Comparative Study on Various Adaptive Learn-Rate Optimisers on Image Tasks

Zitang Ren

March 25, 2024

1 Abstract

This study aims to evaluate the impact of utilising different adaptive learn-rate optimization tools on image classification tasks, as well as the effect of new features on accuracy and computational time. The study compares the performance of an Adam, RMSprop and AMSGrad optimizer on various image recognition tasks (digit recognition, image classification). The study evaluates the computational time for each optimiser as well as several benchmarks of the model's resultant accuracy, being the F1-Score and the mean squared error. This study aims to compare the practical benefit of newer iterations of adaptive learn-rate algorithms in terms of computational time and accuracy of prediction in image classification tasks. The content of the study has been taken from MNIST and CIFAR datasets and used to train models for all three algorithms. The results of this study indicate that the RMSProp optimiser has a generally higher compute speed and accuracy when compared to the Adam and AMSGrad models.

2 Introduction

The necessity and usage of image classification algorithms is rapidly growing in society due to the widespread commercialisation of machine vision based products and services such as driver-less vehicles and medical image classifiers [Cai20]. The ability to replace human vision with a machine is a crucial component in the process of automating tedious and repetitive tasks and is achieved through machine learning. Machine learning is a technique that uses a large corpus of data to perform a specific task without receiving the explicit instruction as to how [Mah19] - in our case, the identification or classification of objects in images. This task is approached in various ways, one of which being a supervised learning approach. It trains a model using labelled sets of data to classify or identify objects into labels based on features [Dou13]. There exist many supervised learning algorithms to classify data into two or more labels

such as Logistic Regression, Support Vector Machines and Convolutional Neural Networks [Pin21].

A classifier's performance is dependant on various factors, with the utilised optimiser being one of the most important. Computational power is irrelevant if the resources are wasted on inefficient model training. The optimization of machine learning models involves navigating complex and often high-dimensional parameter spaces. Classic optimization techniques, such as gradient descent, have laid the groundwork for contemporary methodologies [Ros57]. The optimiser influences the speed the model is able to be trained and the accuracy of the models outputs through adjustments of the models weights and biases. These parameters are relevant as predictions in particular industrial sectors will require higher precision, such as medical image classification, whereas other fields would require rapid computation speed. This study will measure and evaluate the performance of three such optimisation algorithms: Adam, RMSProp and AMSGrad [Kin14] [Tie12] [Red19]. As all the above optimisers are a form of adaptive learn-rate algorithm and are built off each other, the purpose of this paper is to evaluate the practical effectiveness of these additions on basic and complex image classification in terms of computational time and overall accuracy.

This paper is structured as such: Section 2 provides contextual detail for each optimisation algorithm used. Datasets used will be standard datasets from various types of classification tasks, namely from the Modified National Institute of Standards and Technology (MNIST) and the Canadian-based global research organization (CIFAR). Each optimisation algorithm requires a certain amount of hyperparameter tuning which will affect the performance of each algorithm [Li16] and the effects of various degrees of hyperparameter tuning will be demonstrated through the experiments. Section 3 will outline the results of the study with plots and diagrams. Section 4 will discuss the results along with potential experimental inaccuracies. Section 5 serves to conclude the study.

3 Context, Resources and Procedure

Optimization algorithms have been employed to train diverse models for a broad spectrum of tasks. The analysis of these models is conducted using Python 3.11 utilizing hardware specifications that include a 2GHz Ryzen 7 5825U processor, 16GB RAM, and an NVIDIA GeForce RTX 3050Ti laptop GPU. In order to provide an explanation of the basic functional principles of the optimisations used, the fundamental of optimisation will be briefly explored.

3.1 Algorithms

3.1.1 Gradient Descent

Gradient descent serves as the foundation for many optimisation algorithms, where it iteratively adjusts model parameters in a way that minimises the value of a loss function. The gradient points towards the direction of the steepest

descent of the loss value and by negating it, a parameter is guided towards said direction of steepest descent. This is given by:

$$\theta_{t+1} = \theta_t - \alpha \cdot \nabla J(\theta_t) \tag{1}$$

Where θ_t represents the parameter vector at time t, α denotes the learning rate (step size while moving towards minimum), $J(\theta_t)$ is the loss function, and $\nabla J(\theta_t)$ is the gradient of the loss function [Rum86].

