Journal of Universal Computer Science, vol. 25, no. 6 (2019), 647-666
submitted: 7/1/19, accepted: 20/5/19, appeared: 28/6/19 © J.UCS

Hybrid Stochastic GA-Bayesian Search for Deep Convolutional Neural Network Model Selection

Waseem Rawat
(Department of Electrical and Mining Engineering, University of South Africa
Florida 1709, South Africa
wrawat10@gmail.com)

Zenghui Wang
(Department of Electrical and Mining Engineering, University of South Africa
Florida 1709, South Africa
wangzengh@gmail.com)

Abstract: In recent years, deep convolutional neural networks (DCNNs) have delivered notable successes in visual tasks, and in particular, image classification related applications. However, they are sensitive to the selection of their architectural and learning hyperparameters, which impose an exponentially large search space on modern DCNN models. Traditional hyperparameter selection methods include manual model tuning, grid, or random search but these require expert domain knowledge or are computationally burdensome. On the other hand, Bayesian optimization and evolutionary inspired techniques have surfaced as viable alternatives to the hyperparameter problem. In this work, an alternative automated system that combines the advantages of evolutionary processes and state-of-the-art Bayesian optimization is proposed. Specifically, the search space is first partitioned into separate discrete-architectural, and continuous and categorical learning parameter subspaces, which are then efficiently traversed by a stochastic genetic search applied to the former, combined with a genetic-Bayesian search of the latter. Several sequential experiments on prominent image classification tasks reveal that the proposed method results in overall classification accuracy improvements over several well-established techniques, and significant computational costs reductions compared to brute force computation.

Keywords: Convolutional neural networks, Genetic algorithms, Bayesian optimization, Hybrid systems, image classification, model selection
Categories: 1.2.0, I.2.8, I.2.10, I.4.0

1 Introduction

Recent years have seen the rapid advancement of convolutional neural networks (CNNs), fuelled by the application of GPUs for neural network computation, the availability of large labelled datasets, and several algorithmic enhancements. This has resulted in their application to various traditional and diverse computer vision tasks, with ground-breaking success [Suong & Jangwoo 2018]. These accomplishments have led researchers to progress several DCNN components, resulting in a plethora of improvements to their architecture, pooling layers, activation functions, loss functions, regularization techniques, optimization procedures, and computational characteristics [Rawat & Wang 2017]. On the other hand, DCNN successes have prompted others to scrutinize their internal mechanisms and gain a better

understanding of their operation and expressive ability, resulting in research into several open issues. For example DCNNs are not invariant to large scale geometric deformations [Gong et al. 2014], current models impose considerable storage and memory constraints averting mobile deployment [Iandola et al. 2016], and describing the semantic content of images is still a big challenge [Vinyals et al. 2015]. Furthermore, despite some progress [Mallat 2012], [Wiatowski & Bolcskei 2015], [Bengio et al. 2017] theoretical motivations of why DCNNs are successful are largely devoid.

Moreover, deep learning models require numerous architectural and hyperparameter choices, such as the number and size of the convolutional and pooling filters, the need to use or negate regularization techniques such as Dropout [Hinton et al. 2012], and the important choice of which activation function to use. The learning methods such as the optimization technique, and its associated learning rate, the number of epochs and the size of each batch presented to the network, and the weight initialization method to adopt, also need to be selected. When the learning method choices are combined with the architectural choices, the number of possible models grows exponentially with each additional parameter, making DCNNs computationally expensive to use. The problem is exacerbated when the structure of the model is considered (network depth, type of layers etc). The traditional methods for model selection include the grid[1] [Pedregosa et al. 2011] and random [Bergstra & Bengio 2012] search techniques, and manual tuning; however, all of these have their own challenges. The manual model selection approach requires expert domain knowledge or unsystematic rules of thumb [Dernoncourt & Lee 2016], the grid search technique is computationally burdensome [Snoek et al. 2012], and whilst the random search approach relaxes some of the computational load imposed by grid search, it is not directed towards promoting high performing models.

On the other hand Bayesian optimization has emerged as an influential solution for the automated selection of DNN models [Snoek et al. 2012], [Swersky et al. 2013], [Shahriari et al. 2016], and in particular, Bayesian optimization based on Gaussian processes [Rasmussen & Williams 2006] is known to work well for continuous variables [Loshchilov & Hutter 2016]. However, the search spaces, which contain continuous variables, are naturally more complex, and have a higher dimensionality in contrast to discrete spaces, thus making Bayesian optimization well suited to traverse them. Despite this, Bayesian algorithms impose a significant administrative overhead and require expert knowledge in order to obtain sensible results [Dewancker et al. n.d], and furthermore, is inherently sequential in nature, thus preventing superlative parallelization [Loshchilov & Hutter 2016]. Divergently, Genetic Algorithms (GAs) can be parallelized if required, and have been shown to perform better than grid search techniques for support vector machines [Martino et al. 2011], and neural networks [Ding et al. 2013], [Tao et al. 2007]. One of the main advantages of using the GA is its generality, in other words it can be used for a diverse range of applications due to its simplicity, and independence of the underlying problem. More specifically, the GA operators are mostly independent of a given problem, and thus only the codification of the population and fitness function for the problem are required to use the technique [Orive et al. 2014].

