Performance comparison of the convolutional neural network optimizer for photosynthetic pigments prediction on plant digital image

Cite as: AIP Conference Proceedings **2084**, 020020 (2019); https://doi.org/10.1063/1.5094284
Published Online: 22 March 2019

K. R. Prilianti, T. H. P. Brotosudarmo, S. Anam, and A. Suryanto

Performance Comparison of the Convolutional Neural Network Optimizer for Photosynthetic Pigments Prediction on Plant Digital Image

K.R. Prilianti[1,3,a)], T.H.P. Brotosudarmo[2], S. Anam[3] and A. Suryanto[3]

[1]*Department of Informatics Engineering, Ma Chung University, Indonesia*
[2]*Ma Chung Research Center for Photosynthetic Pigments, Indonesia*
[3]*Department of Mathematics, Brawijaya University, Indonesia*

[a)]Corresponding author: kestrilia.rega@machung.ac.id

Abstract. Determination of photosynthetic pigments in intact leaves is an essential step in the plant analysis. Along with the rapid development of digital imaging technology and artificial intelligence, now determination of plant pigments can be done in a non-destructive and real-time manner. In previous research, a prototype of the non-destructive and real-time system has been developed by utilizing the Convolutional Neural Network (CNN) model to produce predictions of three main photosynthetic pigments, i.e., chlorophyll, carotenoid, and anthocyanin. The CNN model was chosen due to its ability to handle raw digital image data without prior feature extraction. In the near future, this ability will be useful for developing analytical portable devices. Input of the system is multispectral plant digital image, and the output are predicted pigment concentration. Convolutional Neural Network performance depends on several factors, among them are data quality, algorithm tuning (weight initialization, learning rate, activation function, network topology, batches and epochs, optimization and loss) and models combination. The focus of this research is to improve the accuracy of CNN model by optimizing the selection for updating CNN architecture parameters which are optimization method and the loss function. As it is already known, there is no single optimizer can outperform for all cases. The selection for the optimizer should be made by considering the variability of the data and the nonlinearity level of the relationship patterns that exist in the data. Because the theoretical calculation is not enough to determine the best optimizer, an experiment is needed to see at firsthand the performance of optimizers that allegedly matches the characteristics of the data being analyzed. Gradient descent optimization method is well known for its ease of computing and speed of convergence on large datasets. Here, 7 gradient descent-based optimizers were compared, i.e., Stochastic Gradient Descent (SGD), Adaptive Gradient (Adagrad), Adaptive Delta (Adadelta), Root Mean Square Propagation (RMSProp), Adaptive Momentum (Adam), Adaptive Max Pooling (Adamax), and Nesterov Adaptive Momentum (Nadam). From the eksperiment result it is known that Adam is the best optimizer to improve LeNet ability in handling a digital image-pigments content relationship. However, when the resources for experimentation is limited, using Adadelta and Adamax is a wise choice to minimize risk.

INTRODUCTION

Convolutional Neural Network is the development of Artificial Neural Network (ANN) models which was initially designed to be able to handle input data in the form of digital images. Unlike ANN models which require the efforts of researchers to determine features of the digital image that are most suitable to be used as an input, the CNN models are designed to be able to find these features automatically. The CNN architecture was first introduced by Fukushima [1] and improved by LeCun [2]. However, CNN did not gain proper attention until the success of the AlexNet architecture in 2012 [3]. AlexNet produces up to 15.3% error rates when classifying ImageNet and won the LSVRC (Large Scale Visual Recognition Challenge) competition. This achievement became the starting point of the development of other CNN models. Nowadays, CNN experienced very rapid development in the various field, e.g., computer vision (face recognition, scene labeling, image classification, action recognition, human pose estimation) document analysis) and natural language processing (speech recognition, text classification) [4].

