Document version: 1.1 (2015-11-15)

Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	Carlisle	Student ID:	19136347
Other name(s):	Tristan Carlisle		
Unit name:	Stochastic and Connectionist Approaches	Unit ID:	COMP6012
Lecturer / unit coordinator:	Senjian An	Tutor:	
Date of submission:		Which assignment?	(Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: TristanCarlisle Date of signature: 26/05/2023

(By submitting this form, you indicate that you agree with all the above text.)

Task 1

What factors affect the training of residual neural networks.

Introduction

The Myrtle.ai tutorials outline the process of optimising Ben Johnsons 18-layer Residual network which reached 94% accuracy in 341s. This was achieved by; Firstly, utilising mixed-precision training which involves storing weights and gradients as FP16 during training iterations to reduce memory usage and training time, then converting them back to FP32 to ensure minimal loss in accuracy. Secondly, choosing a smaller network with sufficient accuracy. Finally, increasing then decreasing learning rates during epochs to increase stochastic gradient descent speed. The tutorials build on this model to further increase training speed. Although many factor were established that influence training accuracy and time not all can be mentioned and thus, the key influencers will be discussed.

Computational resources

Availability and utilization of computational resources are key bottlenecks in the training of deep neural networks. A single forward and backward pass through the 18-layer residual network on a 32x32 CIFAR image requires approximately 2.8x10^9 FLOPs, which is then multiplied by the number of images and epochs. Efficient utilization of CPU and GPU processing capabilities is essential for timely training of deep CNNs. CPUs are general processors that are not optimized for deep learning computations, having fewer cores compared to GPUs, which limits their ability to process in parallel. Training deep learning neural networks on CPUs can be extremely slow and inefficient due to the large quantity of computations involved. On the other hand, GPUs are highly parallel processors that can handle multiple computations simultaneously. However, GPUs also have limitations, such as available memory and compute resources. Therefore, without prioritizing computational optimization, computational resources can become the primary limitation in training. This is immediately made evident when assessing the methodologies utilised in myrtle.ai tutorials. For example:

- They found that image pre-processing which although is handled by the CPU requires PyTorch data loaders to launch fresh processes each iteration. Completing the image pre-processing before training removes repeat work and reduced the training time by 15s.
- Random number generators to select data rather than in the augmentations themselves. Rather than making millions of individual calls to random number generators, combining these into a small number of bulk calls at the start of each epoch can shave off 7s. Furthermore, the launching of even one process to perform the data augmentation outweighs the benefit thus, added the augmentation to the main thread leads to a further 4s reduction.

Architecture

The Myrtle.ai tutorials break down the building of a Residual network. The importance of understanding the networks architecture is clearly established. While it is true that increasing network depth is a viable strategy for achieving higher accuracy, Myrtle.ai managed to batch the highest performing network with half the number of layers (K. He, Zhang, Ren, & Sun, 2016). Therefore, it is essential to explore not only the impact of network depth but also the quality of the layers within the net. For example, utilising a 3x3 kernel with a stride of 1 paired with a max pooling rather than a 1x1 kernel with a stride of 2 resulted in similar levels of down sampling with a significantly larger receptive field . Secondly, fragmenting the residual network and removing the skip connection highlights the benefit they bring for improving gradient flow and network training speed and accuracy (K. He et al., 2016). Order, node choice and branching all contribute significantly to the network and establish themselves as key components for convolution neural network construction and training. Table 1. Is an example of implementations of various architectures at the same depth and the impact they can have on response metrics.

Table 1 Myrtle. Ai's various architectures of a shallow CNN

Network	Test acc	Train time
Original backbone	55.9%	36s
No repeat BN-ReLU	56.0%	32s
3×3 convolutions	85.6%	36s
Max pool downsample	89.7%	43s
Global max pool	90.7%	47s
Better BN scale init	91.1%	47s

Hyperparameters

Hyperparameters play an essential role in the training and success of deep learning models. Hyperparameter selection can significantly impact a model's performance (Buda, Maki, & Mazurowski, 2018). Both speed of training and accuracy are highly dependent on these hyperparameters, however, selection is a complex procedure. Some key hyperparameters include:

- Learning rate: Dictates the how much the weights are updated.
    - Smaller learning rates are slower but can handle larger gradients. May get stuck in local minimums.
    - Larger learning rates are train faster as the require less updates to reach minimums.
- Batch size: number of inputs per a split of the training dataset. Allows for faster training and loss function is more stable with less noise due to fewer updates.
- Weight decay: the level of penalisation for complexity. L2 Norm.