3.1.2 Momentum-Based Descent

Momentum-based gradient descent is an extension of the conventional gradient descent algorithm that introduces a dynamic element of momentum into the optimisation process. The accumulated velocity from prior optimisation iterations guides the current optimisation process, analogous to how a moving object's momentum in the real-world functions. This preserves the algorithm's so-called 'movement', enabling swift traversal of the optimisation landscape. Mathematically, this accumulated velocity is expressed as:

$$v_t = \gamma \cdot v_{t-1} + \alpha \cdot \nabla J(\theta_t) \tag{2}$$

$$\theta_{t+1} = \theta_t - v_t \tag{3}$$

Here, v_t denotes the accumulated velocity at time t and γ represents the momentum coefficient [Pol64]]. γ is often chosen to be a value between 0 and 1, where values close to 1 would allow for fast traversal of flat regions and values close to 0 would dampen the effect of past velocities. Momentum-based descent decreases the computational time spent on training.

3.1.3 Adaptive Learn-Rates

An adaptive learn-rate strategy involves dynamically altering the learning rates for different parameters during learning. This becomes exceptionally important when we approach non-convex, high-dimensional machine learning problems [Dau15]. The core idea behind adaptive learn-rate is to take large steps in less sensitive directions on the gradient and small steps in more sensitive directions. An example of an adaptive learn-rate algorithm is the AdaDelta optimiser:

$$\Delta x_t = -\frac{\eta}{\sqrt{\sum_{\tau=1}^{t} g_\tau^2}} \cdot g_t \tag{4}$$

The parameter update at time t is updated using the equation above [Zei12]. η is the preset hyperparameter learn-rate, $\sqrt{\sum_{\tau=1}^{t} g_\tau^2}$ is the accumulated sum of the squares of the gradients up to time t and g_t denotes the loss function at time t.

3.1.4 Adam Optimiser

The Adam optimiser is an amalgamation of momentum-based gradient descent and adaptive learn-rate strategies. It calculates exponential moving averages of past gradients and past squared gradients to adaptive adjust learn-rates for each parameter [Kin14]. It inherits the advantage of momentum-based descent in its fast traversal of complex non-convex landscapes.

The algorithm calculates the moving average of all the gradients at time t, m_t using the exponential decay rate β_1, and the squared gradients v_t as such:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \tag{5}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{6}$$

To mitigate biases approaching zero, the bias-corrected moving averages $\hat{m}_t$ and $\hat{v}_t$ is computed as follows:

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t} \tag{7}$$

$$\hat{v_t} = \frac{v_t}{1 - \beta_2^t} \tag{8}$$

The parameter update rule is then created, where ϵ is a small constant to prevent zero division from occurring:

$$\theta_{t+1} = \theta_1 - \frac{\alpha}{\sqrt{\hat{v_t} + \epsilon}} \cdot \hat{m_t} \tag{9}$$

The Adam method is a first-order and has low time complexity [Yi20].

3.1.5 Root Mean Square Propagation

Root Mean Square Propagation (RMSProp) is an improvement to the Adagrad that addresses the limitations of accumulation of squared gradients and the overly aggressive decrease in learn-rate over time [Rud16]. RMSProp adapts the learn-rate individually for each parameter and normalises the learn-rate using the RMS of squared gradients [Har13]. Compared to Adam, RMSProp places more emphasis on normalising learn-rates whereas Adam combines momentum-descent and adaptive learn-rate.

The central equation of RMSProp updates parameters using the normalised gradient and the squared RMS of past gradient:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_{t,i} \tag{10}$$

Where $E[g^2]_t$ signifies the weighted moving average of squared gradients of parameter i at time t.

3.1.6 Adaptive Moment Estimation for Stochastic Gradient Descent

Adaptive Moment Estimation for SGD, or AMSGrad for short, addresses a limitation of the Adam optimiser, where the learning rate can grow unbounded and lead to slow convergence to a minima. AMSGrad updates the update rule of Adam to prevent this issue while preserving the benefit of adaptive learning rates and momentum. The update rule of AMSGrad is as follows:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{\hat{v}_{t,ii} + \epsilon}} \cdot \hat{m}_{t,i} \tag{11}$$

This equation is similar to Adam's update rule where the change lies in the denominator. $\hat{v}_{t,ii}$ is the bias-corrected moving average of past gradients for the current parameter i and $\hat{m}_{t,i}$ which now takes into account i [Red19].

