Comparative Study of Surrogate Techniques for CNN Hyperparameter Optimization

Nurshazlyn Mohd Aszemi, Nordin Zakaria, P. D. D. Dominic

Universiti Teknologi PETRONAS, Seri Iskandar, Perak, Malaysia

Corresponding author: Nurshazlyn Mohd Aszemi, Email: nurshazlyn_17007352@utp.edu.my

Optimizing hyper parameters in Convolutional Neural networks is a tedious process for many researchers and practitioners. It requires a high degree of expertise or experience to optimise the hyper parameters, and manual optimisation is likely to be biased. To date, methods or approaches to automate hyper parameter optimization include grid search, random search, and Genetic Algorithms (GAs). However, evaluating large number of sample points in the hyperparameter configuration space, as is typically required by these methods,isa computationally expensive process. Hence, the objective of this paper is to explore regression as a surrogate technique in CNN hyperparameter optimisation. Performance in terms of accuracy, error rate, training time and coefficient of determination (R2) are evaluated and recorded. Although there is no significant performance difference between the resulting optimized Deep Learning and state-of-the-art on CIFAR-10 datasets, using regression as a surrogate technique for CNN hyperparameter optimization contributes to minimising the time taken for the optimization process, a benefit which has not been fully exploredin the literature to the best of the author's knowledge.

Keywords: Convolutional Neural Network, Regression, Hyperparameter, Optimization, Deep Learning, Machine Learning.

2021. In Rahul Srivastava & Aditya Kumar Singh Pundir (eds.), *New Frontiers in Communication and Intelligent Systems*, 463–473. Computing & Intelligent Systems, SCRS, India. https://doi.org/10.52458/978-81-95502-00-4-48

1 Introduction

Designing Deep Learning (DL) architectures and optimising features such as hyperparameters often requires human expertise. Currently, successful implementation of DL has been reported in different applications such as computer vision, image and handwriting classification, object recognition and others. Many researchers are inclined to DL as the accuracy rate is far better than classical machine learning. Most successful networks are deep in scale in order to deal with big data that are growing more and more over the year. One of the examples of deep learning that is getting more attention among researchers is the convolutional neural network (CNN). CNN is the study of image recognition and classification in the neural network. Most popular models have risen from ILSVRC image competition such as LeNet[1], AlexNetkriz[2],Deep Residual Network[3], Inception[4]Inception, and latest achieving state-of-the-art accuracy with CIFAR-10 datasets is Big Transfer (BiT) [5] and XNAS [6]. However, different deep learning applications require a different set of hyperparameters in each neural network architecture. Identifying a good model of architectures and hyperparameters is time-consuming and exhaustive. Moreover, training every possible combination can become infeasible and computationally expensive. Therefore, the need for an automated and structured way of searching hyperparameter space is critical.

Numerous works have been done in optimising hyperparameters [7]–[10]. Unfortunately, [11]claims that none of the approaches considers the impact of setting up the hyperparameter, which in assumptions that: (1) Hyperparameter setting does not matter; however, selecting among default implementations is sufficient and (2) hyperparameter value may have a significant impact on performance and should constantly be optimised. [11] reported that there is a significant improvement after running hyperparameter optimisation than default hyperparameter. Although hyperparameter optimisation has been successfully implemented for deep learning architectures such as CNN, the optimisation process is still expensive as it requires a large amount of training time which can take days to produce the end results.

İnterestingly,[8]managed to speed up the hyperparameter optimisation of deep neural networks with simple three layers (referring to the number of convolutional layers) from 40 to 18 hours. This was done using predictive models to predict the validation performance and terminating the run when it cannot achieve the best performance. Recent research by [12] on his paper on accelerating neural architecture search using performance prediction have surpassthe result in [8], using support vector regression (SVR) and reinforcement learning.Further, the use of Graphics Processing Unit (GPU) to perform training can speed up the whole process from months, weeks and days to only a couple of hours.

In this paper, we seek to further explore the use of regression to speed up hyperparameter optimization time. Specifically, we experiment with different kernel for SVR. We assume not just one or two hyperparameters in CNN but as many as possible. Additionally, both network architecture and hyperparameters are considered, and CIFAR10 is used as our primary dataset.

2 Related work

2.1 Hyperparameter Optimization

Hyperparameter optimisation is an automatic optimisation of the hyperparameters in deep learning by using a set of algorithms. In other words, hyperparameter optimisation is a problem in finding a set of hyperparameters model that gives the lowest error rate and high accuracy in classification [13].