There are many more such as momentum, epochs , kernel sizes, Activation functions etc.

Furthermore, Hyper parameters such as but not limited to learning rate, batch size, momentum and weight decay are all deeply intertwined, and their relationships require exploring. The Myrtle.ai tutorials elucidated some of these interactions and mitigation strategies.

For example.

> In tutorial 2 the author wanted to assess the effect of increasing batch size, since larger batch size should result in more efficient computation. This is due to updates only occurring at the end of the batch. This causes a delay in which if the learning rate is low is considered a higher order effect meaning it has no impact on anything first order. The benchmark CNN is based on utilising high learning rates in the form a learning rate schedule. If maximum learning parameters are being used and thus higher speeds, we would expect lower order effects to balance first order ones resulting in increases in first order steps being offset by curvature penalties. Thus, increasing update delays should result in the same curvature penalties as learning rates. However, when Batch size was increased accuracy was the same. This was due to having a combination of a large learning rate, small batch size and a large dataset. Causing the model to forget.

Therefore, hyperparameters are a key component to the construction and training of any Deep learning network.

Regularisation

Regularization techniques are essential in training deep learning networks to improve generalization and prevent overfitting. Due to the inherent expressiveness of convolutional neural networks, they tend to memorize training data as opposed to robust patterns. This results in poor generalisation or the inability to perform well on unseen data sets as the network has become to specialised on the training dataset. With increasing depth and complexity the risk of overfitting increases (Eigen, Rolfe, Fergus, & LeCun, 2013). Combative strategies for this are called regularisation. There are many different regularisation methods that can be utilised in CNNs such as:

- Batch Norms: Normalizes the activations of each mini batch, reducing the internal covariate shift and improving the stability and speed of training. Batch Norms also has inherent regularisation effect via adding small amounts of noise to the activations of each minibatch.
- Weight decay: an L2 norm which penalises complexity via placing weights in the loss function to ensure they are minimised. Shrinking them towards 0.
- Data augmentations methods: random cropping which takes a random size square from a random image and scales it to the same size as other images. This forces the model to learn features rather than memorising.

There are many more regularisation methods such as Cutout regularisation. Which was used in tutorial 3, Cutout is just the zeroing out of a random subset of each image. This resulted in slightly improved accuracy. Secondly, varying the learning rate schedule to peak earlier and linearly decay resulted in another increase in accuracy. This allowed for 30 epochs to be used and the result was 4/5 runs above 94% in 161s. Further increase in batch size resulted in another 7s drop. Highlighting the relevance of regularisation methods.

Task 2 ResNet on Fashion-MNIST to (80%)

The base model utilise was the ResNet18 model implemented in Dive into Deep Learning. Most of the functions used are derivatives of the code with the CNN and image processing chapters (Zhang, Lipton, Li, & Smola, 2021). Fashion-MINST is a dataset of 60,000 training images and 10,000 test images spanning 10 categories (Zhang et al., 2021).

Table 2 hyperparameter screening ResNet18 FashionMnist

Hyperparameters	Time per epoch (s)	Total time (s)	Accuracy (%)
Lr =0.01 **Batch size**=128 **Epochs**=10	44.9	449.2	91.2
Lr =OneCycle schedule(max = 0.1) **Batch size**=128 **Epochs**=10	45.6	456.4	93.5
Lr =OneCycle schedule(max = 0.1) **Batch size**=512 **Epochs**=2	42.7	85.5	91.2
Lr =OneCycle schedule(max = 0.4) **Batch size**=512 **Epochs**=2	42.7	85.4	90.6
Lr =OneCycle schedule(max = 0.4) **Batch size**=512 **Epochs**=2 With **random erasing**	45.3	90.7	88.7