Optimization algorithm in CNN architectures was first introduced by LeCun with gradient-based technique. It is obtained a good result for the handwritten digit classification problem [2]. In the following years, gradient-based methods are very popular to use because of the ease of implementation. However, since vanishing gradient problem

Proceedings of the Symposium on BioMathematics (SYMOMATH) 2018
AIP Conf. Proc. 2084, 020020-1–020020-8; https://doi.org/10.1063/1.5094284
Published by AIP Publishing. 978-0-7354-1814-1/$30.00

was known may prevent the neural network from further training, many researchers develop the variant of the gradient-based algorithm. The new variant usually comes with an idea to correct the weaknesses of the previous methods. On the other hand, prior experiences show that there is no single optimization method has the best performance in all cases. Therefore, careful analysis is needed to get the right combination of CNN architecture and optimization algorithm to produce the best result.

By utilizing the ability of CNN to process plant digital images as the raw data, we have developed a non-destructive photosynthetic pigments prediction. It has been shown that CNN can be used to predict photosynthetic pigments content in a plant leaf [5]. A multispectral digital image which was consist of 10 channels was used as the input. Three CNN architectures, i.e., ShallowNet, LeNet and AlexNet were compared through 108 experiments. The LeNet architecture was proven slightly superior compared to the other two architectures. Hence, in this research, an analysis of the best optimization method to improve the previous result was conducted. Seven gradient descent-based method were implemented into previous CNN architectures and Mean Square Error (MSE) was used as the performance indicator.

GRADIENT DESCENT OPTIMIZATION

Gradient descent is a well-known technique for finding a local minimum value of a function. In CNN and other neural network-based algorithms, gradient descent is used to minimize the error function during the training process and then updating the internal parameters. In general, an optimization technique is categorized as first-order optimization algorithm and second-order optimization algorithm. Gradient descent belongs to the first order optimization algorithm class. The first order derivative provides information about the direction of the error function at a specific point, whether it is increasing or decreasing. The information is used to guide the error function to move downward until reaching the local minimum [6]. Batch gradient descent is the standard technique that is proved to be inefficient. In order to produce one update for the internal parameters, it computes the gradient of the entire training data (which is usually very large). It makes the updating process run very slow. With the aim of overcoming the weaknesses of such standard method, several algorithms were developed as follow:

1. **Stochastic Gradient Descent (SGD).** The modification is made to the amount of the data being used to update the internal parameters (θ). Rather than includes the entire training data, SGD uses a subset of training data $(x^{(i)}; y^{(i)})$ gradually. It calculates one parameters update based on one small subset of the training data. It will re-computes the gradients for other subsets and perform next parameter update. Each subset i is chosen randomly. Therefore SGD usually runs faster and could perform online learning. Parameters update at time $t + 1$ is computed using equation (1), η is the learning rate and J is the error function [7]. The frequent updates in SGD can provide the possibility to discover new and better local minima. However, the impact of the frequent updates is the difficulty in acquiring convergence.

$$\theta_{t+1} = \theta_t - \eta . \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{1}$$

2. **Adaptive Gradient (Adagrad).** Stochastic Gradient Descent uses the same learning rate (η) for all parameters updates. Adagrad modifies this process by allowing the learning rate to be changeable at each time step regarding the importance of the parameters being update. Significant updates were performed for the infrequent parameters, whereas small updates were performed for the frequent parameters. This mechanism will allow each parameter to discover its optimal value independently and gain the increase of overall accuracy. Parameters update is compute using equation (2), g_t is the gradient of the parameters on the error function at a time t and G_t is the diagonal element of the $\Sigma_{\tau=1}^{t} g_\tau . g_\tau^T$ matrix. Each diagonal element represents the sum of squares of each parameters gradient up to time step [8].

