Hyperparameter Optimization in Convolutional Neural Network using Genetic Algorithms

Nurshazlyn Mohd Aszemi[1], P.D.D Dominic[2]
Department of Computer and Information Sciences, Universiti Teknologi Petronas, Seri Iskandar, Perak, Malaysia

***Abstract*—Optimizing hyperparameters in Convolutional Neural Network (CNN) is a tedious problem for many researchers and practitioners. To get hyperparameters with better performance, experts are required to configure a set of hyperparameter choices manually. The best results of this manual configuration are thereafter modeled and implemented in CNN. However, different datasets require different model or combination of hyperparameters, which can be cumbersome and tedious. To address this, several works have been proposed such as grid search which is limited to low dimensional space, and tails which use random selection. Also, optimization methods such as evolutionary algorithms and Bayesian have been tested on MNIST datasets, which is less costly and require fewer hyperparameters than CIFAR-10 datasets. In this paper, the authors investigate the hyperparameter search methods on CIFAR-10 datasets. During the investigation with various optimization methods, performances in terms of accuracy are tested and recorded. Although there is no significant difference between propose approach and the state-of-the-art on CIFAR-10 datasets, however, the actual potency lies in the hybridization of genetic algorithms with local search method in optimizing both network structures and network training which is yet to be reported to the best of author knowledge.**

***Keywords*—Hyperparameter; convolutional neural network; CNN; genetic algorithm; GA; random search; optimization**

I. Introduction

Human are capable of recognizing things both environment and object within a second. This recognition skills are trained since human are young. Similarly, if the computers are able to recognize or classifying object and environment by looking for low-level features such as edges and curves, then it can build more abstract concepts of what it recognizes through a series of convolutional layers. Hence, image recognition and classification in the neural network are called Convolutional Neural Network (CNN).

Building CNN requires a set of configurations which is external to the data and manually tune by the machine learning researcher. The variable of the network structure and the network trained of CNN are known as hyperparameters [1]. Finding a set of hyperparameters that gives an accurate model in a reasonable time is also part of the hyperparameter optimization problem [2]. Hyperparameter optimization is a problem that identifies a good model of hyperparameter [3] or a problem of optimizing a loss function over a graph-structured configuration space [4]. Testing all the possible set model of hyperparameter can become computationally expensive [5]. Therefore, the need for an automated and structured way of searching is increasing, and hyperparameter space, in general, is substantial.

Numerous works have been done in optimizing the hyperparameters [3], [6]–[8]. Other optimization methods that have been applied using evolutionary algorithms (EAs) as mentioned in [5]. Bochinski et al [5] defines hyperparameters as the configuration of the network structure which will lead to an optimization problem in finding the optimal configuration of the CNN. Others who have applied evolutionary algorithms are [9] and [10]. However, [11] claims that none of the approaches consider 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 always be optimized.

Very less research has been done to validate these assumptions since the optimization of hyperparameter is theoretically and practically significant [11]. Due to these flaws, the idea of automating hyperparameter search is getting attention in machine learning [12]. This means that most common optimization has been done using the random search [3] and a combination of grid search and manual search [13]. Ozaki [14] claim that most people do not have sufficient computing resources and are unwilling to adjust the hyperparameters that use difficult optimization method. Author acknowledge that there is some research that applies genetic algorithms such as [15], [16] on tuning the hyperparameters of the network and the structure of the system [17] and [18]. However, the work aims to hybridize genetic algorithms with local search method in optimizing the CNN hyperparameters that both are of network structures and network trained which is not studied in these prior works.

To the best of author knowledge, there are no approaches that hybridized genetic algorithms with local search method in optimizing both network structures and training algorithms in CNN. As a start, a trial of an experiment on a random search method will be conducted to testify the performance as per said in [3]. The objectives of this work are twofold: (1) to investigate the hyperparameter search method on CIFAR-10 datasets and (2) to perform benchmarking on CIFAR-10 datasets with the state-of-the-art accuracy.

The remainder of this paper is organized as follows. In Section II, the related work in the area of CNNs and GA is provided. Section III presents the background and Section IV

lay out the experimental setup. Experimental results are discussed in Sections V and VI. Finally, the paper is concluded, and future work is recommended in Section VII.

II. Related Work

In this section, the related works are presented and discussed as follows.

A. Search Optimization Method

Grid search method is a trial and error method for every hyperparameter setting on the specific range of values. The advantage of using a grid search is that it can be easily parallelized [3]. Researchers and practitioners will specify the boundary and steps between values of hyperparameters which will form a grid of configurations [19]. However, if one fails, the rest of the jobs will fail accordingly. In usual cases, machine learner will use a limited grid and then extend which will make the grid more efficient to configure the best while continually searching for the new grid [19]. Four hyperparameters will become impractical as the number of functions to evaluate will increase with adding parameter; this is due to the limitation on dimensionality [20].

Random search method samples the hyperparameter space 'randomly'. Based on [3], the random search has more benefits than grid search regarding the application which can still use even the cluster of the computer 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 [21]. Furthermore, a new trial can be added to the experiment without jeopardizing if more computers become are available [22].

Another latest development in hyperparameter tuning is using Bayesian optimization. It uses distribution over functions which is known as Gaussian Process. To train using Gaussian Process; fitting it to given data is essential as it will generate function closely to observe data. In Bayesian optimization, the Gaussian process will optimize the expected improvement and surrogate the model which is the probability of the new trial and will improve the current best observation. The highest expected improvement will be used next, and expected improvement can be calculated at any point in the search space. Widespread implementation of Bayesian optimization includes spearmint that uses Gaussian process [23]. However, Bergstra et al. [3] claim that the method of Bayesian optimization is limited; as it works on high dimensional hyperparameter and it became very computationally expensive. Therefore, it has poor performance.