3.2 Datasets

This study has used the aforementioned Adam, RMSProp and AMSGrad optimisers to classify images. These algorithms are studied on image-only datasets seen in Table 1. The datasets chosen are a mix of big data and small data with different amounts of attributes. Datasets from this study are collected from MNIST and CIFAR. An evaluation of the quality of the dataset is not necessary as the data present is sanitised and ready to use in training. The MNIST dataset provides 70,000 monochromatic images of 28 by 28 pixels of handwritten digits and 10 total classes, for the numbers 0 through 9. The CIFAR-10 dataset consists of 60,000 color 32 by 32 images with 10 classes of common objects (airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck). The CIFAR-100 dataset consists of similar images as CIFAR-10 but instead features 100 total classes with 600 image each.

Dataset Name	**Num. of Instances**	**Num. of Attributes**
MNIST	70,000	784
CIFAR-10	60,000	3072
CIFAR-100	60,000	3072

Table 1: Dataset Information

3.3 Hyperparameter Setting and Tuning

3.3.1 Batch Size

The batch size hyperparameter determines the number of samples used in each train iteration to compute gradients and update model parameters. It determines how many data points are included in each gradient computation and subsequent parameter update [Bot18]. Large batch sizes lead to faster convergence at the cost of high computational power with no significant impact on

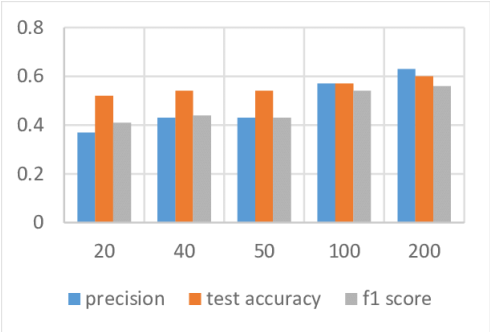

Figure 1: The relationship between number of epochs, precision, test accuracy and F1-Score

accuracy [Kan20], whereas small batch sizes lead to more noise but could potentially create sharper minima [Jas18]. This study will use a batch size of 64, which is deemed a generally acceptable value for image classification and runs sufficiently optimally on laptop computers.

3.3.2 Epochs

An epoch defines how many times a model will iterate over the full dataset during the optimisation process. All optimisation algorithms need the number of epochs as a parameter. A higher number of epochs may not necessitate in higher accuracy, as seen in Fig. 1 [Yas17]. Each model in this study will run 5, 10, 15, 20, 25 and 30 epochs to more fairly evaluate the performance of each optimiser.

3.3.3 Decay Rate

Decay rate controls how much the algorithm considers past gradients when updating parameters. Rates close to 1 gives more weight to recent gradients and vice versa for values close to 0. In RMSProp and AMSGrad, this value influences the exponential moving average of squared gradients [Tie12] [Red19]. In Adam, this value influences the moving average of past gradients [Kin14]. An excessively large decay rate will decrease the learn-rate too rapidly and result in poor performance, whereas a excessively low decay rate will result in no decay at all. This study will use a generally optimal decay-rate of 0.001 [Jas20].

3.3.4 Convolutional Neural Networks

A convolutional neural network (CNN) is a deep learning algorithm that convolves input images with filters, or kernels, to identify and extract images. An image of $N \times N$ is convolved with a $f \times f$ filter and this convolution learns the same feature on the entire image [Zei14]. These features are learnt by feature maps that capture the receptive field of the image. The equation of a

convolution is as follows [Cha18]:

$$O = \max(0, b + \Sigma_{i=0}^{2} \Sigma_{j=0}^{2} w_{i,j} h_{a+i,b+j}) \tag{12}$$

Here, O is the output, w is a matrix of shared weights, $\max(0, x)$ is a Rectified Linear Unit (ReLU) activation function, and $h_{x,y}$ is the activation at position x, y. Since the datasets used have varying dimensions, the CNNs used for each will vary.

4 Methodology

This study will create nine separate CNN models, all of which will be kept simplistic for the purpose of demonstration and brevity, and to demonstrate the effects of optimsers rather than the model itself. Each model uses one convolutional layer with a Rectified Linear Unit (ReLU), a max-pooling layer, a flatten layer and one dense hidden layer.