Hyperparameters are a parameter that is external to deep learning architectures and often specified by the researcher, such as learning rate, batch size, number of layers in network architectures, etc.

In the early stage of hyperparameter optimisation, researchers applied manual and grid search as a trial and error method for every hyperparameter setting on a specific range of values[14]–[16].However, if one fails, the rest of the jobs will fail accordingly. Moreover, four hyperparameters will become impractical as the number of evaluation functions will increase with adding parameters due to dimensionality limitations [17]. The random search method samples the hyperparameter space 'randomly'. Based on[18], the random search has more benefits than grid search regarding the application, which can still use even the computer cluster fail. It allows practitioners to change the 'resolution' on the go, and it is feasible to add new trials to the set or even ignore the fail test. Simultaneously, the random search method can stop any time, and it will form a complete experiment it can be carried out synchronously [19]. Furthermore, a new trial can be added to the experiment without jeopardising if more computers become are available [10].

Another latest development in hyperparameter tuning is using Bayesian optimization [20]. It uses distribution over functions which are known as Gaussian Process. To train using the Gaussian Process, fitting it to given data is essential as it will generate function strictly to observe data. Widespread implementation of Bayesian optimisation includes spearmint that uses the Gaussian process [21]. However, [8]claims that the Bayesian optimisation method is limited, as it works on high dimensional hyperparameters andhas become very computationally expensive. Therefore, it has poor performance.

2.2 Regression Approach

Recent research by[12]shows that the use of simple regression models using support vector regression (SVR) in predicting the final performance of the neural network with reinforcement learning can decrease the training process time of optimising the hyperparameters. The prediction framework then functions as an early stopping method in hyperparameter optimisation that determines which hyperparameter models should be trained further. This method overpasses the research done by [8]that uses predicting learning curves model in terminating bad runs when optimising the hyperparameter on CNN. Both kinds of research show the possibilities of using prediction methods with the neural network while optimising the hyperparameter can achieve higher accuracy, less error and reduce the training process time. However, the gaps between both research that simply, only optimising network architectures in CNN are fully considered. This is because the growing research for more in-depth layers of CNN architectures can help them learn more image classification features such as edges, shapes, and high order features. Unfortunately, [11]claim that none of the approaches considers the impact of optimising the hyperparameter in which is the hyperparameter setting that does not matter and selecting among default hyperparameter is sufficient. The research results reported a significant improvement not only in optimising the network architecture but also in running hyperparameter optimisation than default hyperparameter[11]. Hence, the optimisation of both network architectures and hyperparameters while exploring the viability of the regression approach as a surrogate technique in CNN hyperparameter optimisation are considered in this research.

3 Methods

We experiment with small CNN architecture with randomised search optimisation. We use a random hyperparameter search algorithm as a partially trained model to generate random individual solutions at any point of the search space. These models then will be run with regression for validation experiments from [12].

3.1 CNN Hyperparameter Optimisation with Partially Trained Models

The experiment is performed on CIFAR-10 datasets using the random search method. The model is training with three convolutional layers. First, the height, width, channels and outputs of 32x32x3 are fixed, with ten-fold cross-validation. Then, twenty-four iterations within epoch will run on accuracy check. Finally, the epoch number is fixed on 50 to store time-series validation accuracy. The model then is trained with the hyperparameter configurations based on Table 1 and Table 2. Note that the range values randomly. Then, the hyperparameter evaluations are being stored in training logs.

3.2 Support Vector Regression

We then compare the viability of the regression approach on our partially trained models using support vector regression method. Support vector regression (RSVR) derives from the support vector machine (SVM) that uses a linear separator [22]. This line will separate two independent groups of data. Hence, adding a new data point to the graph will classify this line accordingly with a label [22]. However, in the case of deep learning, we are not predicting a class label, and we do not need to classify. Instead, we predict the next value in a series by using regression [23]. RSVR is a type of SVM that uses the space between data points as a margin of error and predicts the most likely next point in a data set [24]. Our method is similar to [12]. However, we experiment with different kernels such as linear, polynomial and radial basis function kernels (RBF). These kernels function as transforming the data into the required form. Even though [12]have proven the RBF performed well, we will conduct a comparative study by performing different kernel tests. We train our R_{SVR} on 10-fold cross-validation instead of 3 to minimise the running time. Furthermore, we train our R_{SVR} on 1000 randomly sampled neural network configurations for all experiments.