[1] Grid search is sometimes referred to as brute-force computation

Thus, in this paper, we propose a hybrid method, which combines the generality of GAs and the scalability of Bayesian optimization, and use the combined technique to search for the optimal hyperparameters of DCNN's, with the intention of eliminating the need for a computationally costly grid search or the requirement for domain specific expert model selection. The contributions of this paper are the following:

- We separate the large search space of modern DCNNs into discrete architectural, and categorical and continuous learning subspaces, with the intention of applying different optimization techniques to search for their optimal parameters.
- We present a biologically inspired stochastic genetic algorithm (GA) for the model selection problem, and use it to efficiently search the architectural space of DCNNs.
- We combine the architectural search with a state-of-the-art Bayesian approach on top of the stochastic GA, and use the hybrid approach to efficiently traverse the learning subspace.

The results demonstrate the computational superiority of the proposed method over the grid search technique, and whilst it demonstrates characteristics of a random search (Bergstra & Bengio, 2012), it has an additional advantage of using previous fitness evaluations and exploration / exploitation trade-offs to direct it, resulting in improvements in overall classification accuracies when it is compared to other methods. The remainder of the paper is arranged as follows. Section 2 gives a literature review of GAs and Bayesian optimization for DCNN's, and the motivations of the proposed methods, whilst Section 3 provides a brief overview of GA and Bayesian optimization. Thereafter, a methodology of applying the GA-Bayesian approach to the problem of DCNN model selection is formalized in Section 4. Experiments and comparisons to other techniques when using the proposed hybrid stochastic GA-Bayesian technique are given in Sections 5-6, before the paper is closed out with the limitations of the presented method and insights into future work.

2 Existing GA and Bayesian Approaches

2.1 GA Optimization of DCNNs

Neuroevolution, which entails applying evolutionary processes to evolve the structure and architecture of neural networks, has seen several applications [Ding et al. 2013]. Although neuroevolutional-based techniques have been successful, their adaptation to DCNNs has not been studied extensively in the past, probably because of the complicated structure, large model size, and significant computational burden imposed by modern DCNNs [Desell 2017].

Recent studies have begun focusing on the optimization of supervised DNNs. For example, [Loshchilov & Hutter 2016] optimized the hyperparameters of existing DCNNs, in a large-scale parallel setting, whilst [Desell 2017] proposed using a distributed network of over 5000 computers, and over two months of computation, to evolve the architectural and learning parameters of DCNNs. However, given the computational requirements of these methods, their large-scale adaptation is not

practical. Moreover, recent works have shown that using evolution to automatically learn the structure and hyperparameters of CNNs, is at the forefront of the current DCNN advances. For instance, [Xie et al. 2017] freshly proposed encoding CNNs as binary strings so that they can be subject to a standard GA, whilst [Miikkulainen et al. 2017] newly proposed using evolution to learn the topology and hyperparameters of deep models. However, these methods are complex, yield compound and unprincipled structures, and keep several key building blocks of DCNN's such as the number and sizes of the convolutional filters and Dropout rates fixed. Nevertheless, these are essential for optimal classification performance.

In general, traversing the parameter search space to select the optimal model parameters (hereafter referred to as model selection) for modern DCNNs using GAs or other evolutionary strategies requires excessive computation, since each member of the population represents an individual DCNN that needs to be trained and scored. Furthermore, if the architectural (number of filters, filter sizes, activation functions, the use of Dropout and Dropout rate) and learning parameters (optimizer, learning rate, batch size and weight initialization) both form part of the search space, the number of possible models grow exponentially with each additional parameter, thus making a GA based search intractable.

Considering these challenges, a traditional, yet highly stochastic GA, is presented to find the near optimal architectural parameters of a DCNN with a fixed structure. Unlike the complicated approaches of others [Loshchilov & Hutter 2016]; [Miikkulainen et al. 2017], the presented method shows that such sophistication is unnecessary and that standard, yet highly stochastic, evolutionary processes can be used for model selection. Furthermore, previous work relied on elaborate computing power [Loshchilov & Hutter 2016], [Desell 2017] or at least the use of GPU's [Xie et al. 2017], [Miikkulainen et al. 2017] to merge GAs with DCNNs, however, here it is shown that a stochastically orientated GA guided search, can lead to classification improvements over baseline models, even with computation constrained to CPU alone. Moreover, to prevent using large GA populations and running the GA for numerous iterations, both of which will add to computation, the model selection search space is efficiently partitioned into the architectural and learning subspaces. Specifically, the stochastically inclined GA is used to optimize the architectural space.

2.2 Bayesian Optimization of DCNNs

Whilst the stochastic GA alleviates some of the challenges associated with the grid and random [Bergstra & Bengio 2012] search techniques, such as their computational load and the lack of direction towards high performing models, as the model selection search space increases, to search for near optimal solutions requires several runs of evolution with extremely large population sizes. Naturally, this significantly hinders computation [Elbeltagi et al. 2005]. Furthermore, although GAs are well suited to search discrete or categorical parameters, such as the options of the architectural search space, traditional GAs, are not suitable for continuous parameters, since a genetic search will be intractable. Thus, as the dimensionality and complexity of the search space increase, it can be computationally beneficial to use other methods that efficiently seek for the best model parameters.

