in gradient magnitude to its convergence, showing a more effective optimization by SGD algorithm.

Despite these differences, neither activation function setting demonstrates obvious gradient explosion or gradient vanishing issues. The absence of such issues can be attributed to the position of layer 8 within the network architecture. Although positioned deeper in the forward propagation direction, layer 8 occupies an earlier stage during backpropagation. According to the chain rule in gradient calculation, layers at earlier stages in backpropagation typically encounter lower risks of gradient instability such as vanishing or explosion.

Nonetheless, based on the gradient norm trends observed, it can be concluded that using ReLU activation functions throughout all intermediate layers enhances the gradient stability during training compared to incorporating Sigmoid activations. This increased stability is crucial for ensuring the robustness and overall generalization capability of deep neural networks like MobileNet.

Implications

Through the above experiments, this research has found that, under the premise that the model structure has not changed, by adjusting the hyperparameters in the model training, such as Learning Rate, Learning Rate Scheduler, Weight Decay, etc., the performance of the model can be greatly improved. This also reveals that model parameter adjustment and hyperparameter selection are also important tasks in deep learning. On this basis, further investigation shows that the use of some parameter selection techniques, such as **K-fold cross validation**, **GridSearch**, etc, can benefit better selecting the appropriate hyperparameter combination to improve network performance.

References

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861.

Loshchilov, I.; and Hutter, F. 2017. SGDR: Stochastic Gradient Descent with Warm Restarts. arXiv:1608.03983.

Simonyan, K.; and Zisserman, A. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556.

Zhang, G.; Wang, C.; Xu, B.; and Grosse, R. 2018. Three Mechanisms of Weight Decay Regularization. arXiv:1810.12281.