B. Genetic Algorithm Optimization

The difference between genetic algorithms and evolutionary algorithms is that the genetic algorithms rely on the binary representation of individuals (an individual is a string of bits) due to which the mutation and crossover are easy to be implemented. Such operations produce candidate values that are outside of allowing searching space. In contrast, the evolutionary algorithms rely on customized data structures and need appropriately craft mutation and crossover which this will heavily dependents on the problem at hand [24]. The author in [25] has been mentioned that Genetic algorithms can be used when there is no information about gradient function at evaluated points. It can achieve good results when there is several local minima or maxima. Unlike any other search method, the function is not determined in a single place but simultaneously in different areas. They can be carried out in several processors since the calculations of the function on all points of a population are independent of each other [26]. Furthermore, they can be parallelized with little effort which makes many paths to the optimum processed in parallel. In [25], it has been mentioned that genetic algorithm has advantages over local methods as they do not remain trapped in suboptimal local maximum or minimum.

III. Foundation

A. Hyperparameters Optimization

The optimization of hyperparameter can be simplified as how many function evaluations will perform on every optimization to select the best hyperparameter in that model. Besides, optimization can be explained in a simple manner which "given a function that accepts inputs and returns a numerical output, how can it efficiently find the inputs, or parameters, that maximize the function's output?" [27]. Hence, upon tuning or optimizing the hyperparameter, author will take input as a function to the hyperparameter model and the output as the measurement on the model performance. Consequently, the hyperparameter optimization problem setup can be formally defined as [2]:

- $\mathcal{A}$ machine learning algorithm λ is a mapping $\mathcal{A}_\lambda$
- $\mathcal{D} \rightarrow \mathcal{M}$ where $\mathcal{D}$ is the set of all datasets and $\mathcal{M}$ is the space of all models
- $\lambda \in \Lambda$ is the chosen hyperparameter configuration with $\Lambda = \Lambda_1 \times \cdots \times \Lambda_P$ being the is P-dimensional hyperparameter space.
- The learning algorithm estimates a model $M_\lambda \in \mathcal{M}$ that minimizes a regularized loss function $\mathcal{L}$ (e.g. misclassification rate):

$$\mathcal{A}_\lambda\left(\mathcal{D}^{(train)}\right) := \underset{\mathcal{M}_\lambda \in \mathcal{M}}{\arg min}\mathcal{L}\left(\mathcal{M}_\lambda, \mathcal{D}^{(train)}\right) + \mathcal{R}\left(\mathcal{M}_\lambda, \lambda\right) \quad (1)$$

The task of hyperparameter optimization is to find the optimal hyperparameter configuration λ using a validation set,

$$\lambda^* := \underset{\lambda \in \Lambda}{\arg min}\, \mathcal{L}\, \underbrace{\left(\mathcal{A}_\lambda, \mathcal{D}^{(train)}\right), \mathcal{D}^{(valid)}}_{=: \mathcal{F}_{\mathcal{D}}(\lambda)} \quad (2)$$

The $\mathcal{F}$ will be the miscalculation rate or error rate. The hyperparameter space all possible values that are usually defined as acceptable bounds for each hyperparameter and the number of the hyperparameter is the dimension of the function [28].

Referring to [29], optimizing the hyperparameter require knowledge on the relationship between the settings and the model performance. It will first run a trial to collect performance on several configurations and then will make an

inference which will decide what configuration will be applied next. The purpose of optimizing is to minimize the number of trials on hyperparameter while finding the optimum model [29]. Hence, author can consider the process as sequential and not parallel.

B. Convolutional Neural Network

Convolutional neural network gain advantages over inputs that consist of images which neurons are arranged in 3 dimensions of width, height, and depth [30]. For examples, CIFAR-10 datasets have volume dimensions of 32x32x3 (width, height, depth). Fig. 1 describes the visualization between a regular three-layer neural network with CNN. A regular 3-layer neural network consists of input – hidden layer 1 – hidden layer 2 – output layer. CNN arrange the neurons into three dimensions of width, height, and depth. Each layer will transform the 3D input to 3D output volume of neuron activations. Hence, the red input layer holds the image, the dimensions of the image will be width and height, and the depth will be the three RGB (red, green and blue) channels.

CNN architectures build in three main types of sequence layers: Convolutional Layer, Pooling Layer, and Fully-Connected Layer. A simple CNN for CIFAR-10 datasets can have the architecture of [INPUT–CONV–RELU–POOL–FC]. As per describe [30].

- INPUT will hold on the raw pixel value of images.
- CONV will compute the output of the neurons.
- RELU stands for Rectified Linear Unit is an activation function that converts all negative pixel values to 0.
- POOL will pass over sections of the image and pool them into the highest value in the section.
- FC (fully-connected) layer will calculate the class scores such as 10 categories in CIFAR-10, and finally, each neuron will be connected to all number in the previous volume.

However, not all layers are in the same sequence as [INPUT–CONV–RELU–POOL–FC]. Some layers have CONV/FC and do not need RELU/POOL. Others require CONV/FC/POOL but not RELU and vice versa. Fig. 2 shows the example of CNN architecture from small VGG Net [31]. The 3D volumes are sliced into rows as it is manageable to see the architecture. For examples, the input can be taken as raw images of the car that eventually will break down into sequences of convolutional layers that will compute and produce the output into their classes. The last layer holds the score of each class which is labeled. The architecture shown is a tiny VGG Net [31]. There are several CNN architectures that have name in image classifications world such as LeNet [32], AlexNet [33], GoogLeNet [34], VGGNet [31] and ResNet [35]. More information about their architectures and state-of-the-art accuracy can be found in their respective papers.

Defining the model architectures can be difficult as there are numerous design choices made available. Author do not know what the optimal model architecture it should be for a given model immediately. Hence, this paper would like to explore a range of possibilities. An actual machine learner will ask the machine to perform this exploration and configure the optimal model architecture automatically. The variable in the configuration can be called hyperparameters which it is external to the model, and the value cannot be estimated from the data. Hyperparameters can be divided into two types:

a) Hyperparameter that determines the network structure such as:

- *Kernel Size* –the size of the filter.
- *Kernel Type*–values of the actual filter (e.g., edge detection, sharpen).
- *Stride*–the rate at which the kernel pass over the input image.
- *Padding*–add layers of 0s to make sure the kernel pass over the edge of the image.
- *Hidden layer*–layers between input and output layers.
- *Activation functions*–allow the model to learn nonlinear prediction boundaries.

b) Hyperparameter that determines the network trained such as:

- *Learning rate*–regulates on the update of the weight at the end of each batch.
- *Momentum*–regulates the value to let the previous update influence the current weight update.
- *A number of epochs*–the iterations of the entire training dataset to the network during training.
- *Batch size–the* number of patterns shown to the network before the weights are updated.