LR schedule

Performance of CNNs is sensitive to learning rates, the learning rate dictates how much a weight is updated. A small learning rate in a model is slower since only small updates are being conducted on weights. Too high of a learning rate may result in missing the minimum or optimal weight, since large updates may "overshoot" the minimum (Park, Dokkyun, & Ji, 2020). Varying the learning rate allows for dealing with situations which require small learning rates such as near the minimum and when large gradients occur for example at the start large gradients occur thus small learning rate would be applicable and towards the end of training as weights approach the minimum, a small learning rate should be used to ensure the minimum is not missed (Park et al., 2020). Conversely, a larger learning rate speeds up computation since less updates are required before approaching the minimum, a schedule allows for taking advantage of these positive traits. Implementing the learning rate schedule resulted in an increase of 0.7s in time between epochs whilst still resulting in a higher accuracy (2.3%). Furthermore, increasing the learning rate resulted in a 0.6 % drop in accuracy with no difference in speed.

Batch size and number of Epochs

The ratio between batch size and learning rate is highly correlated with generalizability. Thus, if learning rate is optimised increasing batch size results in an increase of said ratio and a reduction in generalizability (F. He, Liu, & Tao, 2019). That being said a larger batch is likely to have lower variance similarly to momentum resulting in a smoother line which leading to quicker optimisations. Increasing batch size significantly reduced epoch time (2.9s). However, we saw a reduction

in accuracy. This could also be due to the reduction of epochs to 2. This was due to the model converging extremely quickly. The accuracy was very high without needing many epochs. This resulted in reducing the in total time by a factor of 5.

Random erasing

Random erasing was implemented to try and improve the models test accuracy via forcing it to learn more robust features. This is completed by erasing a random square subset of pixels of random size on random images. Increasing image variability. Random erasing resulted in an increase in time and a reduction in accuracy. It was removed from the model as the model seems to generalise to the test data early. Thus, extra regression may not be required.

Architecture

Since we saw an extremely dramatic drop-in time whilst maintain accuracy. It seemed applicable to adjust it further to see if some layers could be reduced. Reducing layers would result in faster epochs and a reduction in time. As per Myrtle.ai recommendations the network was stripped of all its long branches and the backbone was the result.

1. Backbone

The Backbone only consists of a preparation layer 4 layers with two batch norms and two ReLU activation functions , three of the layers also have a 1x1 kernel with stride 2 and a dense block. This backbone achieved an accuracy of 68.4% in 4 epochs at 122.2s in total. This is under our accuracy goal of 80% , thus, further structural changes were made.:

1. The repeating batch norm-ReLU groups were removed further shortening the backbone and resulting in an accuracy of this backbone achieved an accuracy of 76.4% in 4 epochs at 91.8s in total.
2. 3x3 convolutions with a stride of 1 were used instead on 1x1. The backbone networks primary issue is that the down sampling convolutions have 1×1 kernels and a stride of two, so that rather than enlarging the receptive field they are simply discarding information. The down sampling effect of the 1x1 kernels was addressed by the addition of max pooling. This resulted in 69.1% in 4 epochs at 121s in total.
3. A Global max pooling as then added to the dense layer doubling the output dimension of the final convolution to combat the reduction in input dimensions for the fully connected layer introduced from the maxpool down sampling. This dramatically increased accuracy to 82.8% after 4 epochs at 95.8 s.
4. Concerted all the ReLU activation functions to CELU activation function, which is a smooth activation function, possessing curvature at the origin. Assisting the optimisation process as well as better generalisation due to its less expressive nature. This paired with switch the maxpooling and batch norms produced further improvements at 83.3 % at 95.9s.

The 18-layer ResNet18 still performs better at 87.2% in 42.9s after 1 epoch with a batch size of 512 and a One cycle learning rate schedule. Further adaptations of the 9-layer ResNet could result in being able to achieve 805 in 1 epoch. Including Augmentation, label smoothing, weight decay , dropout layers etc. These methods would potentially allow for the model to generalise better and perform better on unseen data thus, resulting in better accuracy.

Task 3 ResNet on CIFAR-10 to (>90%)

Initial Architecture

Table 3. Table of models some of their initial hyper parameters and the speed and performance metrics.