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} . g_t \tag{2}$$

3. **Adaptive Delta (Adadelta).** Adadelta modifies the Adagrad formula to reduce its monotonically decreasing learning rate. In Adagrad, the squared gradients from each epoch were accumulated, therefore the denominator value grows with the training process such that the learning rate on each dimension is shrinking. The idea of Adadelta to overcome such problem is accumulating only the last w squared gradients [9]. For efficiency reason, those accumulation is implemented as a decaying running average of the squared gradients as equation (3) with β is decay constant. The parameter update at time step $t + 1$ is computed using equation (4) and (5).

$$RMS[g]_t = E[g^2]_t = \beta.E[g^2]_{t+1} + (1-\beta).g_t^2 \tag{3}$$

$$\Delta\theta_t = -\frac{RMS|\Delta\theta|_{t+1}}{RMS|g|_t}.g_t \tag{4}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t \tag{5}$$

4. **Root Mean Square Propagation (RMSProp)** RMSProp was developed to resolve the Adagrad problem, just like Adadelta does. It divides the learning rate by an a running average of squared gradients ($E[g^2]_t$) which is exponentially decaying [10]. The parameter update at time step $t+1$ is computed using equation (6) dan (7).

$$E[g^2]_t = 0.9E[g^2]_{t+1} + 0.1g_t^2 \tag{6}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}}.g_t \tag{7}$$

5. **Adaptive Momentum (Adam)** Adam computes adaptive learning rates for each parameter. Just like Adadelta and RMSProp, Adam store squared gradients from the bias-corrected past epoch ($\hat{v}_t$). As an addition, Adam also stores the bias-corrected past average gradient ($\hat{m}_t$). Both values (v_t and m_t) estimate the decaying first moment and the second moment of the gradients as computed by equation (8)-(11), β is the decay constant [11]. The parameter update at time step $t+1$ is computed using equation [12].

$$m_t = \beta_1.m_{t-1} + (1-\beta_1).g_t \tag{8}$$

$$\hat{m}_t = m_t/(1-\beta_1^t) \tag{9}$$

$$v_t = \beta_2.v_{t-1} + (1-\beta_2).g_t \tag{10}$$

$$\hat{v}_t = v_t/(1-\beta_2^t) \tag{11}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_{tt} + \varepsilon}}.\hat{m}_t \tag{12}$$

6. **Adaptive Max Pooling (Adamax)** Adamax is a development of Adam. The modification is in the use of infinity norm (u_t). It was shown that v_t value in Adam with ℓ_∞ will converges to more stable value [11]. The parameter update at time step $t+1$ is then computed using equation (13) and (14).

$$\theta_{t+1} = \theta_t - \frac{\eta}{u_t}.\hat{m}_t \tag{13}$$

$$u_t = \beta_2^\infty.v_{t-1} + (1-\beta_2^\infty).|g_t|^\infty = max(\beta_2.v_{t-1}, |g_t|) \tag{14}$$

7. **Nesterov Adaptive Momentum (Nadam)** Nadam is the combination of Adam and Nesterov Accelerated Gradient (NAG). It is incorporating Nesterov momentum into Adam [12]. Since Nesterov momentum could achieve better result than classical momentum, it is hypothesized that Nadam will be better than Adam. The parameter update at time step $t+1$ is computed using equation (15).

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_{tt} + \varepsilon}}.(\beta_1.\hat{m}_t + \frac{(1-\beta_1).g_t}{1-\beta_1^t}) \tag{15}$$

MATERIAL AND METHOD

As much as 391 leaves multispectral images were produced, 80% was reserved as training data, and the rest was reserved as test data. Multispectral image acquisition was done at optic laboratory Ma Chung Research Center for Photosynthetic Pigments using Pcopixelfly 14 bit CCD camera with Thorlabs visible bandpass filter. Each multispectral image was consist of 10 channel i.e., 350, 400, 450, 500, 550, 600, 650, 700, 750 and 800 nm. Those images were then become the input of the system whereas the output were pigments content. The content of three main photosynthetic pigments i.e., chlorophyll, carotenoid, and anthocyanin were measured using non-destructive spectrometer (Ocean Optic USB-4000). The measurement was conducted along with the multispectral image acquisition. A leaf sample was exposed to light from the tungsten halogen using a probe. The probe captured the reflected light and sending it to the spectrometer. The spectrometer measures the intensity and sends it to the computer for the quantification and visualization.