Recently, Bayesian optimization [Mockus et al. 1978] has emerged as a sophisticated, yet effective and powerful, solution to the model selection problem

[Snoek et al. 2012]. Bayesian optimization can efficiently find near optimal parameters for a diverse range of models including statistical methods such as Markov chain Monte Carlo models [Hamze et al. 2013], deep belief networks [Bergstra et al. 2011], and most significantly DCNNs [Snoek et al. 2012], [Swersky et al. 2013]. Whilst the optimization of the architectural parameters are conducted through a stochastic GA, the learning parameters, some of which are of the continuous type thereby making them intractable for a genetic search, are optimized through the better suited Bayesian optimization approach. Bayesian optimization has been applied to the model selection problem for DCNNs previously [Snoek et al. 2012], [Swersky et al. 2013], however, the approach presented here aims at combining it with GAs, which has not been studied in prior work. Furthermore, different from other complex Bayesian algorithms or other intricate GAs, the presented hybrid method is simple to implement, and can be parallelized if required.

3 Background

3.1 Genetic Algorithms

GAs, a subclass of Evolutionary Algorithms (EA's), maintain a population of solutions that traverse a solution space and they use evolutionary processes to obtain near optimal solutions. Each solution is evaluated and based on the score or fitness of the individual solutions, the population is evolved. During the evolutionary process, the genetic operations of selection, mutation and crossover are used to produce offspring chromosomes (or children) and this simulates the natural process of survival of the fittest. These genetic operations evolve the population by improving its overall fitness and thus generate feasible solutions to the optimization problem. While other parameters are required, the performance of a GA is principally governed by the population size, number of generations, crossover rate and mutation rate. As the population size and number of generations increase, the probability of finding an optimal solution is also increased, however this comes with an increase in computational costs [Elbeltagi et al. 2005].

3.2 Bayesian Optimization

Bayesian optimization, explores the search space of a given domain, through deliberating exploring new areas and exploiting areas where good performance has been perceived, by using previous observations of an objective function to define the next point of observation. Similar to the GA optimization approach discussed in the previous section, and other typical types of optimization, in the framework of Bayesian optimization, we are interested in searching for the global maximum (or minimum) of an unspecified objective function. Formerly, for the maximum case, we have:

$$\mathrm{x}^* = \arg\max_{\mathrm{x} \in \mathcal{X}} f(\mathrm{x})$$

where f is the objective function and $\mathcal{X}$ is a bounded set or the search space of interest, which can be conceived as a subset of $\mathbb{R}^d$. For a general optimization

problem, $\mathcal{X}$ is more often than not a compact subset of $\mathbb{R}^d$, however in the Bayesian case, it can be generalized to more uncommon spaces that consider conditional or categorical inputs, or several of these inputs in the case of combinatorial search spaces. Bayesian optimization has two fundamental components. Firstly, it constructs a surrogate regression model, which is inherently probabilistic and consists of a prior distribution, to capture the confidence regarding the behavior of the black-box objective function, and secondly, an observational model defines the mechanism that generates the data [Shahriari et al. 2016].

4 Method

4.1 Stochastic GA for architectural search

4.1.1 Methodology

The architectural parameters are optimized using the GA, whilst the learning parameters are held fixed and separately optimized using Bayesian optimization. For the proposed GA, each member of the population is subjected to the evolutionary operators, and constitutes a set of topological choices, and thus an individual CNN model, denoted by $\mathrm{I}_{t,n}$. An example of these choices is illustrated in Table 1.

4.1.2 Evolutionary process

4.1.2.1 Initialization, Selection and Retention

The details of the genetic process is summarized by **Algorithm 1**. Formerly, a set of randomized individual CNN models $\{\mathrm{I}_{0,n}\}_{n=1}^{N}$ are used to initialize the population of CNNs. Each network is then trained on a subset (training set $\mathcal{D}_{tr}$) of an image classification dataset $\mathcal{D}$, before being evaluated on its test set $\mathcal{D}_{ts}$. Since the fitness function of the GA channels the evolutionary process, and is dependent on the optimization task, it is imperative to use an appropriate fitness function [Lessmann et al. 2005]. Given that the task is image classification, classification accuracy is selected. Here the classification accuracy takes the notation $a_{t-1,n}$, as the evaluation of the n-th individual CNN $I_{t-1,n}$ takes place before the crossover operation of the t-th generation. The training and evaluation process is computationally expensive, as each model is trained and evaluated from scratch, and thus, this step is the bottleneck of the evolutionary process. The networks are then categorized according to their classification accuracy, and only the top performing individual CNNs $\{\mathrm{I}_{t,n}\}_{n=1}^{r_p}$, where r_p represents the predetermined percentage of models to be retained, are selected to evolve the population via reproduction and become part of the next generation $\{\mathrm{I}_{t,n}^{'}\}_{n=1}^{N}$. To prevent getting trapped in local extremes, a subset of the poor performing CNNs, are also retained, with a random probability p_r.

4.1.2.2 Crossover

Initialization, selection and retention, is followed by random crossover c_r, in which children CNNs are breed from randomly selected pairs of parent members $\mathrm{I}_{t,2n-1}, \mathrm{I}_{t,2n}$ of the retained population (top performing CNN's and the retained poor performers). The number of children that are breed is dependent on the number of individuals *N* in the initial population $\{\mathrm{I}_{0,n}\}_{n=1}^{N}$, and the number of retained models. For example if $N = 16$, in the initial population, and 25% of the top performers were retained, plus another two of the weaker CNNs, ten children will need to be breed in order to maintain the original population size for the next generation $\{\mathrm{I}_{t,n}^{'}\}_{n=1}^{N}$. With this scheme, there is a possibility of an individual CNN appearing in different generations, since *N* remains unchanged from the initialized population $\{\mathrm{I}_{0,n}\}_{n=1}^{N}$. During crossover, randomly selected topological choices from parent CNNs are crossed over to children CNNs, as illustrated by Table 1, where the selected parameters are represented by the shaded blocks.