Models can have more than 10 hyperparameters and finding the best combination can be view as the search problem. Hence, the right choice of hyperparameter values can affect the performance of the model.

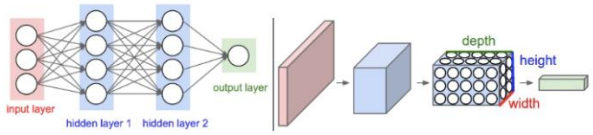

Fig. 1. A regular 3-Layer Neural Network vs. CNNs [30].

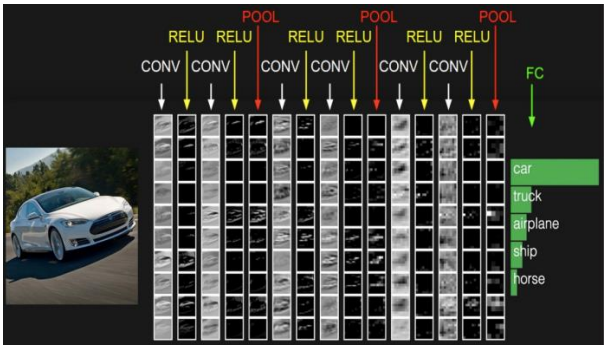

Fig. 2. Small VGGNet Architectures [30].

IV. Experimental Setup

CIFAR10 will be used as datasets [31] as it is a subset of the 80-million tiny image database. There are 50,000 images for training, and 10,000 images for testing. All of them are 32 × 32 RGB images. CIFAR10 contains 10 basic categories, and both training and testing data are uniformly distributed over these categories. To avoid using the testing data, 10,000 images have been left from the training set for validation. This will protect from getting overfitting.

Searching for the best combination of hyperparameters requires computational resources. Fortunately, Nvidia Tesla K80 is a supercomputer that is used as a computing platform in this research. It can dramatically lower data center costs by delivering exceptional performance with fewer and more powerful servers. TensorFlow is a framework that will be used in representing computations as graphs, allows easier computation and analysis of these models and utilizing multi-dimensional arrays called Tensors and by computing these graphs in sessions. TensorFlow will implement Keras as the backend to allow for easy and fast prototyping through user friendliness, modularity, and extensibility. It supports the convolutional neural networks as well running smoothly in CPU and GPU.

To test the understanding on search method, experiments were performed on CIFAR-10 datasets using random search method. In the first experiments, the model only had two convolutional layers as is trained on CPU. The height, width, channels and outputs are fixed which is 32x32x3 with output of 10. 24 iterations within epoch will run on accuracy check. The maximum number of epochs was 500 with 4 rounds of early stopping. The model then is trained with the hyperparameter configurations based Table I and Table II. Note that the range values randomly. Then, the hyperparameter evaluations are being stored in training logs. The results were being evaluated with two other experiments that will run on GPU. Additional hyperparameters might be added to improve the accuracy, and then the selected hyperparameter configuration will be compared with the state-of-the-art accuracy on CIFAR-10 datasets.

Table I below highlights the hyperparameters considered in this study. Each of the hyperparameters is labeled with shorter and easier name (abbreviation). Also, ranges are indicated within square brackets. The following table presents the network trained hyperparameters.

TABLE I. Network Structure Hyperparameters

Hyperparameter	Abbreviation	Range
Number of Filters	Filters_1	[16, 32, 64, 96]
Kernel Size	Ksize_1	[3, 4, 5]
Number of Filters	Filters_2	[48, 64, 96, 128]
Kernel Size	Ksize_2	[3, 4, 5]
Number of Filters	Filter_3	[64, 96, 128]
Kernel Size	Ksize_3	[3, 4, 5]
Hidden Layer	full_hidden1	[60, 100, 125]
Hidden Layer	full_hidden2	[60, 100, 125]
Activation	activation	[‘relu’, ‘lrelu’, ‘elu’]

TABLE II. Network Trained Hyperparameters

Hyperparameter	**Abbreviation**	**Range**
Learning rate	learning_rates	[0.001, 0.003, 0.01,0.03]
Batch Size	batch_sizes	[32, 64, 128, 256]
Momentum	momentum	[0.9, 0.95, 0.99]
Optimizer	optimizer	[‘Adam, ‘rmsprop’, ‘Nesterov']

In Table II above, the four-network trained hyperparameters are listed alongside their abbreviations. These abbreviations will be constantly used as reference to any of the above-listed hyperparameters.

A. Results on Small CNNs on CPU

The accuracy only reached at **60.85%** and takes **5 days** to run on **CPU**. Hence, the hyperparameter space were minimized by only performing optimization on network structure, learning rate, and batch size. Overall, small CNNs contain 9 network hyperparameters, 2 network trained hyperparameters and 2 layer hyperparameters resulting total of $9 + 2 + 2 \times 2 = 15$ configurable hyperparameters. The possible combinations of hyperparameters are 15,116,544. Based on the results, training accuracy did not reach satisfaction level and overfitting is not an issue. However, the learning rates were not congregating as shown in Fig. 3. Hence, this will be eliminated on further search experiments. Architecture with more features in fully connected layers perform better. Next experiments, the third convolutional layer are added. Few selections of activation functions and optimization algorithms are included and run on GPU.

B. Results on Small CNNs on GPU

In second experiments, activation functions were added and optimization algorithms along with momentum for batch normalization. The border pixels [0, 1, 2] were removed. Third convolutional layers were added and running **3 days** on **GPU.** Based on the results, the accuracy was increased from **60.85%** to **71.17%** and appears the model had difficulty on converging. Overall, small CNNs contain 9 network hyperparameters, 4 networks trained hyperparameters and 3-layer resulting in a total of $9 + 4 + 3 \times 3 = 22$ configurable hyperparameters.

In Fig. 3 below, the accuracy on fail learning rate were tested by 0.01 and 0.03. The numerical representation of these results is presented in Appendix 1. Subsequently, the accuracy was also tested on small CNN on GPU as depicted in Fig. 4 below. Please refer to Appendix 2 for detailed results.