Three CNN architectures were used in the experiment. In order to acquire enough information of the optimizers performance, the three architectures were selected with different level of complexity. Each of the architecture was experimented using 7 gradient descent-based optimizers (as explained in the previous section), 2 options of the data batch size (30 and 60), and 3 options of the epoch (15, 30, and 45). Therefore the total number of the experiment is 126. Details of the architectures are as follows:

1. **ShallowNet**, the architecture consists of 1 convolution layer with 32 filters in size 3x3 and 1 output layer with 5 nodes. Each node in the output utilizes LeakyRelu activation function and represents the pigment indices (Fig. 1). This architecture is the simplest one of the three architectures.

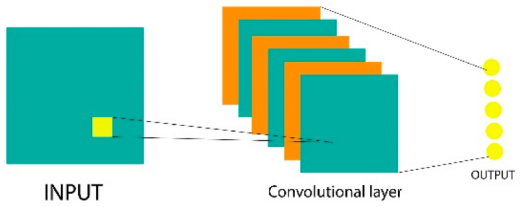


FIGURE 1: ShallowNet architecture

2. **LeNet**, the architecture consists of 2 convolution layers and 2 fully connected layers. The first convolution layer consists of 6 filters in size 3x3 with max pooling in size 2x2, and the second convolutional layer consists of 16 filters in size 5x5 with max pooling in size 2x2. The first fully connected layer consist of 120 nodes and the second fully connected layer consist of 84 nodes. Each node in the output utilizes LeakyRelu activation function. Figure 2 depicts the LeNet architecture.

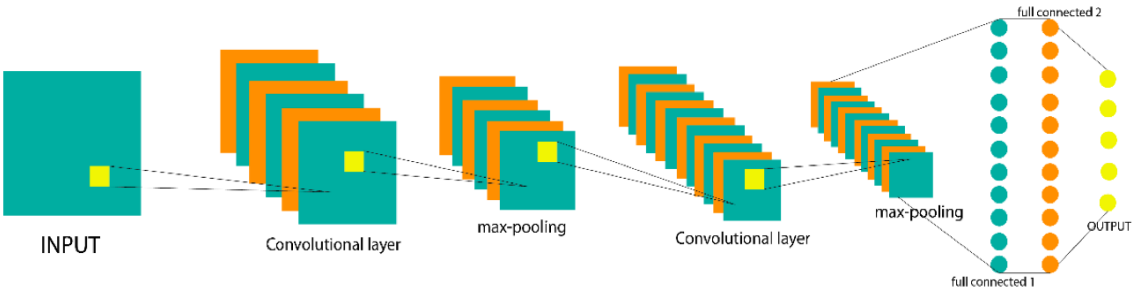


FIGURE 2: LeNet architecture

3. **AlexNet**, the architecture consists of 5 convolution layers and 3 fully connected layers. Detail of the design can be seen in Table 1 and Fig. 3 depicts the AlexNet architecture.

TABLE 1: Detail of the AlexNet architecture

Hidden Layer		Design
Convolution	1	96 filters in size 11x11 with max pooling in size 3x3
	2	256 filters in size 5x5 with max pooling in size 3x3
	3	384 filters in size 3x3 without pooling in size 3x3
	4	384 filters in size 3x3 without pooling
	5	256 filters in size 3x3 with max pooling in size 3x3
Fully Connected	1	4096 nodes with LeakyRelu activation function
	2	4096 nodes with LeakyRelu activation function
	3	100 nodes with LeakyRelu activation function