4.1.2.3 Mutation

Crossover is followed by mutation of the children, where the rate of mutation is controlled by p_m. The lack of mutation can cause a population to lack diversity and devolve, and thus mutation is imperative to promote diversity, augment the capability of the population and facilitate propagation [Floreano & Mattiussi 2008]. To implement mutation, a child CNN is selected with probability p_m and a randomly selected topographical feature of it is replaced with another arbitrarily selected feature, resulting in a mutated population $\{\mathrm{I}_{t,n}^{r}\}_{n=1}^{N}$. The effect of mutation on CNN architecture is illustrated in Table 1, where the shaded blocks represent the mutated topographical parameters. Mutation in this fashion facilitates the retention of the majority of the strong topographical characteristics of the selected CNN, whilst also providing a chance of evaluating new CNN architectures. The entire evolutionary process is repeated for a predetermined number of generations T.

4.2 GA-Bayesian optimization of the learning parameter subspace

4.2.1 Methodology

The learning parameter optimization procedure for the given image classification task can be formalized by **Algorithm 2**, which applies Bayesian optimization to the learning parameter search. Here the unknown blackbox function denoted by f, represents the GA inspired DCNN model, with selectable learning parameters x, and stochastic and independently computed accuracy $y = f(\mathrm{x})$. In this context, the Bayesian algorithm is used to query f, for a designated set of learning parameters, denoted by the point x_{i+1} and the results are observed at a tentative observation point y_{i+1}, which represents the accuracy on the validation set, computed after training the model on the training set. The sequential queries of the Bayesian optimized learning subspace continue for a predetermined number of maximum iterations I, which is set by a specified computational budget. When I is reached, a final set of learning parameters, denoted by $\bar{\mathrm{x}}_{\mathrm{I}}$ is proposed by the Bayesian algorithm, and this represents the last GA derived architectural, and Bayesian optimized learning parameter

recommendation. The best recommendation of the search is denoted by x_B, since it is possible that $\bar{x}_I \neq x_B$. The learning parameters are optimized using Bayesian optimization, whilst the GA inspired architectural parameters are held constant. For the proposed GA-Bayesian model selection procedure, the top performing GA derived model, is subject to a Bayesian search, where each iteration i of the Bayesian loop represents a set of learning parameters, and thus an individual CNN model. The overall hybrid optimization procedure is shown in Figure 1.

Conv. 1		**Conv. 2**		**Fully conn. 1**			**Fully conn. 2**			**Act.**
Filt. no.	Filt. size	Filt. no.	Filt. size	Filt. no.	Drop. use?	Drop. rate	Filt. no.	Drop. use?	Drop. rate	Funt. to use?
Parent CNN_A										
64	3*3	32	5*5	16	No	0.25	256	No	N/A	ELU
Parent CNN_B										
32	5*5	16	4*4	128	Yes	0.5	512	Yes	0.75	ReLu
Child CNN_C										
32	3*3	16	5*5	128	Yes	0.5	256	No	N/A	ReLu
Child CNN_M – Before mutation										
32	3*3	16	5*5	128	Yes	0.5	256	No	N/A	ReLu
Child CNN_M´ – After mutation										
32	3*3	16	5*5	128	Yes	0.5	256	No	N/A	ELU

Table 1: Illustration of crossover between parent CNN_A and CNN_B resulting in a child CNN_C and mutation in the offspring after crossover has taken place, resulting in a mutated CNN_ M´

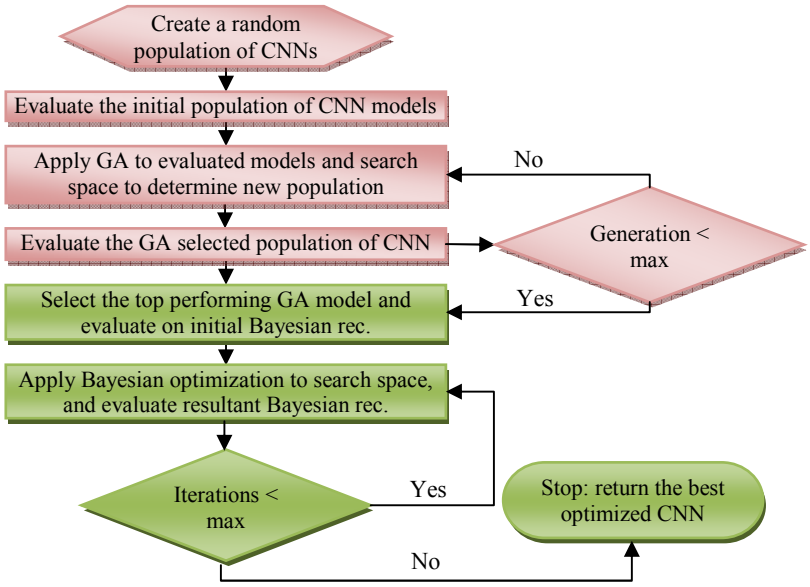

Figure 1: The proposed hybrid GA-Bayesian approach

Algorithm 1: DCNN model selection flow, using genetic processes of selection, crossover, and mutation