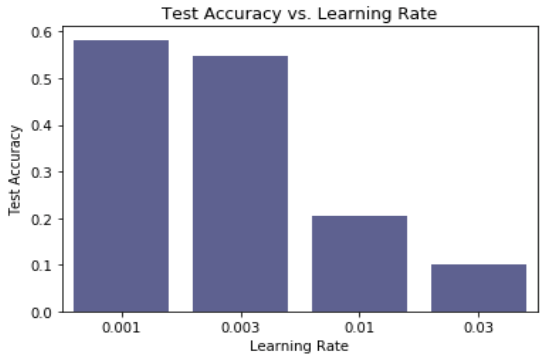

Fig. 3. Test Accuracy on Fail Learning Rate by 0.01 and 0.03.

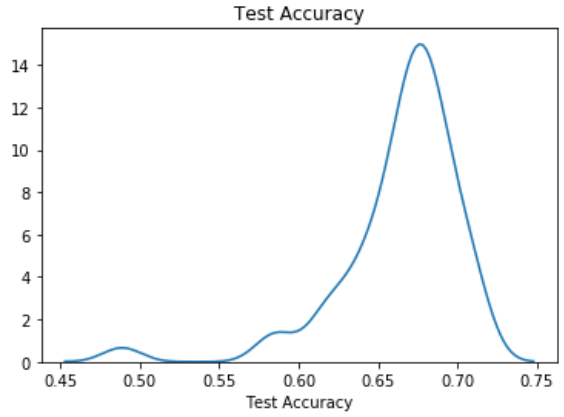

Fig. 4. Test Accuracy of Small CNNs on GPU.

C. Benchmarks on CIFAR-10 Datasets

In final experiments, hyperparameters that tended to underperform were eliminated. Fig. 4 shows the distribution of test accuracy appears to be models that had difficulty converging. The stride of first CNN layer are reduce from 2 to 1 and patch reduction to 0. The possible combinations of hyperparameters were reduced from **15,116,544** to **3,072.** The increasing number in hyperparameter to learn results in increasing training time. Hence, the hyperparameter setting was sampled accordingly as in Table III. Fig. 5 illustrates the test accuracy in finding the optimum hyperparameter that fits the CNNs model with additional third convolutional layers. 'relu' or 'lrelu' offer performing better than 'elu', with 'relu' dominating on activation functions. The Adam optimizer algorithm are better than 'Nesterov Accelerated Gradient' or 'RMSProp' in this case. Batch size of 32 gives inconsistent results. A batch size of 64 can achieve good results and is more reliable. However, while other CNNs have had success with this approach, there is no benefit from the current architecture.

In the first convolutional layer, number of filters are increased and gives an improvement performance. Interestingly, the next two layers are not performing well with more filters. Perhaps because of overfitting. A 4x4 kernel gives slightly an average performance. In the second experiments, a batch size of 64 and the "lrelu" activation function looks promising. Hence, these were both fixed and with more features in 1st conv layers are helping gaining better performance. The stride is reduced from 2 to 1 in the final experiments. Signs of overfitting are shown. Thus, regularization is applied which is the dropout layers and running on **2 days** on **GPU**. Based on the results, it achieved **80.62% accuracy** which then compared with the state-of-the-art on CIFAR-10 datasets without interfering with the data (data augmentation).

This put the results on the bottom of the leaderboard for cifar-10 in Table IV, which is satisfying and the purpose of this study, was to investigate hyperparameter search on CIFAR-10 datasets [36].

D. Random Search and Genetic Algorithm

In [41], random search generates randomly individual solutions at any point of search space via calculating and comparing the value of each solution while genetic algorithms (GA) mimic the natural evolutions process. In the search space, GA works with populations of solutions which each generation is subjected to selection, crossover and mutation operations. This will help GA in obtaining the newly improved generations of the solutions. However, the questions arise as can genetic algorithm become the random search. The author in [41] stated that GA does not have the potential to become a purely random search alone. Conversely, Yahya et al. [42] considered genetic algorithms as "Guided Random Search Algorithms". The randomness of the algorithms can be controlled and become guided search as GA takes inspirations from the evolution concepts such as survival of the fittest, crossover, mutation and selections. Moreover, the degree of randomness of algorithms can be determined by setting the values of its control parameters, and through it, the algorithms can be purely random search algorithms or deterministic algorithms [42].

The following figures depict the results obtained from the experiments. Starting from Fig. 5 to Fig. 7, accuracy is tested on activation function, optimizer, and batch size.

It can be observed from Fig. 5 that 'relu' or 'lrelu' offer performing better than 'elu', with 'relu' dominating on activation functions. The raw results are shown in Appendix 2.

On the other hand, Adam optimizer algorithm is better than 'Nesterov Accelerated Gradient' or 'RMSProp' in this case, please see Appendix 2.

TABLE III. NETWORK STRUCTURE HYPERPARAMETERS

Hyperparameter	Abbreviation	Range
Learning rate	learning_rate	[0.001, 0.003, 0.002,0.0015]
Batch Size	batch_size	[64]
Momentum	momentum	[0.9, 0.95, 0.99]
Optimizer	optimizer	['adam']
Number of Filters	filters1	[64, 96]
Kernal Size	ksize1	[4, 5]
Number of Filters	filters2	[96, 128]
Kernal Size	ksize2	[4, 5]
Number of Filters	filter3	[96, 128]
Kernal Size	ksize3	[4, 5]
Hidden Layer	full_hidd1	[100, 125]
Hidden Layer	full_hidd2	[100, 125]
Activation	activation	['lrelu']

TABLE IV. ACCURACY ON CIFAR-10 WITHOUT DATA AUGMENTATION

Network	CIFAR-10 (%)
ALL-CNN [37]	92.00
Deeply-supervised [38]	90.22
Network in Network [39]	89.60
Maxout [40]	88.32
3Conv + 2FC + Dropout + stride 1 on all layers	**80.62**

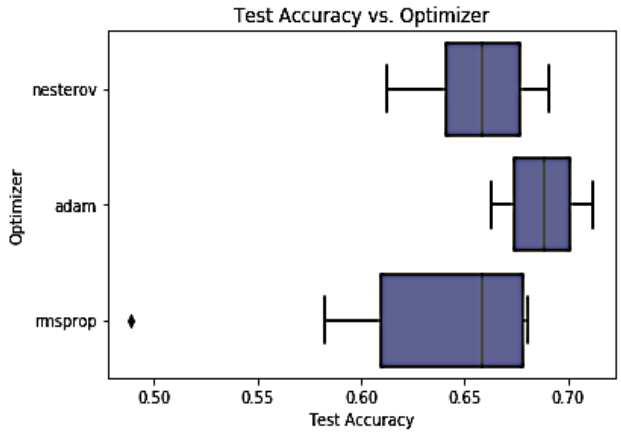

Fig. 5. Accuracy vs Optimizer.