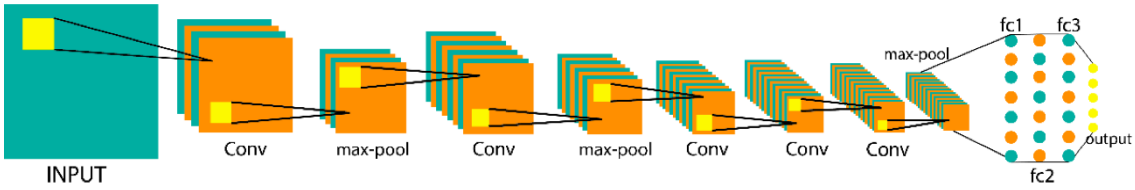


FIGURE 3: AlexNet architecture

RESULT AND DISCUSSION

Table 2, 3 and 4 provide the summary of the experiment results. The result of the experiments using batch size 60 was not shown because it was found that almost all experiments using the batch size 30 produce smaller MSE than using the batch size 60. Therefore, the result of the experiments using batch size 60 no longer in the discussion. From those tables, it could be inferred that in sample MSE for all CNN architecture being used is slightly smaller than out sample MSE. There is also no indication of overfitting occurrence, means that all architecture perform well in learning the multispectral image-pigment content relationship. The smallest MSE was obtained at LeNet architecture, i.e., 0.003 for the pigments content range -0.2 up to 2.2. In ShallowNet and LeNet, Adam has successfully trained the network, whereas in AlexNet, Adadelta was the most successful optimizer (notice the grayed-out cell).

TABLE 2: Mean Square Error (MSE) comparison of the seven optimization methods on ShallowNet

Epoch	Batch Size = 30													
	In Sample							Out Sample						
	SGD	Adagrad	Adadelta	RMSProp	Adam	Adamax	Nadam	SGD	Adagrad	Adadelta	RMSProp	Adam	Adamax	Nadam
15	0.063	0.014	0.516	0.021	0.007	0.563	0.547	0.055	0.014	0.533	0.024	0.009	0.583	0.566
30	0.007	0.342	0.310	0.016	0.005	0.378	0.319	0.012	0.347	0.312	0.019	0.007	0.386	0.322
45	0.052	0.262	0.007	0.047	0.129	0.021	0.223	0.051	0.259	0.012	0.054	0.113	0.032	0.215

TABLE 3: Mean Square Error (MSE) comparison of the seven optimization methods on LeNet

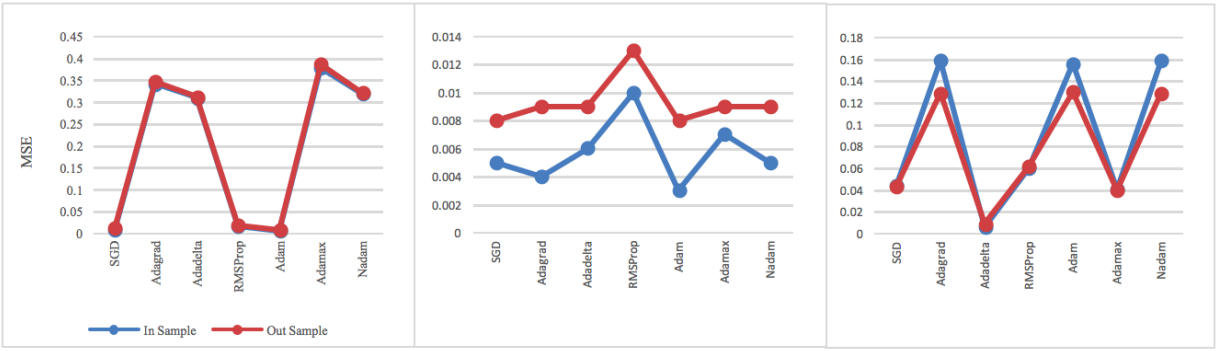
Epoch	Batch Size = 30													
	In Sample							Out Sample						
	SGD	Adagrad	Adadelta	RMSProp	Adam	Adamax	Nadam	SGD	Adagrad	Adadelta	RMSProp	Adam	Adamax	Nadam
15	0.010	0.010	0.007	0.012	0.015	0.007	0.008	0.013	0.012	0.009	0.013	0.016	0.010	0.010
30	0.009	0.005	0.009	0.007	0.009	0.007	0.006	0.011	0.008	0.011	0.010	0.009	0.010	0.010
45	0.005	0.004	0.006	0.010	0.003	0.007	0.005	0.008	0.009	0.009	0.013	0.008	0.009	0.009