Algorithm 1: CNN model selection using the GA **Data:** Reference classification training set $\mathcal{D}_{tr}$ and test set $\mathcal{D}_{ts}$ **Genetic process inputs:** The number of CNNs in the initial and subsequent generations *N*, the maximum number of generation *T*, the percentage of top performing networks in each generation r_p and the probability p_r of random poor performers being retained, random crossover c_r, and the rate of mutation p_m.
1: GA initialization: Generate a random initial population of CNNs $\{\mathbf{I}_{0,n}\}_{n=1}^{N}$
2: Train and evaluate the initial population: Train each individual $I_{t-1,n}$ on $\mathcal{D}_{tr}$ and evaluate its accuracy $a_{t-1,n}$ on $\mathcal{D}_{tr}$
for each generation, t = 1; 2; 3; 4; :: ; T, **repeat** the following genetic operations:
3: Selection: Select a percentage r_p of the top performing CNNs $\{\mathbf{I}_{t,n}\}_{n=1}^{r_p}$ to retain for the next generation, plus add random poor performing models with probability p_r, to form the next generation $\{\mathbf{I}'_{t,n}\}_{n=1}^{N}$.
4: Crossover: Perform random crossover c_r, for each CNN pair $\mathbf{I}_{t,2n-1}$, $\mathbf{I}_{t,2n}$ to maintain the population at *N*
5: Mutation: Select children randomly with probability p_m, and mutate a randomly selected topographical choice.
6: Evaluation: Repeat step 2 for each generation
until the predetermined number of generations *T* is complete
Result: The final generation $\{\mathbf{I}_{T,n}\}_{n=1}^{N}$ of CNNs with their classification accuracies.

Algorithm 2: GA-DCNN model learning parameter selection through Bayesian optimization

Algorithm 2: GA-DCNN learning parameter selection using Bayesian optimization **Data:** Reference classification training set $\mathcal{D}_{tr}$ and test set $\mathcal{D}_{ts}$ **Bayesian optimization inputs:** The iteration number i for each sequential Bayesian search, the learning parameters x_0 for the initial Bayesian search, and subsequent parameters x_{i+1}, the observed classification accuracy $y = f(\mathrm{x})$ observed at an initial point y_0, and the succeeding points of observation, denoted by y_{i+1}.
1: Bayesian loop initialization: Get an initial learning parameter suggestion x_0 from the Bayesian loop
2: Train the initial DCNN: Train the top performing GA derived model on the training set $\mathcal{D}_{tr}$, with the learning parameters x_0, suggested by the Bayesian loop
3: Observe the results: Validate the model on the test set $\mathcal{D}_{ts}$, at y_0 by observing the classification accuracy $y = f(\mathrm{x})$.

for each iteration, i = 1; 2; 3; 4; :: ; I, **repeat** the following steps:
4: Subject the results to a Bayesian search: Pass the observed accuracy $y = f(\mathrm{x})$, from step 3 to the Bayesian optimization loop, and obtain the next set of learning parameters
5: Model parameter fitment and training reiteration: Fit the GA selected DCNN from step 2, with the new Bayesian suggested learning parameters x_{i+1}, and retrain the model on D_{tr}
6: Evaluation: Repeat step 3, and observe the results on D_{ts} for each iteration i, at y_{i+1}
until the predetermined number of iterations I is complete
Result: The final Bayesian recommend learning parameter $\bar{\mathrm{x}}_{\mathrm{I}}$, and the best found Bayesian selection x_{B}, for the GA optimized architecture, and their associated classification accuracies y_I and y_B.

5 Experiments

5.1 Development Environment

All simulations were conducted on an 8-core Intel i7-6700k CPU, clocked at 4.0 GHz (4.2 GHz maximum frequency), with an 8 MB cache, and 16GB DDR4 random access memory (RAM). The software experiments were implemented in Python using the Keras [Chollet et al. 2015] application program interface (API) whilst TensorFlow [Abadi et al. 2015] was used as the backend. Other necessary Python libraries and dependencies are used, such as numPy and sciPy.

5.2 Data

Experimentation was conducted on the MNIST [LeCun 1998] and CIFAR-10 [Krizhevsky 2009] datasets. For the former, the traditional train-test split of 60000-10000, was maintained for all simulations, whilst for the latter the standard 50000-10000 train-test split was utilized. Except for normalization and data shuffling, no further preprocessing or data augmentation was considered.

5.3 Experimentation on MNIST

5.3.1 Base model and GA architectural choices

The selected base model for the GA search was derived from the LeNet-5 model [LeCun 1998]; however, selected modern architectural changes such as maximum pooling [Ranzato et al. 2007] and Dropout [Hinton et al. 2012] were included to improve performance. Compared to MNIST, CIFAR-10 is more complex and generally requires deeper models, however, this comes with high computational costs (Rawat & Wang, 2017). Given the limited computational resources, only one additional set of convolutional and pooling layers were added to the model used for the MNIST simulations. Whilst the classification performance is not state-of-the art, the parameters inherent to the model depth provides greater opportunities for hyperparameter optimization compared to the MNIST model and thus it was used to test the proposed GA-Bayesian search technique on a more complex benchmark than

MNIST. The softmax loss function was used as the objective function and it was minimized using the traditional and widely accepted Stochastic Gradient Decent [SDG - Bottou 1998] algorithm, whilst the batch size was set to 64. The convolutional and fully connected filters were initialized using the scheme presented in [Glorot & Bengio 2010]. The architecture variants subjected to optimization, can be seen in Table 3.