We compare different kernels to reassure the regression method used[12]. We apply the early stopping method to speed up our partially trained data process. We will remove the early stopping method for the final experiment and perform it in R_{SVR} to speed up the process. We perform this experiment using Linear, Polynomial and Radial Basis Function (RBF) kernels. The kernels are essential in performing an R_{SVR} as eachdefines a way of computing the product of two vectors, x and y. We split partially trained data into testing and training data. We then fit the data into three different kernels and analyse it using accuracy as measurement. Finally, we report the R^2 by plotting the chosen kernels with true and predicted values.

3.3 Datasets and Training Settings

Datasets: CIFAR10 will be used as a dataset as it is a subset of the 80-million tiny image databases [25]. There are 50,000 images for training and 10,000 images for testing. All of them are 32 × 32 RGB images. CIFAR10 contains ten basic categories, and both training and testing data are uniformly distributed over these categories. Additionally, 10,000 images have been left from the training set for validation to avoid using the testing data. This will protect the performance of the dataset from overfitting.

Platform: Searching for the best combination of hyperparameters requires computational resources. Fortunately, Nvidia Tesla K80 is a supercomputer used as a computing platform in this research. It can dramatically lower data centre costs by delivering exceptional performance with fewer and more powerful servers. TensorFlow is a framework that will represent computations as graphs, allowing more straightforward computation and analysis of these models and utilising multi-dimensional arrays called Tensors and by computing these graphs in sessions. Tensor Flow will implement Keras as the backend to allow easy and fast prototyping through user-friendliness, modularity, and extensibility. It

supports the convolutional neural networks and runs smoothly in CPU and GPU. The programs are coded in Python 3.7 along with their packages.

CNN architecture: The Convolutional Neural Network (CNN) architecture design followed [26] and[27] with three CNN layers and two fully connected layers. Batch normalisation layers[26], [27] and dropout layers are added as recommended by[28]. Batch normalisation helps standardise the input (images) volume before passing to the subsequent layers, while dropout is a regularisation method that prevents the model from overfitting [16]. Regularisation is a method that ensures the models can make correct classifications to the data that are not trained (ability to generalise). Based on [28], if no regularisation method is applied, the models will not be able to generalise the data, and the data have been trained too well (overfitting). The CNN architectures pattern is as follows: CONV is the convolutional layer that is the core of the CNN architecture. BN is the batch normalisation layer that helps to standardise the input (images) volume before passing to the subsequent layers. POOL is a max-pooling layer that reduces or decreases the input volume's size (width and height). DO is a dropout layer that helps the models generalise by making multiple or redundant nodes of layer activated when presenting with similar inputs. FC is fully connected layers that process the given input (image) classification results. The patterns of CNN architecture will be as:

INPUT => [CONV => BN] *3 => POOL => DO => FC => FC

Hyperparameters: Hyperparameters in deep learning can be divided into two types. One type is those associated with the learning algorithms, such as determining what learning rate is appropriate, after how many iterations or epochs for each training, etc. The other type of hyperparameter is related to how we design deep neural networks. For example, how many layers we need for our network, how many filters in given convolutional layers needs, etc. Choosing different values and setting these hyperparameters properly is often critical for reaching the full potential of the deep neural network chosen or designed, consequently influencing the quality of the produced results. We list out hyperparameters that will be utilised throughout the training process. Table 1 displays the state space of the randomised search algorithm.

Table 1. Range of hyperparameter use in randomised search optimisation

Hyperparameter	Range	Abbreviation
Learning rate	[0.001, 0.002, 0.003, 0.0015]	learning_rates
Batch size	[32, 64, 128]	batch_size
Momentum	[0.9 ,0.95 , 0.99]	momentum
Optimizer	[adam, rmsprop, nesterov]	optimizer

Network architectures. Table 2 displays the network architecture varying number of filters, kernel size, hidden layer and activation unit.