Figure 4 depicts the performance comparison of all optimizer in each architecture. Figure 4(a) shows that in ShallowNet, SGD, RMSProp, and Adam provide the smallest MSE. Figure 4(b) shows that in LeNet, Adam provides

TABLE 4: Mean Square Error (MSE) comparison of the seven optimization methods on AlexNet

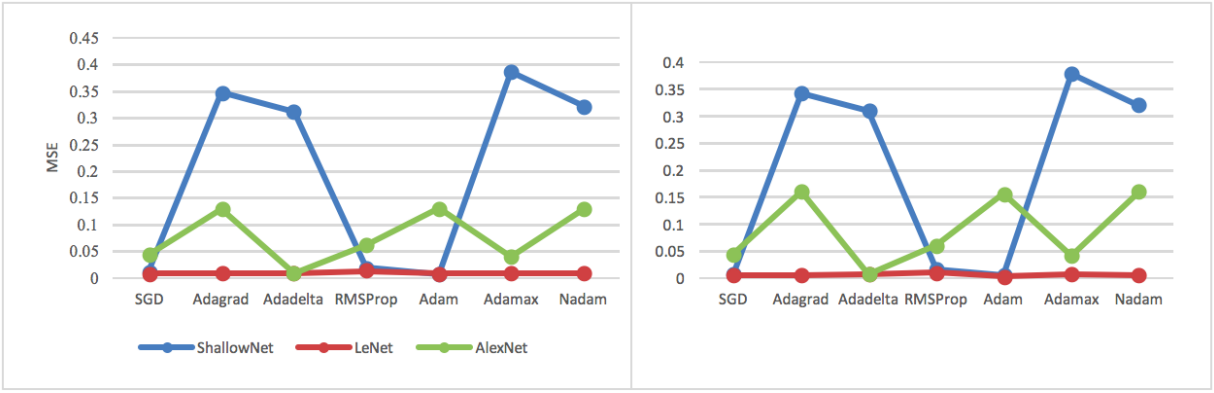
Epoch	Batch Size = 30													
	In Sample							Out Sample						
	SGD	Adagrad	Adadelta	RMSProp	Adam	Adamax	Nadam	SGD	Adagrad	Adadelta	RMSProp	Adam	Adamax	Nadam
15	0.158	0.160	0.009	0.132	0.009	0.161	0.157	0.129	0.132	0.012	0.113	0.012	0.131	0.129
30	0.156	0.160	0.009	0.021	0.156	0.158	0.156	0.129	0.129	0.010	0.023	0.131	0.129	0.131
45	0.044	0.159	0.006	0.060	0.156	0.041	0.159	0.043	0.129	0.009	0.062	0.130	0.040	0.129

the smallest MSE whereas RMSProp provides the biggest MSE. Besides Adam, SGD and Adagrad are also potential to provide small MSE. Figure 4(c) shows that in AlexNet, Adadelta provides the smallest MSE whereas Adagrad, Adam, and Nadam provide the biggest MSE. Hence, it could be inferred that each optimizer shows different performance across different CNN architectures. This fact complies the general theory which is stated that there is no single optimizer will perform the best result for all cases [6].



(a) (b) (c)

FIGURE 4: Mean Square Error (MSE) comparison for all optimizers in each CNN architecture, (a) ShallowNet; (b) LeNet; (c) AlexNet



(a) (b)

FIGURE 5: Mean Square Error (MSE) comparison for all optimizers in all CNN architecture, (a) In sample MSE; (b) Out sample MSE

Figure 5 depicts head to head comparison of all optimizers performance in the three architectures. Figure 5(a) is the comparison for in sample MSE and Fig. 5(b) is the comparison of out sample MSE. It is shown that among the three architectures, all optimizer consistently provide small MSE on LeNet. Therefore, it is concluded that LeNet is

the most suitable architecture for representing the multispectral image-pigment content relationship. As shown in Fig. 4 before, Adam should be considered to be used as the optimizer for LeNet.