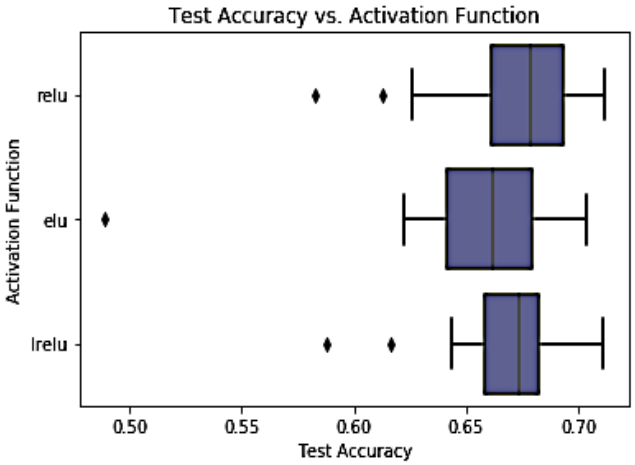

Fig. 6. Accuracy vs Activation Function.

On batch size, results shows that batch size 32 have higher median performance. Although this seems to be good achievement, a higher batch size (64) is capable of achieving better and reliable results. Next, the results of the optimised network structure hyperparameters are presented. Appendix 2 shows the results in a tabular form.

In Fig. 8 Optimizing the network structure using random search for filters seems to improve performance at first convolutional layer. However, as number of layers increase, the performance begins to decline significantly. Although the exact reasons that have caused the drastic decline might be inconclusive at this stage, author attributed it to the possible overfitting. This is nonetheless subject to debate through more detailed experiments. Please refer to appendix 2 for raw results of filters after being optimized by random search. In Fig. 9 below however, the situation shows better results.

Alternatively, optimizing the network structure using random search for kernel performs better on 4x4 kernel. However, a similar situation with filters is observed in kernel too (Fig. 9). Kernel 1x1, 2x2 and 3x3 performed below average respectively. Yet, slightly better performance is noticed on the rest of the kernels. For better understanding of the results resented in Fig. 9, raw data of the results are as shown in Appendix 2.

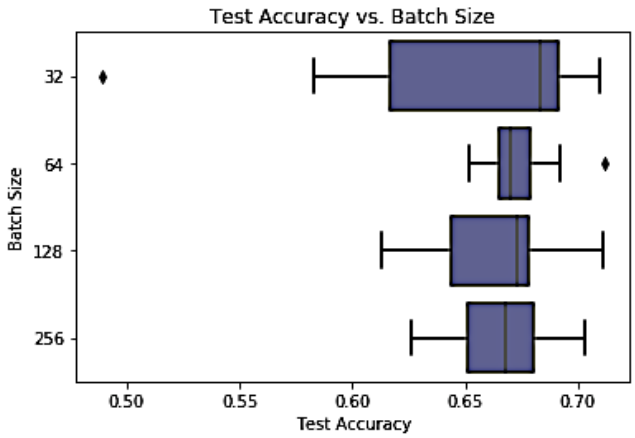

Fig. 7. Accuracy vs Batch Size.

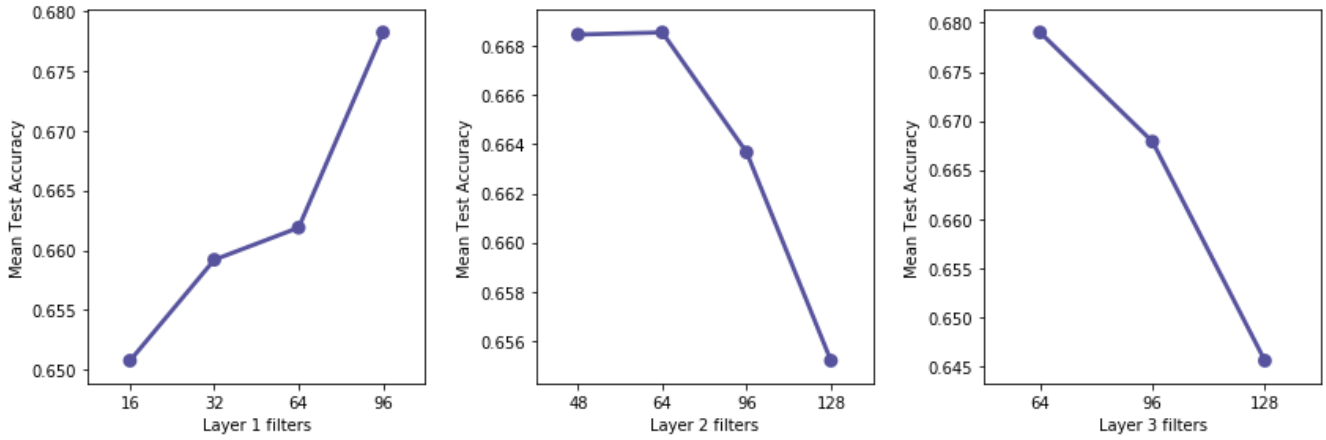

Fig. 8. Network Structure Hyperparameters after being Optimized by Random Search (Fiters).