Model	Hyper parameters	Accuracy	Time per epoch	Total time
ResNet18	Batchsize:128 Epochs:40 Lr:0.01	67.3	10.2	408.4
	Batchsize:512 Epochs:20	55.1	8.0	159.2

	Lr:0.01			
ResNet9	Batchsize:128 Epochs:10 Lr:0.01	68.5	5.6	56.0
	Batchsize:512 Epochs:20 Lr:0.01	59.9	4.9	98.6

Augmentation

The primary issue seems to be overfitting. The divergence between training accuracy and test accuracy elucidates the networks limited ability to generalise. A mitigation strategy for this overfitting is data augmentation. By modifying the images via flipping, resizing and random cropping we produce a greater variety of images and reduce the models ability to memorise images rather than recognise features (Takahashi, Matsubara, & Uehara, 2020). These three methods were implemented using Dive into deep learnings 14.13 method. Images were scaled up to a square 40x40 pixel image, random cropping was then used to randomly select a small square of 0.64-1 times the area of the image. The crop was then scaled to a 32x32 pixel square. Finally, the image was Normalised.

Learning rate scheduling

Implementation of a learning rate schedule significantly benefited the model's accuracy (18.7%) and reduced epoch times by 1.6s. The max learning rates were varied for rough tuning purposes.

Table 4 Learning rate schedules and their results

Learning rate max	Accuracy	Time per epoch	Total time
0.8	78.8	3.3	66.6
0.4	78.5	3.3	66.5

Although they both provided very similar results, the model with the lower learning rate max (0.04) shows signs of overfitting at the end as it begins to diverge from the training accuracy.

Momentum (78.8% at 66.9s)

A momentum term was then also added to the stochastic gradient descent algorithm to try and increase the speed of convergence. When SGD approaches a local minimum, it behaves like a damped harmonic oscillator. The momentum term can be viewed as the level of dampening applied to the harmonic oscillator, the dampening is achieved by computing the exponentially weighted average and using that to update the weights of the denoise weights (Robert, 2014). Small or no momentum is equivalent to underdamping thus oscillating about the minimum slowing convergence and two high is equivalent to over dampening establishing a smooth but shifted line missing the true minimum (Qian, 1999). A good momentum term pushes the systems eigenvalue components towards critical dampening accelerating convergence. This also stops SGD from getting stuck in local minimums. A momentum term of 0.9 was chosen. The result was unexpected as there was no reduction in training time. However, it was left in the model to assess the impact on larger runs.

Weight decay (78.8% at 66.9s)

Since the primary issue with accuracy seems to be over fitting implementation of weight decay seemed like a local progression. Weight decay is a L2 regularization method. Adding our weight parameters to the loss function minimises the weights penalising complexity (Li, Jian, Wen, & AlSultan, 2022). This allows us to create models that are more generalizable reducing prediction error. The weight decay was set to 1e-4. No difference in time or accuracy was seen however, it did not result in an increase epoch time, so it was kept in the model to ensure not to lose any potential benefits in later runs. A rough tuning was completed however, accuracies began to drop dramatically.

Random erasing (78.7% at 66.9s)

Random erasing was introduced as a substitution for Cutout regression. Random erasing masks a subsection of an image, however it randomly decides the size and aspect ratio of the masked region as well as which images to complete it on (Takahashi et al., 2020). Similarly, this reduces overfitting and allows for a more generalizable model as it must learn more robust features to minimise the loss. The implementation of random erasing resulted in no increase in accuracy or time over 20 epochs. However, the training accuracy and test accuracy are significantly tighter together and thus, it was kept in the model.

Epoch and Batch size (81.2% ,133.7s ,40 epochs), (78.8% ,66.9s ,20 epochs) & (67.4% ,64.8s ,20 epochs)

Varying Epochs from 20-40 at this stage resulted in no substantial gain with only a 2.4% accuracy increase. Furthermore, increasing batch size from 512 to 726 resulted in a 1.2% loss with only a 0.1s gain in speed per epoch. Thus, establishing themselves as not very viable options at this stage.

Label smoothing & Scaling batch norm scales (87.9%, 68.8s, 20 epoch)

Label smoothing was then utilised to as a further regression technique to reduce overfitting. Label smoothing introduces a small amount of uncertainty into the labels used for training. Rather than binary hot encoded labels, label smoothing assigns a probability to each label. This smooths the labelling reducing the model's confidence in predictions and improving generalisability (Szegedy, Vanhoucke, Ioffe, Shlens, & Wojna, 2016).