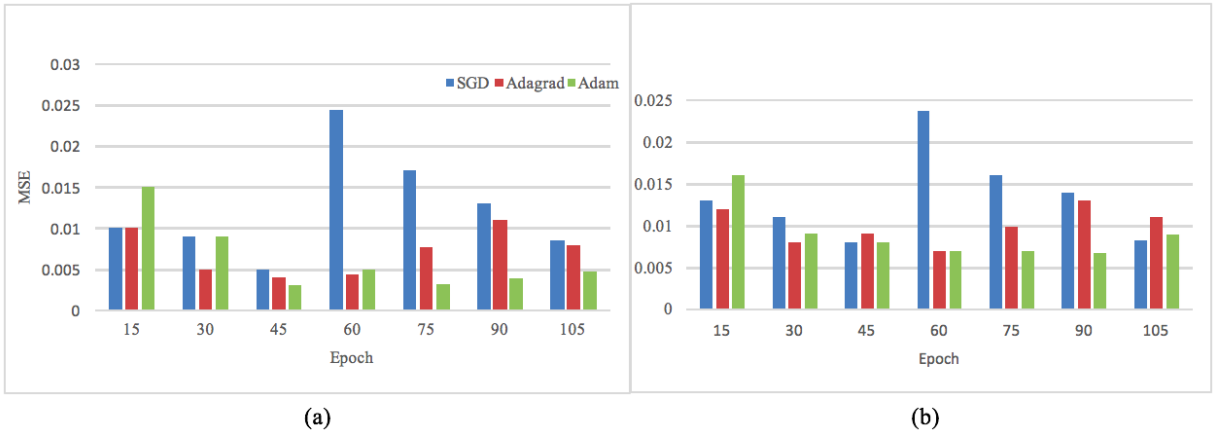


FIGURE 6: Mean Square Error (MSE) comparison for SGD, Adagrad and Adam in LeNet with increasing number of the epoch, (a) In sample MSE; (b) Out sample MSE

Figure 6 depicts the further analysis of the potential optimizers for LeNet, i.e., SGD and Adagrad. A number of the epoch was added to describe its behavior against the possibility of overtraining. As much as 60, 75, 90 and 105 epoch was experimented to train the CNN with SGD, Adagrad, and Adam optimizer. It was shown that all optimizer did not need much epoch to reach the minimum MSE, indicate that all optimizers could converge very well. Nevertheless, among the three optimizers, Adam is the most stable one. After the 45th epoch, Adam has the ability to maintain the small MSE. On the other hand, both SGD and Adagrad failed to do so. The MSE of SGD and Adagrad have a tendency to increase again after the 45th epoch. Compared to Adagrad, such tendency is worse in SGD. Therefore, Adam is the best choice for LeNet architecture.

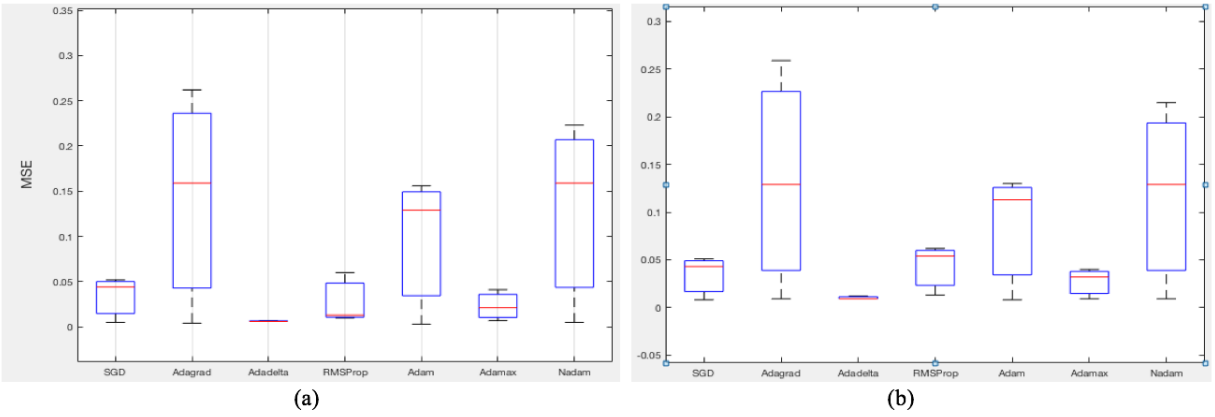


FIGURE 7: Boxplot comparison of MSE result from seven optimizer in three different CNN architectures , (a) In Sample MSE; (b) Out Sample MSE