4.1 Evaluation Methodology

This study will use several metrics to gauge the prediction accuracy and computational speed of each model. Raw collected data will be presented as line graphs or bar charts, plotting epochs against accuracy and F1 score. The models degree of correctness is calculated and measured with three main metrics: accuracy, precision and F1 Score. Accuracy is calculated as:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{13}$$

Where TP/TN is the number of True Positive/Negative predictions, and FP/FN is the number of False Positive/Negative predictions. Precision is calculated as follows:

$$precision = \frac{TP}{TP + FP} \tag{14}$$

F1 Score is a more representative weighted average of precision and recall (given by $\frac{TP}{TP+FN}$), calculated by:

$$Score = 2 \times \frac{(Recall \times Precision)}{(Recall + Precision)} \tag{15}$$

5 Results

5.1 MNIST Results

All models performed well on the MNIST dataset, averaging very similar accuracies of around 98%. Accuracy and F1 score variation across number of epochs was not significant. A low number of epochs resulted in an expected drop in

Table 2: Effect on Adam Optimiser Computation Time with Changing Epoch Count in the MNIST Dataset

No. Epochs	**Compute Time (sec)**
5	80.51
10	154.02
15	223.46
20	295.51
25	387.08
30	467.60

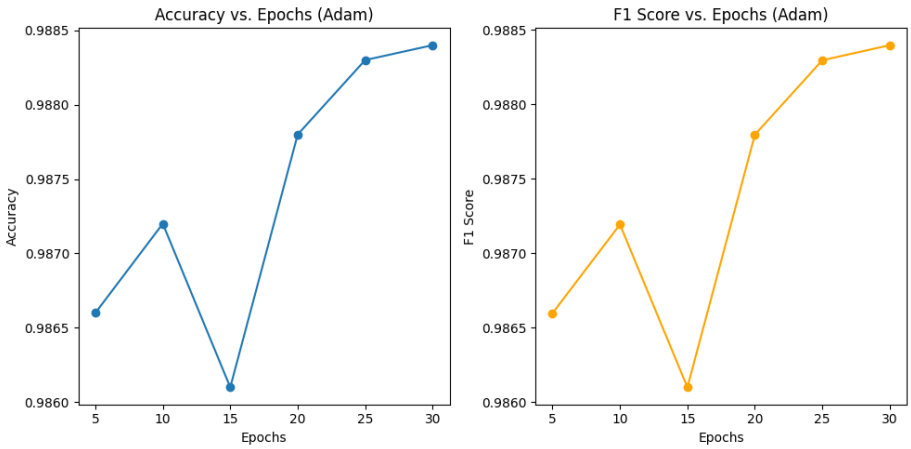

Figure 2: Adam Performance on MNIST

accuracy and all optimisers evened out at 10 epochs and above. The compute time varied more, with Adam having the highest average compute time across all number of epochs. The fastest optimiser was RMSProp with the lowest running average across all number of epochs.

Table 3: Effect on RMSProp Optimiser Computation Time with Changing Epoch Count in the MNIST Dataset

No. Epochs	**Compute Time (sec)**
5	47.72
10	93.63
15	152.14
20	207.30
25	265.05
30	309.01

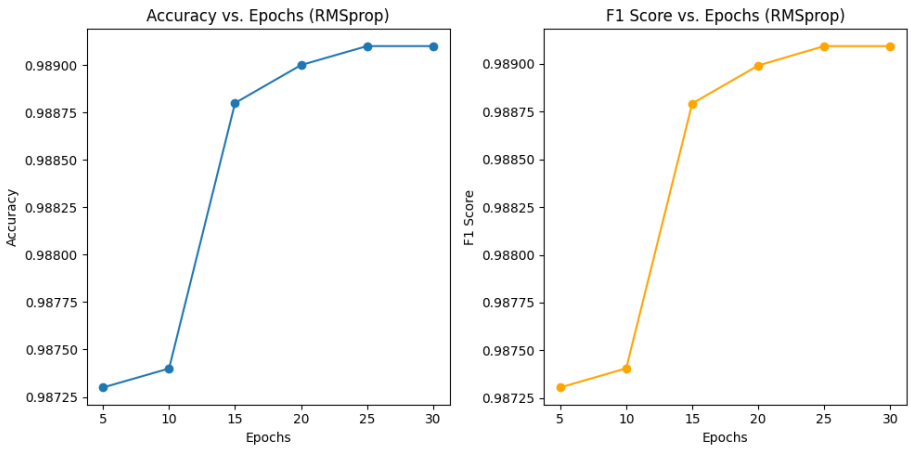

Figure 3: RMSProp Performance on MNIST

Table 4: Effect on AMSGrad Optimiser Computation Time with Changing Epoch Count in the MNIST Dataset

No. Epochs	**Compute Time (sec)**
5	66.69
10	127.58
15	191.35
20	258.95
25	339.41
30	395.86