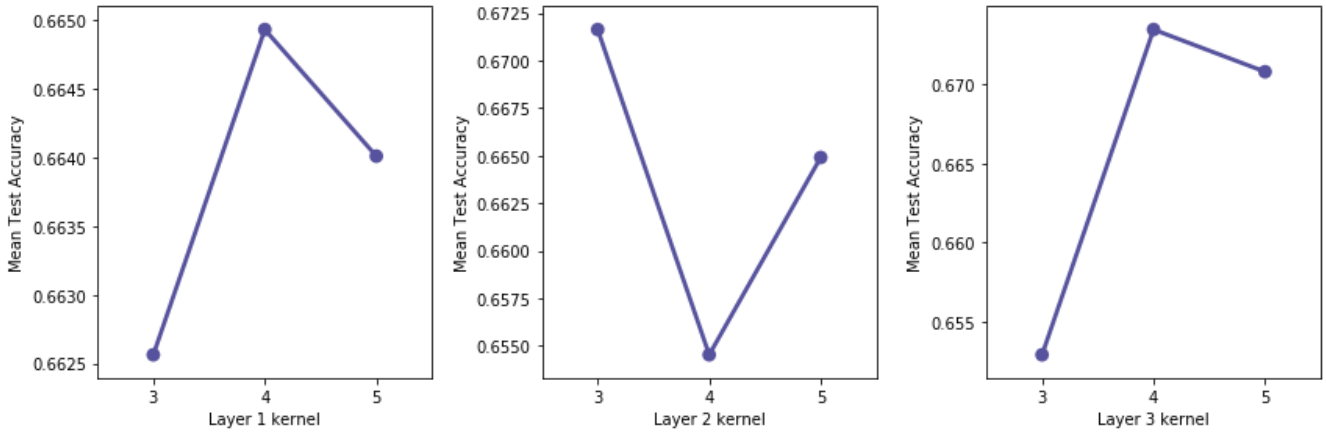

Fig. 9. Network Structure Hyperparameters after being Optimized by Random Search (Kernel).

E. Computational Considerations

Computing random search on CPU is computational costly as it took around 5 days with limited 3GB memory. Running on GPU accelerate the computing process into 2 days while eliminating the huge hyperparameter combinations. Moreover, constructing CNNs architectures can consume a lot of memory. Modern GPU is equipped on limited of 3/4/6GB memory with the best GPUs of 12GB memory [30]. To avoid this, three sources of memory need to keep track on:

- **Activations:** The raw number of activations can consume a lot. Running CNNs only at test time can reduce this by storing the currents activations at any layer and discards the previous activations.
- **Parameters:** These are the number that holds the network parameters, step cache for optimization such as momentum, 'Adagrad' or 'RMSprop'. The memory store needs to be multiplied by a factor of 3 or so.
- **Miscellaneous**: Every CNN as to maintain the miscellaneous such as image data batches, etc.

Calculate the amount of memory by taking the right estimate of the total number values of activations, parameters, misc. and multiply by 4 to get a raw number of bytes. And then divided by 1024 multiple times to get in KB, MB and finally GB. Another way to make the network fit is to decrease the batch size as the most memory was consumed by activations [30].

V. Conclusions and Future Work

Author presented a genetic algorithm method on optimizing the convolutional neural network hyperparameter. For this purpose, author first investigate the hyperparameter search method focusing on image classification of CIFAR-10 datasets. Random search method was to choose on running the trial of experiments. The small CNNs on CPU showed that learning rates with range of 0.01 and above are not giving good performance on the models. Performing grid search will waste a lot of time with the mention learning rates. Millions combination of hyperparameter will make Grid search seems impractical. Random search allows some hyperparameter values to be selected by process of elimination or selection. The random search did not achieve state-of-the-art accuracy by 90% above. However, it satisfies enough to quality over par around 80% on the leaderboard of CIFAR-10 [36].

In the future, genetic algorithms (GA) will be used as an optimization search method with tested CNNs architectures. Combining search methods such as grid search, manual search, random search and local search with a global search like GA will be implement for further research. However, running GA as hyperparameter search space is computationally cost. Hence, running on multiple GPU are taken into considerations by the author. Given enough time, more optimization method such as Bayesian can be investigated like random search on this paper.

Acknowledgment

The author would like to thank High-Performance Cloud Computing Centre (HPC³) and Universiti Teknologi PETRONAS for supporting this study.

References

[1] L. Xie and A. Yuille, "Genetic CNN," in Proceedings of the IEEE International Conference on Computer Vision, 2017, vol. 2017-October, pp. 1388–1397.

[2] M. Wistuba, N. Schilling, and L. Schmidt-Thieme, "Hyperparameter optimization machines," in Proceedings - 3rd IEEE International Conference on Data Science and Advanced Analytics, DSAA 2016, 2016.

[3] J. Bergstra, J. B. Ca, and Y. B. Ca, "Random Search for Hyper-Parameter Optimization Yoshua Bengio," 2012.

[4] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag, "Collaborative hyperparameter tuning," Proc. 30th Int. Conf. Mach. Learn., 2013.

[5] E. Bochinski, T. Senst, and T. Sikora, "Hyperparameter Optimization For Convolutional Neural Network Committees Based on Evolutionary Algorithms," I2017 IEEE Int. Conf. Image Process., pp. 3924–3928, 2017.

[6] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," Jun. 2012.

[7] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in IJCAI International Joint Conference on Artificial Intelligence, 2015.

[8] K. Eggensperger, M. Feurer, and F. Hutter, "Towards an empirical foundation for assessing bayesian optimization of hyperparameters," NIPS, BayesOpt Work., pp. 1–5, 2013.

[9] C. Gagn, "DEAP : Evolutionary Algorithms Made Easy," J. Mach. Learn. Res., vol. 13, pp. 2171–2175, 2012.

[10] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets," in Proceedings - 20th International Conference on Artificial Inteligence and Statistics (AISTATS) 2017, 2017.

[11] S. Sanders and C. Giraud-Carrier, "Informing the use of hyperparameter optimization through metalearning," in Proceedings - IEEE International Conference on Data Mining, ICDM, 2017, vol. 2017-November, pp. 1051–1056.

[12] M. Claesen and B. De Moor, "Hyperparameter Search in Machine Learning," 2015.

[13] P. LeCun, Yann; Bottou, L.;Bengio, Y.;Haffner, "Lecun-01a," Proc. IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[14] Y. Ozaki, M. Yano, and M. Onishi, "IPSJ Transactions on Computer Vision and Applications Effective hyperparameter optimization using Nelder-Mead method in deep learning," IPSJ Trans. Comput. Vis. Appl., vol. 9, 2017.