As elucidated by Myrtle.ai tutorials by default Pytorch initialises the batch norm scales uniformly at random between 0 and 1. Initialising a channel near zero could result in the loss of channels thus, it was initialised with 1. This results larger signals through the network which was compensated for via the scaling of the final classifier. The value 0.125 was used as per the recommendations of Myrtle.ai.

The combination of labelling's regression and normalisation of the batch norm scales resulted in an increase in accuracy by 9.1% whilst keeping over all time at 68.8 seconds. Outcompeting all the previous models.

Epoch and Batch size (90.2%, 191.1s, 50 epochs)

The epochs where then doubled again to try and achieve 90 % however it fell short. (89.2%). The batch size was reduced from 512 to 256 . The number of epochs also was increased by 10 from 40 to 50. Furthermore, a slight increase in learning rate was paired and resulting in passing 90% accuracy.

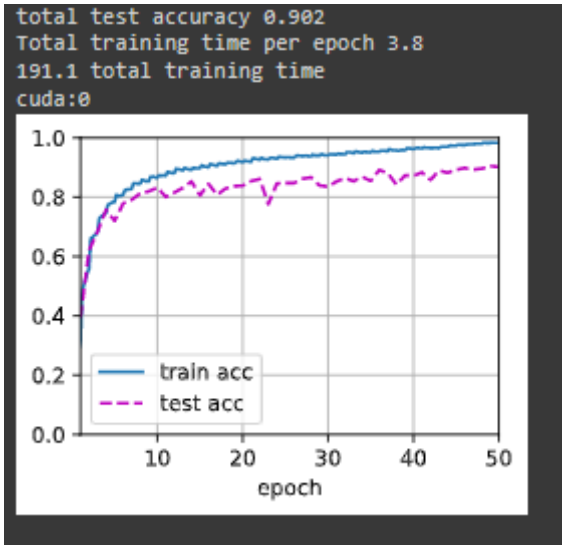

Figure 1 Final output for CIFAR10 data set

Architecture

Finally, the models had their maxpool and batch norms switched. This was done as it was recommended and resulting in a reduction in accuracy.

References

Buda, M., Maki, A., & Mazurowski, M. A. (2018). A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks, 106*, 249-259. doi:10.1016/j.neunet.2018.07.011

Eigen, D., Rolfe, J., Fergus, R., & LeCun, Y. (2013). Understanding Deep Architectures using a Recursive Convolutional Network. doi:10.48550/arxiv.1312.1847

He, F., Liu, T., & Tao, D. (2019). Control batch size and learning rate to generalize well: theoretical and empirical evidence. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems* (pp. Article 103): Curran Associates Inc.

He, K., Zhang, X., Ren, S., & Sun, J. (2016, 27-30 June 2016). *Deep Residual Learning for Image Recognition.* Paper presented at the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Li, G., Jian, X., Wen, Z., & AlSultan, J. (2022). Algorithm of overfitting avoidance in CNN based on maximum pooled and weight decay %J Applied Mathematics and Nonlinear Sciences. *7*(2), 965-974. doi:doi:10.2478/amns.2022.1.00011

Park, J., Dokkyun, Y., & Ji, S. (2020). A Novel Learning Rate Schedule in Optimization for Neural Networks and It's Convergence. *Symmetry, 12*(4), 660. doi:https://doi.org/10.3390/sym12040660

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks, 12*(1), 145-151. doi:https://doi.org/10.1016/S0893-6080(98)00116-6

Robert, C. (2014). Machine Learning, a Probabilistic Perspective. In (Vol. 27, pp. 62-63). Abingdon: Taylor & Francis.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016, 27-30 June 2016). *Rethinking the Inception Architecture for Computer Vision.* Paper presented at the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Takahashi, R., Matsubara, T., & Uehara, K. (2020). Data Augmentation Using Random Image Cropping and Patching for Deep CNNs. *IEEE Transactions on Circuits and Systems for Video Technology, 30*(9), 2917-2931. doi:10.1109/TCSVT.2019.2935128

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2021). Dive into Deep Learning. doi:10.48550/arxiv.2106.11342