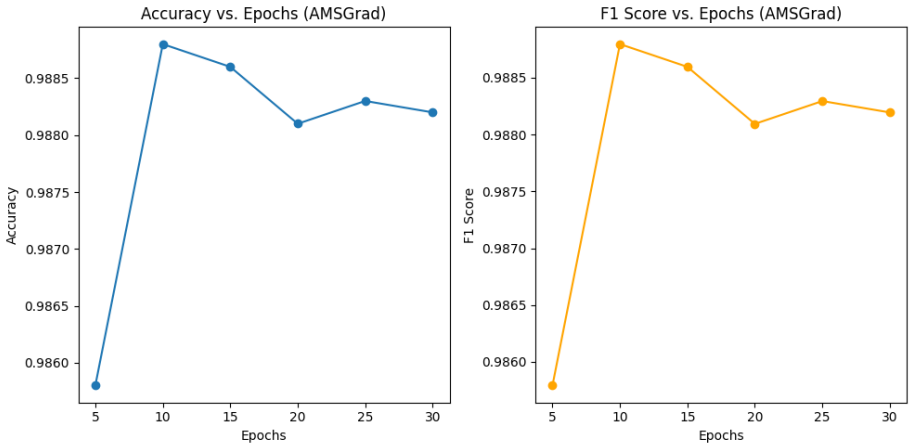

Figure 4: AMSGrad Performance on MNIST

5.2 CIFAR-10 Results

As expected, the average accuracy dropped when facing more complex image tasks. However, there is still no notable difference between the performance of each optimiser, except that accuracy decreases with an increase in epoch count, likely due to underfitting in low epoch counts. Compute time for each optimiser is relatively close, with RMSProp still performing the fastest across all number of epochs. AMSGrad and Adam had very similar times across all epochs.

Table 5: Effect on Adam Optimiser Computation Time with Changing Epoch Count in the CIFAR-10 Dataset

No. Epochs	**Compute Time (sec)**
5	66.61
10	149.47
15	218.97
20	292.47
25	366.73
30	441.16

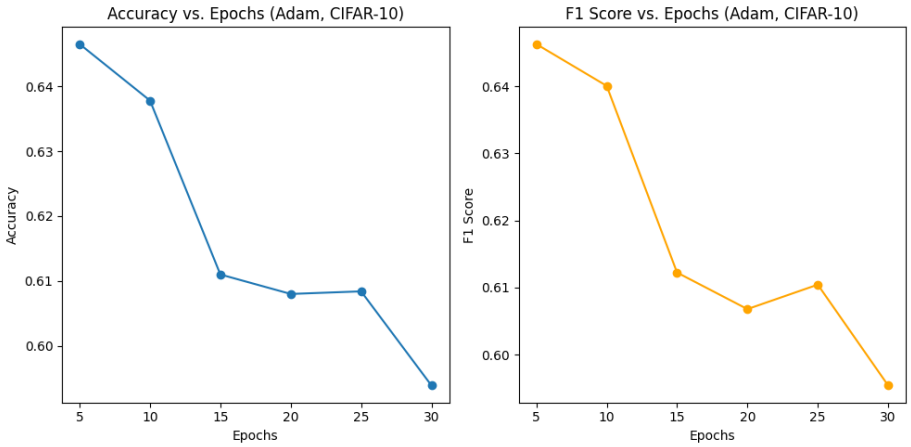

Figure 5: Adam Performance on CIFAR-10

Table 6: Effect on RMSProp Optimiser Computation Time with Changing Epoch Count in the CIFAR-10 Dataset

No. Epochs	**Compute Time (sec)**
5	57.81
10	113.39
15	175.41
20	239.36
25	296.22
30	356.53