5.3.2 Brute-force / random search model selection

To validate the GA algorithm (CNN_GA_1.X), the technique was tested against the grid search approach (CNN_BF_1.X), whilst for the remaining runs CNN_GA_2.X, CNN_GA_3.X; it was compared against the random search technique, due to the intractability of searching a large space using brute force computation. On MNIST, the models were trained for twenty epochs, whilst ten was used for CIFAR-10. For random search, the number of models searched was equivalent to the number of models of the GA search.

5.3.3 GA based model selection

Algorithm 1 was applied to the same search space as the grid and random searches. The GA parameters for the different runs are shown in Table 2. For the first run, a small population size and number of generations were used to save on computation and facilitate comparison to grid search; however, these were increased for the subsequent runs of the algorithm. Moreover, high mutation rates were set to promote diversity within the population, and prevent the GA from getting stuck in a local maxima. Other options of these GA specific hyperparameters were also tried but these led to suboptimal results. Although other combinations may produce better results, no hyperparameter tuning was done, since this would require the training and evaluation of a complete population of CNNs for several generations, which will be a computationally exorbitant procedure.

GA Hyperparameter	CNN_GA_1.X	CNN_GA_2.X	CNN_GA_3.X
Population	$N = 12$	$N = 40$	$N = 20$
No. of generations	$T = 10$	$T = 20$	$T = 10$
Percentage of models to retain	$r_p = 50\%$	$r_p = 25\%$	$r_p = 25\%$
Probability of retention	$p_r = 10\%$	$p_r = 10\%$	$p_r = 10\%$
Probability of mutation	$p_m = 0.3$	$p_m = 0.3$	$p_m = 0.3$
Total no. of models evaluated	*61*	*483*	*121*
Total no. of possible models	*108*	*11664*	*34992*

Table 2: GA parameters for the different GA runs

Layer	Hyper -parameter	CNN_GA_1.X / CNN_BF_1.X / CNN_RA_1.X	CNN_GA_2.X / CNN_RA_1.X	CNN_GA_3.X / CNN_RA_3.X
Convolutional layer 1	Number of filters	{8, 16, 32, 64}	{16, 32, 64}	{16, 32, 64}
	Kernel size	5*5	{3*3;4*4;5*5}	{3*3; 4*4; 5*5}
Max pooling layer 1	Filter size	2*2	2*2	2*2
Convolutional layer 2	Number of filters	{16, 32, 64}	{16, 32, 64}	{16, 32, 64}
	Kernel size	5*5	{3*3; 4*4; 5*5}	{3*3; 4*4; 5*5}
Max pooling layer 2	Filter size	2*2	2*2	2*2
Convolutional layer 3	Number of filters			{16, 32, 64}
	Kernel size			{3*3; 4*4; 5*5}
Max pooling layer 2	Filter size			2*2
Fully connected layer 1	Number of filters	{64, 128, 256}	{64, 128, 256}	{64, 128, 256}
	Dropout rate / Alpha Dropout rate	0.5	{0, 0.25, 0.5, 0.75} / {0, 0.025, 0.05, 0.1}	{0, 0.25, 0.5, 0.75}
Fully connected layer 2	Number of filters	{64, 128, 256}	Same as Fully connected layer 1	{64, 128, 256}
	Dropout rate / Alpha Dropout rate	0.5	{0, 0.25, 0.5, 0.75} / {0, 0.025, 0.05, 0.1}	{0, 0.25, 0.5, 0.75}
Global parameters	Activation Function - All layers except softmax layer	ReLu / Softmax	{ReLu; ELU; SeLu} / Softmax	{ReLu; ELU; tanh} / Softmax
	Optimization	SGD – Lr. Rate: 0.01	SGD – Lr. Rate: 0.01	SGD – Lr. Rate: 0.01

Table 3: Search space for the different approaches of the architectural search

5.3.4 Experimental results and analysis

The mean accuracies of the top performing models (top-10, and final GA generations vs. their grid counterparts), the entire computational runs, and the top performing models for each run, are shown in Tables 4-5. Notable redundancies in parameter choices are observed for all techniques. Notwithstanding these redundancies and the stochastic nature of the computation, when compared to grid search (Table 4), the GA search obtained very similar results, for a drastic reduction in computation (40%[2]), which illustrates its ability to find the top performing models, whilst against random search[3] the GA technique achieved superior mean and top performing accuracies, as illustrated by Table 5.

Evaluation criteria	Brute-force search CNN_BF_1.X	GA search GA_CNN_1.X
Mean accuracy of top 12 models (brute / GA)	98.94	**98.95**
Mean accuracy of top 61 models searched / all GA models	98.81	**98.86**
Top performing model	98.98	**99.02**

Table 4: Limited GA model selection and comparison to grid search

Evaluation criteria	MNIST		CIFAR-10	
	Random search CNN_RA_2.X	GA search CNN_GA_2.X	Random search CNN_RA_3.X	GA search CNN_GA_3.X
Mean accuracy - top 10 models	99.05	**99.12**	61.16	**64.33**
Mean accuracy of the top models / last generation of the GA	98.98	**99.01**	60.15	**62.34**
Mean accuracy of complete search	98.68	**98.96**	52.18	**60.08**
Top model	99.06	**99.17**	63.93	**65.51**

Table 5: Extended GA model selection and comparison to other techniques on MNIST and CIFAR-10

[2] Grid force – 1044 minutes; GA – 620 minutes
[3] Computation times were similar as a result of experimental design

5.4 Bayesian optimization implementation

Bayesian optimization was carried out using the SigOpt API[4] (see Dewancker et al. 2016a, 2016b, 2016c for recent work), which utilizes an ensemble of state-of-the art Bayesian optimization techniques to find the optimal parameters for a wide variety of tasks, such as the DCNN's studied here. More specifically, it aims to maximize a blackbox objective function f [Dewancker et al. 2016b], by utilizing Gaussian processes to model it [Dewancker et al. 2016c] and the expected improvement acquisition function to trade-off exploration and exploitation of the search space.