Table 2. Range of network architectures used in randomised search optimisation

Layer Type	Network Architecture	Range	Abbreviation
1st Convolutional Layer	Number of filters	[16, 32, 64, 96]	Filters1
	Kernel Size	[3, 4, 5]	Ksize1
	Activation	[relu, lrelu, elu]	Activation_unit
2nd Convolutional Layer	Number of filters	[64, 96, 128]	Filters2
	Kernel Size	[3, 4, 5]	Ksize2
	Activation	[relu, lrelu, elu]	Activation_unit
3rd Convolutional Layer	Number of filters	[32, 64, 128]	Filters3
	Kernel Size	[3, 4, 5]	Ksize3
	Activation	[relu, lrelu, elu]	Activation_unit
1st Fully Connected Layer (FC)	Hidden Layer	[60, 100, 125]	Full_hidd1

2nd Fully Connected Layer (FC)	Hidden Layer	[80, 100, 125]	Full_hidd2

3.4 Measuring the hyperparameter optimisation problem in CNN

Accuracy: The accuracy is measured to accurately assess the optimisation method's ability in hyperparameter optimisation to classify the images in the CIFAR10 datasets. Image classification involves a process whereby giving a set of images n to CNN, thus how the hyperparameter model can produce the correct images [17]. A model performance P can be determined after the model undergoes the training process, and accuracy A is calculated in the form of a percentage.

$$N \rightarrow P = A\,(\%) \qquad (1)$$

Error Rate / Loss: Another way of measuring the performance of the hyperparameter model is by using the error rate. The error rate is also known as loss. When given a set of images n to CNN, the error rate is how the hyperparameter model can produce the incorrect images [29]. The error rate can be calculated through loss functions. Loss functions for classification can be several. However, determining the datasets used is helpful to know which loss functions can be used. [30] and [28] both agreed that the most common loss function used in machine learning is hinge loss. Hinge loss is inspired by linear Support Vector Machines (SVM) or known as maximum-margin classification [28]. Supposed, we were to classify correct images and incorrect images. We need to design matrix, M, where each row is an image that we want to label a correct image. Accessing the i image inside M can be viewed as syntax M_i. V vector will contain the "ground truth labels" or original image class labels that we hope the models are able to predict correctly. Hence, M_i enables us to access images while V_i enables us to access the labels. ($i \neq k$) is a sum product of incorrect classes by hinge loss function, which then will compare to the output of score function s. Max operation is applied to retain values at zero as we want the negative values. However, loss functions that are used in deep learning (CNN) use cross-entropy loss where probabilities of predicting correct and incorrect images are involved. Hinge loss gives the output in terms of margin, while cross-entropy (inherited from logistic regression classifiers known as softmax) gives the output in terms of probabilities [28]. Probabilities are more accessible to interpret than margins. The involvement of negative log-likelihood in cross-entropy loss is necessary when dealing with the product of probabilities and converting it to the log of probabilities. Hence our error rate was evaluated based on [28] and [30].

$$s = f(\mathcal{M}_i) \qquad (2)$$
$$s_k = f(\mathcal{M}_i)k \qquad (3)$$
$$\mathcal{L}_i = \sum_{k \neq v_i} \max(0, s_k - sv_i + 1) \qquad (4)$$

Our cross-entropy loss function can be defined as:

$$\mathcal{L}_i = -\log \mathcal{P}\,(\mathcal{V} = v_i \mid k = \kappa_i) \qquad (5)$$
$$s = f(\mathcal{M}_i) \qquad (6)$$
$$\mathcal{P}(\mathcal{V} = x \mid \mathrm{K} = \kappa_i) = \frac{e^{sv_i}}{\sum_k e^{s_k}} \qquad (7)$$
$$\mathcal{L}_i = -\log \frac{e^{sv_i}}{\sum_k e^{s_k}} \qquad (8)$$

Regression: The measurement of regression is using R-squared (R^2), which is a metric that describes the quantity of variance in the relationship between two or more variables in the data and it is also known as the coefficient of determination or how close the data can fit the regression line[31]. The model is considered fit if the differences between observed or actual values and predicted values are small and unbiased. For example, a linear model Y can be represented by paired variables of X_i and Y_i with e as a mean zero error, and B_0 and B_1 represent the parameters that minimised the sum of squared residuals between variable Y_i and the model of $B_0 + B_1X_i$. Hence, calculating the R^2 can be viewed as the

ratio of variations that are presented by the model over the total variations present in $\acute{Y}$. SSR represents the regression sum of squares which is given the total sum of squares (SST) minus the minimum sum of square errors (SSE). SSE is the measurement of the variability of Y that is not explained by the model, and SST is the total variation in the Y variable, which represents the variability present in the data. R^2 is determined by the range of 0 and 1 where $R^2 = 1$ explains all the variability Y of the data in the model while $R^2 = 0$ explain none of the variability Y of the data in the model, and any value larger than 0.5 is considered as a significant coefficient of determination (R^2)[31]. [32]and [33] use R^2 to analyse the performance of the regression method correlating to the given datasets, which are identical to this research. We then result our findings as R_{SVR} into tables and graphs.