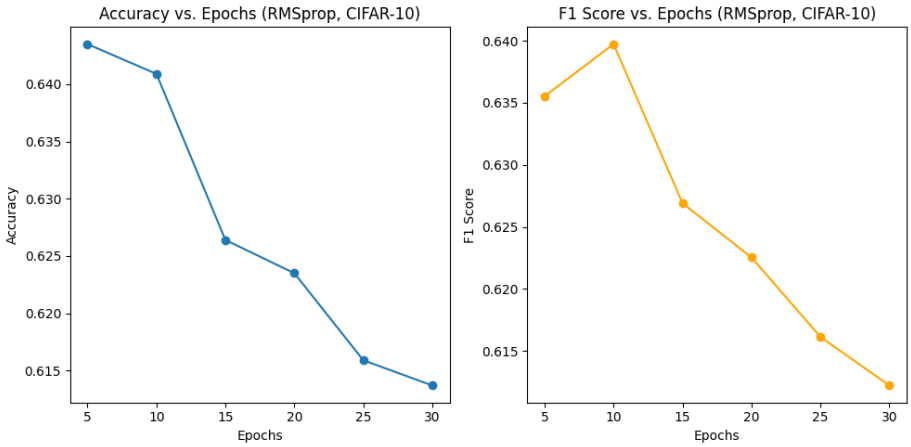

Figure 6: RMSProp Performance on CIFAR-10

Table 7: Effect on AMSGrad Optimiser Computation Time with Changing Epoch Count in the CIFAR-10 Dataset

No. Epochs	**Compute Time (sec)**
5	69.75
10	179.23
15	244.21
20	312.75
25	383.71
30	458.06

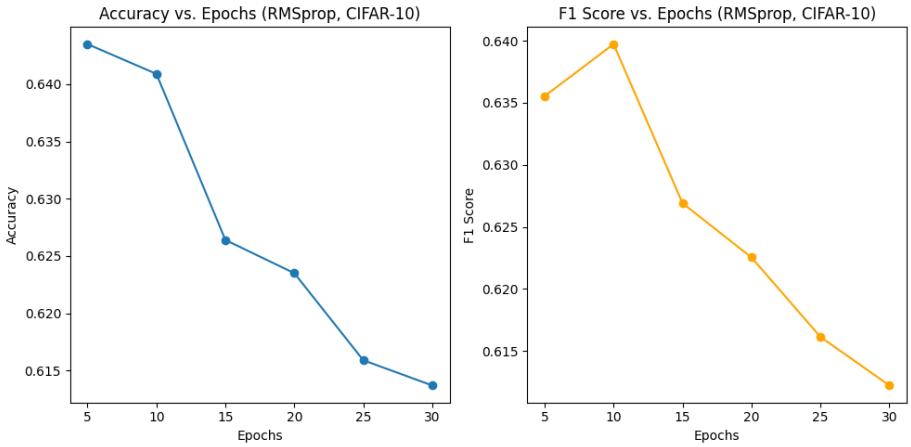

Figure 7: AMSGrad Performance on CIFAR-10

5.3 CIFAR-100 Results

Variation in accuracy became more significant between each optimiser. RMSProp achieved the highest average accuracy of 36.56% while Adam and AMSGrad performed similarly at around 34.5%. RMSProp had very similar compute time compared to Adam with a negligably small difference across all epochs. AMSGrad had the slowest compute time across all epochs. Accuracy experienced a drop after 25 epochs in AMSGrad and RMSProp models whereas Adam seems to stabilise at around 35% accuracy after 15 epochs.

Table 8: Effect on Adam Optimiser Computation Time with Changing Epoch Count in the CIFAR-100 Dataset

No. Epochs	**Compute Time (sec)**
5	48.20
10	94.03
15	142.50
20	190.04
25	231.91
30	284.50

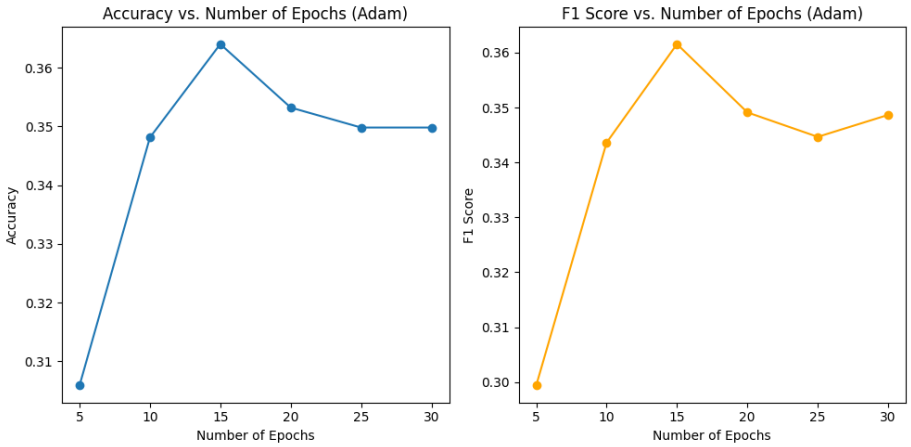

Figure 8: Adam Performance on CIFAR-100

Table 9: Effect on RMSProp Optimiser Computation Time with Changing Epoch Count in the CIFAR-100 Dataset