5.4.1 Brute-force / random search, and Bayesian model selection

For the first Bayesian exploration, CNN_BA_1.X (70 Bayesian observations), the technique was tested against the brute force and random search approaches, whilst for the remaining runs CNN_BA_2.X (120 Bayesian observations), CNN_BA_3.X (70 Bayesian observations), it was compared against random search, due to the intractability of searching a large space using brute force computation. On MNIST, the models were trained for twenty epochs, whilst ten was used for CIFAR-10. For the random search, the number of models searched was equivalent to the number of models of the GA search. The search spaces are shown in Table 6.

Global parameter	**CNN_RA4_1.X / CNN_BA_1.X**	**CNN_RA_5/6.X / CNN_BA_2/3.X**
Initialization method	Random_normal; Lecun_normal; He_normal; Glorio_normal	Random_normal; Lecun_normal; He_normal; Glorio_normal
Learning rate	0.001 – 0.01 (step size = 0.01)	0.001 – 0.01 (continuous)
Optimizer	Adagrad; RMSprop; SGD	RMSprop; SGD; Adagrad; Adam; Adamax; Nadam
Batch size	64	32 - 64 (with step size = 1)

Table 6: Learning parameter choices exposed to the first, second and third Bayesian searches, and their random and grid counterparts

5.4.2 Experimental results

The mean accuracies of the top performing models (top-10), the entire computational runs, and the top performing models for each run are shown in Tables 7 and 8. When compared to grid search, as expected, the top performing models obtained similar results, however, the mean accuracies of the top performing models and the entire runs were superior for the Bayesian search, albeit for a drastic reduction in computation (> 40%[5]). The Bayesian approach outperforms random search for all the measures used in this section on both MNIST and CIFAR-10. Moreover, when the complexity of the search space is increased (variables treated as continuous variables

[4] https://sigopt.com
[5] Grid force – 1580 minutes; GA-Bayesian search – 950 minutes

– see Table 6), the superiority of the Bayesian search over random search is amplified, as illustrated by Table 8.

MNIST Dataset	**Limited grid vs. random vs. Bayesian search**			**Extended random vs. Bayesian search**	
Evaluation criteria	**CNN_BF_2.X**	**CNN_RA_4.X**	**CNN_BA_1.X**	**CNN_RA_5.X**	**CNN_BA_2.X**
Mean accuracies of top 10 models	99.20	99.13	**99.32**	99.37	**99.44**
Mean accuracies of the complete search	98.79	98.31	**99.11**	98.71	**99.32**
Top performing model	99.37	99.24	**99.38**	99.40	**99.47**

Table 7: Performance analysis between the brute-force, random and Bayesian approaches on MNIST

Evaluation criteria	**Random search CNN_RA_6.X**	**Bayesian search CNN_BA_3.X**
Mean accuracy of top 10 models (random / Bayesian)	69.81	**70.24**
Mean accuracy of all randomly searched models / all the Bayesian searched models	63.21	**67.70**
Top performing model	70.08	**70.78**

Table 8: Performance analysis between the random, and Bayesian approaches on CIFAR-10

6 Comparison to other optimization techniques

The best GA-Bayesian run, which is a combination of CNN_GA_2.X and CNN_BA_2.X on MNIST, and CNN_GA_3.X and CNN_BA_3.X on CIFAR-10 were also compared against TPE[6], which is also a Bayesian technique, and the Simulated Annealing (SA) optimization techniques of the state-of-the-art Hyperopt framework [Bergstra et al. 2015]. The number of evaluations used for these techniques were equated to the total number of combined GA-Bayesian evaluations, whilst the epochs used per model was maintained the same as the GA-Bayesian runs. Moreover, they were asked to traverse the combined search space traversed by the

[6] Tree-structured Parzen estimator

combined GA-Bayesian searches. As illustrated in Table 9 below, the GA-Bayesian technique outperformed its counterparts on all the classification measures used in this section, albeit for a slightly greater computational time on MNIST.

Evaluation criteria	MNIST			CIFAR-10		
	SA	TPE	GA-Bayesian	SA	TPE	GA-Bayesian
Mean acc. of top 10 models	99.23	99.29	**99.44**	69.54	69.79	**70.24**
Mean acc. of the complete search	96.64	97.1	**99.14**	58.76	59.11	**63.89**
Top performing model	99.3	99.34	**99.47**	70.55	70.65	**70.78**
Computation time (mins)	6226	**5841**	7321	2390	2395	**2158**

Table 9: Performance analysis between TPE, SA and the GA-Bayesian search

7 Discussion and conclusions

A grid search of the architectural search space dissects the possible topological choices into equivalently sized (with regards to each dimension) grids, resulting in a uniform sampling of all the possible architectures, and this entails training and validating complete DCNN models at each intersection of the partitioned space. The downside is that this requires searching over an exponential number of dimensions, which is computationally exorbitant given the cost of training a single model, and this is further compounded when the learning search space is also traversed, since this subspace introduces variables that are continuous in nature. Whilst some of these computational costs can be reduced if a random search is conducted, random search is not directed towards the top performing models. On the other hand, the GA-Bayesian approach uses several random operations (random population generation, crossover and mutation) of a stochastic GA to promote a type of random search of the architectural space, however, it poses an additional benefit of directing the search towards the selection of high performing models through its selection and retention mechanisms, the latter of which is also stochastic in nature.