From the previous discussion, it is revealed that the combination of CNN architecture and optimization method is unique for each case. However, from the bunch of our experiment data, it is possible to describe in general the behavior of each optimizer against different CNN architectures. Figure 7 depicts the boxplot of MSE provides by each optimizer in all CNN architectures. It is shown that Adam is not the best optimizer among the seven optimizers, it may result in the smallest MSE, but only with the LeNet architecture. In other architectures, Adam could perform greater values. The Adadelta and Adamax optimizers seem to work better with different CNN architectures. Therefore, if there

are limited resources so that several experiments cannot be carried out with different CNN architectures, Adadelta and Adamax are wise choices to minimize risk.

CONCLUSIONS

It is shown from the experiment that all optimizer show different performance across different CNN architectures. However, all optimizer perform consistent small MSE on LeNet architecture. Among 7 optimizers, Adam provides the smallest MSE on the LeNet architecture. Adam also shows the ability to maintain the MSE still low in the next epoch after it reaches the minimum on a particular epoch. On the other hand, the SGD and Adagrad were failed. If there are limited resources to run several architectures, Adadelta and Adamax should consider being used to provide minimal risk. It is shown that the Adadelta and Adamax optimizers consistently produce a small MSE across different architectures.

ACKNOWLEDGMENTS

This research is partially funded by the Ministry of Research, Technology, and Higher Education of the Republic of Indonesia through a national research grant program year 2018. The authors also acknowledge Machung Research Center for Photosynthetic Pigments (MRCPP) for the support of laboratory facilities.

REFERENCES

[1] K. Fukushima, Neural Networks **1(3)**, 119–130 (1988).
[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Shock Compression of Condensed Matter-2001*, IEEE Conference Proceedings (Institute of Electrical and Electronics Engineers, Melville, NY, 1998), pp. 2278–2342.
[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural network," in *International Conference on Neural Information Processing Systems* (Lake Tahoe, Nevada, 2012), pp. 1097–1105.
[4] A. Bhandare, M. Bhide, P. Gokhale, and R. Chandavarkar, International Journal of Computer Science and Information Technologies **7(5)**, 2206–2215 (2016).
[5] K. R. Prilianti, I. C. Onggara, M. A. S. Adhiwibawa, T. H. P. Brotosudarmo, S. Anam, and A. Suryanto, "Multispectral imaging and convolutional neural network for photosynthetic pigments prediction," in *2018 15th International Conference on Electrical Engineering, Computer Science and Informatics*, IEEE Conference Proceedings (Institute of Electrical and Electronics Engineers, Malang, Indonesia, 2018), pp. 749–754.
[6] S. Ruder, An overview of gradient descent optimization algorithms, ArXiv preprint arXiv:1609.04747 (2016).
[7] P. Toulis, T. Horel, and E. M. Airoldi, Stable robbins-monro approximations through stochastic proximal updates, ArXiv preprint arXiv:1510.00967v3 (2018).
[8] J. Duchi, E. Hazan, and Y. Singer, Journal of Machine Learning Research **12**, 2121–2159 (2011).
[9] M. D. Zeiler, Adadelta: An adaptive learning rate method, ArXiv preprint arXiv:1212.5701 (2012).
[10] G. Hinton, N. Srivastava, and K. Swersky, Overview of mini batch gradient descent, Lecture 6a. Class Lecture, Computer Science Department, University of Toronto (2015).
[11] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations* (San Diego, 2015).
[12] T. Dozat, "Incorporating nesterov momentum into adam," in *International Conference on Learning Representations Workshop* (San Juan, Puerto Rico, 2016), pp. 2013–2016.