No. Epochs	**Compute Time (sec)**
5	48.05
10	95.39
15	141.67
20	189.79
25	263.26
30	291.71

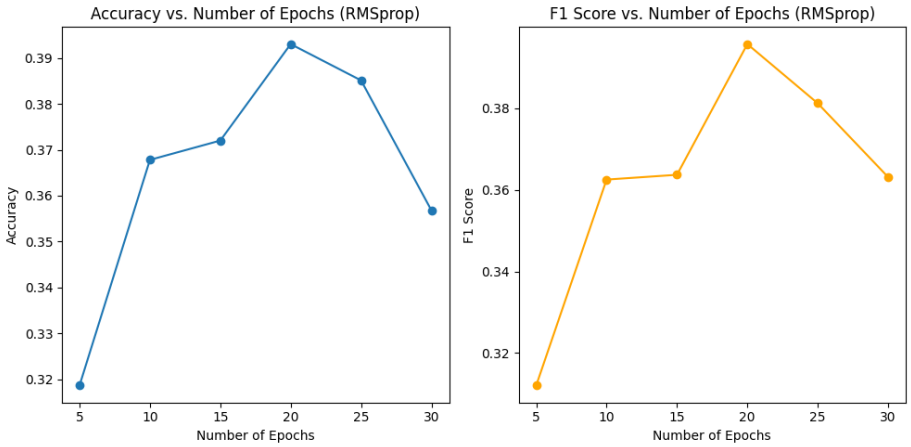

Figure 9: RMSProp Performance on CIFAR-100

Table 10: Effect on AMSGrad Optimiser Computation Time with Changing Epoch Count in the CIFAR-100 Dataset

No. Epochs	**Compute Time (sec)**
5	54.55
10	101.64
15	152.29
20	200.30
25	251.60
30	303.18

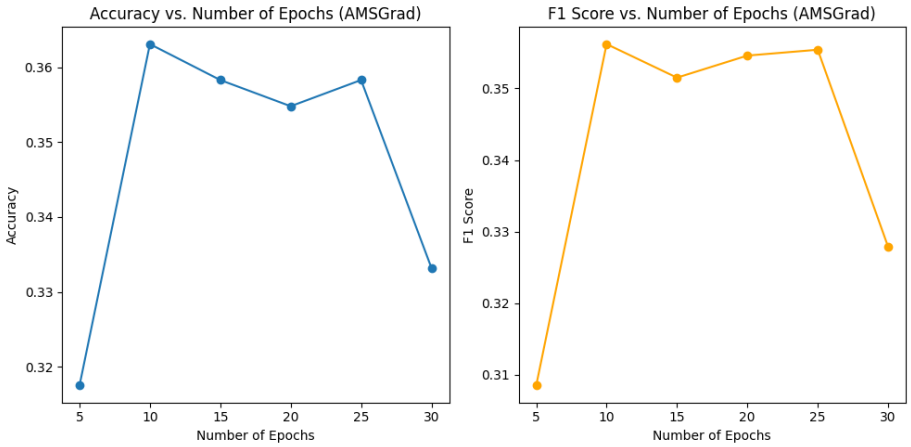

Figure 10: AMSGrad Performance on CIFAR-100

6 Evaluation

This study has conducted a comparison of various optimiser algorithms on various image tasks, evaluating their accuracy and computational speed. The experimental results show that RMSProp is overall the fastest optimiser compared to Adam and AMSGrad, and while all models performed similarly in terms of accuracy. RMSProp showed a 2% increase when compared to Adam and AMSGrad when faced with a image classification task with many classes.

7 Conclusion and Future Work

After evaluating the impact of various optimisation algorithms and observing whether improvements to the update rule itself has significant tangible impact on compute speed and accuracy, the results suggest the difference may not be major when working with smaller dataset of around 60,000 instances. Across the experimental tests, RMSProp demonstrated the lowest compute time and higher accuracy when faced with complex image classification problems. It may not be conclusive due to the low feature space dimension but still provides insight into the effectiveness of different optimisation algorithms. This study can be extended to test on realistic datasets or expanding the number of optimisers examined.

References

[1] [Cai20] Cai, L., Gao, J., & Zhao, D. (2020). A review of the application of deep learning in medical image classification

and segmentation. Annals of translational medicine, 8(11), 713. https://doi.org/10.21037/atm.2020.02.44

[2] [Mah19] Mahesh, Batta. (2019). Machine Learning Algorithms - A Review. 10.21275/ART20203995.