$$\acute{Y} = \mathcal{B}_0 + B\mathcal{B}_i x \quad (9)$$

$$\gamma = \mathcal{B}_0 + \mathcal{B}_i x + e \quad (10)$$

$$\mathcal{R}^2 = \frac{SSR}{SST} = \frac{SST - SSE}{SSE} \quad (11)$$

$$SSR = SST - SSE \quad (12)$$

$$SSE = \sum_{i=1}^{n} (\gamma_i - \acute{Y}_i)^2 \quad (13)$$

$$SSE = \sum_{i=1}^{n} (\gamma_i - \acute{Y}_i)^2 \; where \; \gamma = \frac{1}{n}\sum_{i=1}^{n} \gamma_i \quad (14)$$

Training Time: We compute the training time as the number of times taken in seconds, minutes, hours, or days needed to run complete hyperparameter optimisation. The training time will be calculated when the optimisation of *n*-hyperparameter model start and ends.

4 Results and Discussion

4.1 CNN hyperparameter model validation

Table 3 shows the results obtained from running hyperparameter optimisation model using random search. An alternate number of hyperparameter models dataset (20-200) was set as an experiment to determine the effect size of hyperparameter models to the accuracy, loss, and training time. The best accuracy was obtained with a 200 hyperparameter model with 78% validation accuracy and 0.63 error rate in table 3. The results also showed that the more hyperparameter models datasets, the better the accuracy obtained. The search space with more hyperparameter models allows CNN to explore more possibilities of different hyperparameter model combinations, which can produce better accuracy. This hyperparameter optimisation using random search can outperform fixed hyperparameter that is tuned manually by machine learning practitioners, which is proven by [34], [35]. However, the training time or resource use increases as the hyperparameter model datasets increase (see table 3). This called for another solution to find a way to decrease the training time taken to achieve the best accuracy.

Table 3. Results of random search hyperparameter model in terms of training time, accuracy and error rate

Model	Training Time (s)	Accuracy	Loss
20	2876	73.73	0.75
40	5,411	74.55	0.74
60	6,448	74.54	0.73
80	8,828	73.15	0.77
100	12,816	75.41	0.70
120	14,619	75.70	0.69
140	13,237	77.85	0.63
160	15,441	74.44	0.74
180	21,229	75.91	0.69
200	18,882	78.02	0.63

We derive the hyperparameter models into a time-series dataset to implement a comparative study of R_{SVR} kernels. Figure 1 shows the time-series performance accuracy for 50 epochs within 200 hyperparameter models that have been found using random search hyperparameter. We can simplify the time series by presenting the best accuracy found between the search models. Nonetheless, by looking at figure 1, the accuracy in every epoch has fluctuated. We can infer that when the learning process is involved, the power of Graphics Processing Unit (GPU) play a significant role in making the training process run smoothly without interruption. Big data requires higher computational resources that involve multiple GPU or GPU high power. The higher the power of GPU, the more smooth the training process of the neural network. However, we did not cover this as part of our research due to our goal is to speed up the training process by regression approach.

4.2 Comparative study of the regression approach as surrogate techniques

We perform this experiment using Linear, Polynomial and Radial Basis Function (RBF) kernels with 1000 CNN hyperparameter models. The kernels are essential in performing as R_{SVR} as it is a way of computing the product of two vectors x and y. We split partially trained data into testing and training data. We then fit the data into three different kernels and analyse it using accuracy. We report the R^2 by plotting the chosen kernels with true value and predicted values as shown in table 4. It is proven that using Linear kernels is the most suitable kernels to be used in this partially trained data. The R^2 give a positive value of 0.8, which is closer to 1. This means R_{SVR} can be used in solving this set of image classification optimisation problems. R_{SVR} running time takes 1 min per prediction and a total of 30 minutes to run all the hyperparameter model predictions within the epoch 1 – 50. It is still showing a good correlation of a goodness fit. R-square allows us to evaluate the accuracy of a performance prediction model across the search space, which is essential because we do not want the performance prediction model to overestimate the performance of poorly performing hyperparameter optimisation nor underestimate the performance of well-performing hyperparameter optimisation. Having a predictor that works well in all parts of the search space is helpful for optimisation methods.