[15] L. Xie and A. Yuille, "Genetic CNN," in Proceedings of the IEEE International Conference on Computer Vision, 2017.

[16] H. Pérez-Espinosa, H. Avila-George, J. Rodriguez-Jacobo, H. A. Cruz-Mendoza, J. Martínez-Miranda, and I. Espinosa-Curiel, "Tuning the Parameters of a Convolutional Artificial Neural Network by Using Covering Arrays," Res. Comput. Sci., vol. 121, no. 2016, pp. 69–81, 2016.

[17] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks.," IEEE Trans. Neural Netw., vol. 8, no. 3, pp. 694–713, 1997.

[18] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2011, vol. 6683 LNCS, pp. 507–523.

[19] L. Li and A. Talwalkar, "Random Search and Reproducibility for Neural Architecture Search," ArXiv, 2019.

[20] I. Dewancker, M. McCourt, and S. Clark, "Bayesian Optimization for Machine Learning : A Practical Guidebook," Prepr. ArXiv, 2016.

[21] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for Hyper-Parameter Optimization.".

[22] J. Bergstra, D. L. K. Yamins, and D. D. Cox, "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures," Icml, pp. 115–123, 2013.

[23] J. Snoek et al., "Scalable Bayesian Optimization Using Deep Neural Networks," Feb. 2015.

[24] R. Chiong and O. K. Beng, "A Comparison between Genetic Algorithms and Evolutionary Programming based on Cutting Stock Problem," Eng. Lettters, vol. 14, no. 1, 2007.

[25] R. Rojas, Neural Networks: A Systematic Introduction. Springer-Verlag, 1996.

[26] H. M. uhlenbein GMD Schloss Birlinghoven D- and S. Augustin, "Asynchronous parallel search b y the parallel genetic algorithm."

[27] Q. V Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On Optimization Methods for Deep Learning," Proc. 28th Int. Conf. Int. Conf. Mach. Learn., pp. 265–272, 2011.

[28] M. Wistuba, N. Schilling, and L. Schmidt-Thieme, "Learning hyperparameter optimization initializations," in Proceedings of the 2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015, 2015.

[29] E. Hazan, A. Klivans, and Y. Yuan, "Hyperparameter Optimization: A Spectral Approach," ICLR, Jun. 2018.

[30] A. Karpathy, "CS231n Convolutional Neural Networks for Visual Recognition," Standford Education..

[31] G. Krizhevsky, A., Nair, V., & Hinton, "The CIFAR-10 dataset," 2014. [Online]. Available: online: http://www. cs. toronto. edu/kriz/cifar. htm.

[32] Y. Lecun, L. Eon Bottou, Y. Bengio, and P. H. Abstract|, "Gradient-Based Learning Applied to Document Recognition."

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Adv. Neural Inf. Process. Syst., pp. 1097–1105, 2012.

[34] C. Szegedy et al., "Going Deeper with Convolutions," Sep. 2014.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," Dec. 2015.

[36] R. Benenson, "Classification datasets results," 2016.

[37] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for Simplicity: The All Convolutional Net," 2014.

[38] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-Supervised Nets," Sep. 2014.

[39] M. Lin, Q. Chen, and S. Yan, "Network In Network," Dec. 2013.

[40] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout Networks," 2013.

[41] A. Aab et al., "De Mauro, 41 J.R.T. de Mello Neto, 35 I. De Mitri, 31, 32, 36," J.R. Hörandel, vol. 56, p. 24, 2017.

[42] A. A. Yahya, R. Mahmod, and A. R. Ramli, "Dynamic Bayesian networks and variable length genetic algorithm for designing cue-based model for dialogue act recognition," Comput. Speech Lang., vol. 24, no. 2, pp. 190–218, Apr. 2010.

APPENDICES

APPENDIX 1 (NUMERICAL RESULTS OF ACCURACY ON FAIL LEARNING RATE BY 0.01 AND 0.03)

To get the best accuracy in the test sets, the experiment was performed in three rounds. Up to 60.85% was achieved in the first round (using 2 convolutional, 2 fully connected). In this first round are equipped with two convolutional layers. This was progressively increased to 3 in the subsequent rounds as shown in Appendix 2 and 3. The training of this model was done on CPU; thereby limiting the number of convolutional. This results maps with Fig. 3.

```
In [2]: import pandas as pd
        hyperparam_df = pd.read_csv('./log/model_hyperparam_1.csv')
        hyperparam_df.sort_values(by = ['best_acc'], ascending = False)
```

Out[2]:

	batch_size	best_acc	best_loss	filters_1	filters_2	full_hidd_1	full_hidd_2	ksize_1	ksize_2	learning_rate	no_epochs
6	256.0	0.6085	1.124841	64.0	96.0	125.0	125.0	5.0	5.0	0.001	18.0
5	128.0	0.6002	1.142996	32.0	64.0	100.0	125.0	3.0	5.0	0.001	20.0
0	32.0	0.5978	1.147990	32.0	96.0	100.0	80.0	3.0	4.0	0.001	19.0
25	64.0	0.5903	1.196584	64.0	96.0	100.0	125.0	3.0	3.0	0.001	23.0
3	256.0	0.5806	1.192041	32.0	48.0	60.0	125.0	3.0	3.0	0.001	24.0
2	32.0	0.5749	1.211263	16.0	32.0	60.0	100.0	5.0	4.0	0.003	35.0
10	64.0	0.5735	1.226173	32.0	96.0	60.0	100.0	3.0	3.0	0.001	20.0
18	64.0	0.5606	1.266603	32.0	64.0	60.0	100.0	5.0	4.0	0.003	21.0
12	256.0	0.5598	1.247521	8.0	32.0	125.0	80.0	5.0	3.0	0.003	30.0
26	64.0	0.5589	1.245890	32.0	48.0	125.0	80.0	4.0	5.0	0.003	18.0
4	128.0	0.5307	1.304451	8.0	96.0	125.0	100.0	4.0	4.0	0.003	23.0
24	128.0	0.5296	1.316891	8.0	64.0	100.0	125.0	3.0	4.0	0.003	18.0
16	32.0	0.5266	1.396713	16.0	64.0	60.0	80.0	5.0	4.0	0.003	28.0
21	256.0	0.5261	1.326381	8.0	48.0	125.0	125.0	4.0	4.0	0.001	20.0
8	128.0	0.4189	1.511615	16.0	96.0	125.0	80.0	4.0	4.0	0.010	20.0
28	64.0	0.4175	1.584211	16.0	32.0	125.0	80.0	3.0	3.0	0.010	32.0
22	64.0	0.1001	2.302906	8.0	64.0	100.0	125.0	3.0	5.0	0.010	14.0
15	32.0	0.1000	2.303990	16.0	32.0	125.0	80.0	4.0	3.0	0.030	12.0
17	256.0	0.1000	2.302994	8.0	64.0	125.0	80.0	3.0	4.0	0.030	12.0
1	32.0	0.1000	2.302861	32.0	64.0	60.0	125.0	4.0	3.0	0.010	23.0
19	256.0	0.1000	2.303015	32.0	48.0	60.0	125.0	4.0	4.0	0.030	13.0
20	128.0	0.1000	2.303385	64.0	96.0	60.0	100.0	3.0	4.0	0.030	12.0
13	128.0	0.1000	2.302733	8.0	48.0	100.0	80.0	4.0	4.0	0.010	12.0
23	32.0	0.1000	2.303946	8.0	32.0	60.0	100.0	3.0	4.0	0.030	12.0
11	32.0	0.1000	2.303967	16.0	96.0	125.0	80.0	3.0	3.0	0.030	12.0
9	32.0	0.1000	2.303927	8.0	32.0	125.0	125.0	3.0	3.0	0.030	12.0
7	32.0	0.1000	2.303434	32.0	96.0	60.0	80.0	5.0	5.0	0.030	13.0
27	128.0	0.1000	2.302751	8.0	96.0	60.0	125.0	4.0	5.0	0.010	12.0
14	32.0	0.1000	2.303959	8.0	96.0	100.0	125.0	3.0	3.0	0.030	12.0

APPENDIX 2 (NUMERICAL REPRESENTATION OF THE RESULTS PRESENTED FROM FIG. 4 – FIG. 9)

In the second round of the experiments, up to 71.17% was achieved using 3 convolutional fully connected layers. This experiment was conducted to test the accuracy of small CNNs on GPU. Also, accuracy was tested against activation function (RELU, LRELU or ELU), optimizer (Adam, RMSProp or Nesterov Adam) as well as batch size. In addition, random search was used to optimise network structure hyperparameters for both filters and kernel.

Table of results

```
In [115]: hyperparam_df = pd.read_csv('./log/model_hyperparam_2.csv')
          hyperparam_df.sort_values(by = ['best_acc'], ascending = False)
```

Out[115]:

activation	batch_size	best_acc	best_loss	best_train_acc	best_train_loss	filters1	filters2	filters3	full_hidd1	full_hidd2	ksize1	ksize2	ksize3	learning_rate
relu	64	0.7117	0.919377	0.754637	0.711197	96	96	64	100	125	3	5	3	0.0020
lrelu	128	0.7109	0.906563	0.758348	0.701145	32	64	64	125	80	5	3	5	0.0020
relu	128	0.7093	0.897427	0.743490	0.734613	96	128	96	100	100	5	3	4	0.0020
relu	32	0.7093	0.873374	0.722251	0.805403	96	64	64	60	80	5	4	4	0.0030
elu	256	0.7033	0.910308	0.734158	0.771317	96	48	128	100	125	5	5	4	0.0015
relu	128	0.7008	1.032702	0.755814	0.708703	96	96	128	60	125	5	5	5	0.0030
relu	32	0.6966	0.940112	0.704069	0.840530	64	48	64	125	125	4	5	5	0.0030
lrelu	32	0.6947	0.906030	0.692000	0.867232	96	48	96	60	80	3	5	3	0.0020
relu	64	0.6923	0.946625	0.724659	0.792448	16	64	128	100	125	5	5	5	0.0030
lrelu	32	0.6911	0.905265	0.746788	0.762321	32	128	128	100	125	4	5	5	0.0015
relu	128	0.6898	0.896807	0.708308	0.837255	64	48	64	60	100	3	3	5	0.0010
elu	32	0.6891	0.934515	0.723434	0.806494	32	64	64	125	80	3	4	5	0.0030
elu	32	0.6870	0.904974	0.698303	0.855170	32	48	96	60	125	4	4	3	0.0010
lrelu	32	0.6831	0.923607	0.696061	0.872021	96	64	64	100	125	4	5	3	0.0010
lrelu	256	0.6810	0.904818	0.669038	0.952634	64	128	128	100	80	5	3	4	0.0020
relu	64	0.6806	0.998204	0.728308	0.783186	96	64	64	100	125	5	3	4	0.0020
elu	32	0.6806	0.928317	0.682667	0.900012	32	64	96	125	80	4	4	3	0.0030
relu	256	0.6802	1.084211	0.721635	0.786616	64	128	64	125	100	3	3	4	0.0020

APPENDIX 3

In third and final round of the experiments, up to 80.62% was achieved using 3 convolutional layers, This shows that increase in parameters learning, increase the training time. Accordingly, only 5 hyperparameter were sampled and values of hyperparameter that are not performing well will be eliminated.

Table of results

```
In [4]: hyperparam_df = pd.read_csv('./log/model_hyperparam_3.csv')
        hyperparam_df.sort_values(by = ['best_acc'], ascending = False)
```

Out[4]:

activation	batch_size	best_acc	best_loss	best_train_acc	best_train_loss	filters1	filters2	filters3	full_hidd1	full_hidd2	ksize1	ksize2	ksize3	learning_rate
lrelu	64	0.8062	0.579502	0.736910	0.751420	96	128	128	125	100	4	5	5	0.0030
lrelu	64	0.8048	0.587198	0.761165	0.673304	96	96	128	125	125	4	5	5	0.0010
lrelu	64	0.7894	0.622922	0.721077	0.797818	64	128	128	100	100	5	5	5	0.0030
lrelu	64	0.7799	0.632761	0.714930	0.794581	64	96	96	125	100	4	5	5	0.0010
lrelu	64	0.7750	0.649881	0.697231	0.850535	64	96	128	100	125	5	4	5	0.0015