[3] [Dou13] G. Dougherty (2013), Pattern Recognition and Classification: An Introduction. Spring New York, ISBN 978-1-4614-5323-9.

[4] [Pin21] Pin Wang, En Fan, Peng Wang, Comparative analysis of image classification algorithms based on traditional machine learning and deep learning, Pattern Recognition Letters, Volume 141, 2021, Pages 61-67, ISSN 0167-8655, https://doi.org/10.1016/j.patrec.2020.07.042.

[5] [Ros57] Rosenblatt, F. (1957). The perceptron, a perceiving and recognizing automaton. Cornell Aeronautical Laboratory. Psychological Review, 65(6), 386-408.

[6] [Kin14] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[7] [Duc11] Duchi, John & Hazan, Elad & Singer, Yoram. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research. 12. 2121-2159.

[8] [Tie12] Tieleman, T. and Hinton, G. (2012) Lecture 6.5-rmsprop: Divide the Gradient by a Running Average of Its Recent Magnitude. COURSERA: Neural Networks for Machine Learning, 4, 26-31.

[9] [Li16] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2016). Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. Journal of Machine Learning Research, 18(185), 1-52.

[10] [Rum86] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533-536.

[11] [Pol64] B.T. Polyak, Some methods of speeding up the convergence of iteration methods, USSR Computational Mathematics and Mathematical Physics, Volume 4, Issue 5, 1964, Pages 1-17, ISSN 0041-5553, https://doi.org/10.1016/0041-5553(64)90137-5. (https://www.sciencedirect.com/science/article/pii/0041555364901375)

[12] [Dau15] Dauphin, Y., De Vries, H., & Bengio, Y. (2015). Equilibrated adaptive learning rates for non-convex optimization. Advances in neural information processing systems, 28.

[13] [Zei12] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.

[14] [Yi20] Yi, D., Ahn, J., & Ji, S. (2020). An effective optimization method for machine learning based on ADAM. Applied Sciences, 10(3), 1073.

[15] [Ru16] Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

[16] [Ha13] Hardmeier, C., Tiedemann, J., & Nivre, J. (2013). Latent anaphora resolution for cross-lingual pronoun prediction. In EMNLP 2013; Conference on Empirical Methods in Natural Language Processing; 18-21 October 2013; Seattle, WA, USA (pp. 380-391). Association for Computational Linguistics.

[17] [Ku17] Kurbiel, T., & Khaleghian, S. (2017). Training of deep neural networks based on distance measures using RMSProp. arXiv preprint arXiv:1708.01911.

[18] [Re19] Reddi, S. J., Kale, S., & Kumar, S. (2019). On the convergence of adam and beyond. arXiv preprint arXiv:1904.09237.

[19] [Kri10] Krizhevsky, A., & Hinton, G. (2010). Convolutional deep belief networks on cifar-10. Unpublished manuscript, 40(7), 1-9.

[20] [Cha18] R. Chauhan, K. K. Ghanshala and R. C. Joshi, "Convolutional Neural Network (CNN) for Image Detection and Recognition," 2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC), Jalandhar, India, 2018, pp. 278-282, doi: 10.1109/ICSCCC.2018.8703316.

[21] [Bot18] Bottou, L., Curtis, F. E., & Nocedal, J. (2018). Optimization methods for large-scale machine learning. SIAM review, 60(2), 223-311.

[22] [Kan20] Kandel, I., & Castelli, M. (2020). The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. ICT express, 6(4), 312-315.

[23] [Jas18] Jastrzebski, S., Kenton, Z., Arpit, D., Ballas, N., Fischer, A., Bengio, Y., & Storkey, A. (2018). Width of minima reached by stochastic gradient descent is influenced by learning rate to batch size ratio. In Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III 27 (pp. 392-402). Springer International Publishing.

[24] [Yas17] Yasser, A., Clawson, K., Bowerman, C., & Lévêque, M. (2017, July). Saving cultural heritage with digital make-believe: machine learning and digital techniques to the rescue. In HCI'17: Proceedings of the 31st British Computer Society Human Computer Interaction Conference (No. 97, pp. 1-5). ACM.

[25] [Jas20] Jason, B. (2020, September). Understand the Impact of Learning Rate on Neural Network Performance. In Deep Learning Performance, https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

[26] [Zei14] Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13 (pp. 818-833). Springer International Publishing.