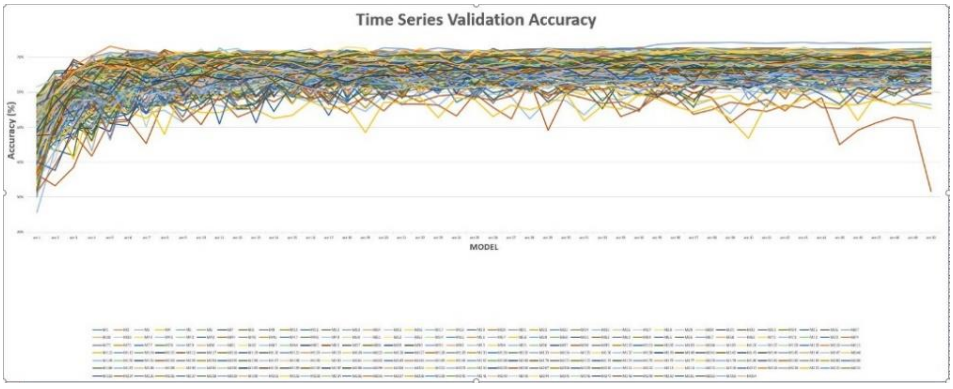

Fig. 1. Time series validation accuracy for 50 epochs of the 200 random search hyperparameter models

Table 4. Results of R^2 with different kernels of SVR compared with [12]

Regression	(R^2)
SVR (Linear Kernel)	**0.800**
SVR (Poly Kernel)	0.586
SVR (RBF Kernel)	0.653
SVR [12]	0.970

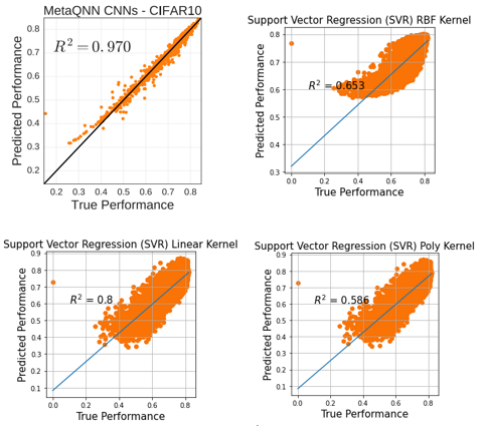

Fig. 1.R^2 graph analysis of predicted performance and true performance between Rsvr kernels of CNN hyperparameter optimisation using both network architectures and hyperparameters while comparing with[12]as only optimising their network architectures.

5 Conclusion

The method of speeding up the training process of CNN hyperparameter optimisation in CIFAR-10 datasets using R_{SVR} is explored in this paper. We validate the regression approach as a surrogate technique in optimising network architecture and hyperparameter in CNN using CIFAR-10 datasets. The results of the comparative study on R_{SVR} show that R_{SVR} has the potential to be used in speeding up hyperparameter optimisation. Comparative study was done on different R_{SVR} kernels. Further experiments will be conducted with other regression approaches such as Gradient Boosting, Random Forest and Ordinary Least Square (OLS) to optimise CNN hyperparameter with both network architectures and learning hyperparameters incorporated.

References

[1] Lecun, Y. et al. (1998). Gradient-Based Learning Applied to Document Recognition. In *Proceedings of the IEEE*, 86(11): 2278-2324.

[2] Krizhevsky, A. et al. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in neural information processing systems*, 1097–1105.

[3] He, K. et al. (2015). Deep Residual Learning for Image Recognition. *arXiv*: 1512.03385.

[4] Szegedyet, S. (2014). Going Deeper with Convolutions. *arXiv:* 1409.4842.

[5] Kolesnikov, A. et al. (2020). Big Transfer (BiT): General Visual Representation Learning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12350: 491–507.

[6] Nayman, N. et al. (2019). XNAS: Neural architecture search with expert advice. *Advances in Neural Information Processing Systems*.

[7] Snoek, J. et al. (2015). Scalable Bayesian Optimization Using Deep Neural Networks. *arXiv:* 1502.05700.

[8] Domhan, T. et al. (2015). Speeding up automatic hyperparameter optimisation of deep neural networks by extrapolation of learning curves. In the *Proceeding of the 24th International Conference on Artificial Intelligence,* 3460-3468.

[9] Eggensperger, K., Feurer, M. and Hutter, F. (2013). Towards an empirical foundation for assessing bayesianoptimisation of hyperparameters. *NIPS, BayesOpt workshop*, 1–5.

[10] Bergstra, J. et al. (2013). Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. *Icml*, 115–123.

[11] Sanders, S. and Giraud-Carrier, C. (2017). Informing the use of hyperparameter optimisation through metalearning. doi: 10.1109/ICDM.2017.137.

[12] Baker, B. et al. (2018). Accelerating neural architecture search using performance prediction. In *6th International Conference on Learning Representations,* 2.

[13] Wistuba, M., Schilling, N. and Schmidt-Thieme, L. (2016). Hyperparameter optimisation machines. doi: 10.1109/DSAA.2016.12.

[14] Larochelle, H. et al. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In *ACM International Conference Proceeding Series*, 227(2006): 473–480.

[15] Nair, V. and Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines Vinod. doi: 10.1123/jab.2016-0355.

[16] Sanchez, O. R. et al. (2021). Evaluating ML-based DDoS Detection with Grid Search Hyperparameter Optimization. In *Proceedings of the 2021 IEEE Conference on Network Softwarization: Accelerating Network Softwarization in the Cognitive Age,* 402–408.

[17] Dewancker, I., McCourt, M. and Clark, S. (2016). Bayesian Optimization for Machine Learning : A Practical Guidebook. *preprint ArXiv*.

[18] Bergstra, J. et al. (2012). Random Search for Hyper-Parameter Optimization Yoshua Bengio. http://scikit-learn.sourceforge.net.

[19] Bergstra, J. et al. (2011). Algorithms for hyper-parameter optimization. In *25th Annual Conference on Neural Information Processing Systems*, 1–9.

[20] Klein, A. et al. (2017). Learning Curve Prediction With Bayesian Neural Networks. *International Conference on Learning Representations*, Mcmc: 1–16.

[21] Snoek, J. et al. (2015). Scalable Bayesian Optimization Using Deep Neural Networks. doi: 10.1002/j.2161-1912.1997.tb00313.x.

[22] Drucker, H. et al. (1997). Support vector regression machines. *Advances in Neural Information Processing Systems*, 1: 155–161.

[23] Paisitkriangkrai, P. (2012). Linear Regression and Support Vector Regression. In *Proc. Nat. Acad. Sci*, 97(2): 11050-11055.

[24] Wieringe*t*, M. A. et al. (2013). The neural support vector machine. *Belgian/Netherlands Artificial Intelligence Conference*, 247–254.

[25] Krizhevsky, A. et al. (2009). Learning Multiple Layers of Features from Tiny Images.

[26] Xie, L. and Yuille, A. (2017). Genetic CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2017: 1388–1397.

[27] Young, S. R. et al. (2015). Optimising deep learning hyper-parameters through an evolutionary algorithm. *Proceedings of MLHPC 2015: Machine Learning in High-Performance Computing Environments - Held in conjunction with SC 2015: The International Conference for High Performance Computing, Networking, Storage and Analysis*. doi: 10.1145/2834892.2834896.

[28] Rosebrock, A. (2017). *Deep Learning for Computer Vision with Python: Starter Bundle*. PyImageSearch.

[29] Goleman, A. et al. (2019). Dive into Deep Learning. *Journal of Chemical Information and Modeling*, 53(9): 1689–1699.

[30] Patterson, J. and Gibson, A. (2017). *Deep Learning A Practitioner's Approach*, 53(9). doi: 10.1017/CBO9781107415324.004.

[31] Lewis-Beck, M. et al. (2012). R-Squared, *The SAGE Encyclopedia of Social Science Research Methods*, 1187–1190.

[32] Baker, B. et al. (2017). Practical Neural Network Performance Prediction for Early Stopping. In *6th International Conference on Learning Representations*, 2: 1–12.

[33] Chiromaet, H. et al. (2017). Neural networks optimisation through genetic algorithm searches: A review. *Applied Mathematics and Information Sciences*, 11(6): 1543–1564.

[34] Li, L. and Talwalkar, A. (2019). Random Search and Reproducibility for Neural Architecture Search. *ArXiv*. http://arxiv.org/abs/1902.07638

[35] Baker, B. et al. (2017). Accelerating Neural Architecture Search using Performance Prediction. *ArXiv*, 2: 1–7.