Whilst the GA search is well suited to the discrete hyperparameters of the architectural subspace, using it to traverse the complex learning parameter subspace is not practicable. Thus, to mitigate this, the space was searched using a Bayesian search on top of the GA derived models, and whilst the GA-Bayesian search exhibited randomness with regards to the initial Bayesian samples, and subsequent exploration related samples, it was focussed towards the exploitation of the top performing

DCNN models. Subsequently, the combined hybrid search approach is able to significantly reduce the computational load, when compared to grid search [Pedregosa et al. 2011], and improve the classification accuracy, when compared to random search, as well as SA and TPE, as demonstrated by several simulations on the MNIST and CIFAR-10 benchmarks.

The GA and Bayesian specific hyperparameters such as the crossover and mutation rates and the number of Bayesian evaluations, were not optimized due to limited computational resources, however, with a larger computational budget, optimizing them can lead to improved overall performance and is thus left for future work. The GA component of the search is time consuming, especially if a large number of generations are used, which was the case with the MNIST simulations. Whilst, this leads to improved classification performance, optimizing the algorithm to improve the time costs is an attractive direction that still needs to be explored. Furthermore, whilst the presented technique was used to search for the near optimal set of hyperparameters in this work, extending it to optimize other parameters such as the depth of the network (i.e. number of layers in a network) and using it to explore the various building blocks of modern DCNNs to search for novel architectures, is another interesting direction left for upcoming research. Moreover, the presented method was commissioned on the task of image classification, which has been dominated by DCNNs in recent years [Rawat & Wang 2017]. However, DCNNs have also been shown to work well for object detection and segmentation tasks [Girshick et al. 2014], [Girshick 2015]. Further to the requirement of different architectural hyperparameters compared to image classification tasks, the techniques presented for object detection and segmentation tasks, such Regions with CNN features (R-CNN) [Girshick et al. 2014], have their own hyperparameters that require tuning. These hyperparameters include region warping padding, bounding box-regression and non-maximum suppression threshold choices. Thus, given the computational and accuracy gains of using the proposed GA-Bayesian searching strategy, and the fact that the method works independently of model training and validation, it is conceivable that the presented optimization methods can be generalized to search for the hyperparameters for object detection, segmentation and other similar computer vision related tasks.

In closing, separating the search space imposed by DCNN models into architectural and learning parameter subspaces, to respectively promote the convergence of small evolutionary populations in minimal generations, and to facilitate a Bayesian search for continuous and continuous-like learning parameters can lead to significant overall performance improvements over well-established techniques. The proposed technique becomes especially useful when operating on a tight computational budget, and whilst further experimentation on natural colour images is still required, the performance of the presented approach justifies its use as a viable option for traversing the high dimensional and complex search space of our current models.

Acknowledgements

This work was supported in part by the South African National Research Foundation (Grant Nos. 112142 and 112108), South African National Research Foundation

Incentive Grant (No. 114911) and the Tertiary Education Support Programme (TESP) of South African ESKOM

References

[Abadi et al. 2016] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., & Ghemawat, S.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*. (2016).

[Bengio et al. 2017] Bengio, Y., Mesnard, T., Fischer, A., Zhang, S., &Wu, Y.: STDP-compatible approximation of backpropagation in an energy-based model. *Neural Computation*, *29*(3), 555–577.

[Bergstra & Bengio 2012] Bergstra, J., & Bengio, Y.: Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, *13*, (281-305). (2012).

[Bergstra et al. 2015] Bergstra, J., Komer, B., Eliasmith, C., Yamins, D. & Cox, D.D., 2015. Hyperopt: A python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, *8*(1), (1-24). (2015).

[Bottou 1998] Bottou, L.: Online learning and stochastic approximations. *On-Line Learning in Neural Networks, 17*(9), (142-177). (1998).

[Chollet 2015] Chollet, F.: Keras. *URL http://keras. io*. (2015).

[Dernoncourt & Lee 2016] Dernoncourt, F., & Lee, J. Y.: Optimizing neural network hyperparameters with Gaussian processes for dialog act classification. In *Spoken Language Technology Workshop (SLT), 2016 IEEE* (406-413). IEEE. (2016).

[Desell 2017] Desell, T.: Large Scale Evolution of Convolutional Neural Networks Using Volunteer Computing. *arXiv preprint arXiv:1703.05422*. (2017).

[Dewancker et al. 2016a] Dewancker, I., McCourt, M., Clark, S., Hayes, P., Johnson, A., & Ke, G.: A Stratified Analysis of Bayesian Optimization Methods. *arXiv preprint arXiv:1603.09441*. (2016a).

[Dewancker et al. 2016b] Dewancker, I., McCourt, M., Clark, S., Hayes, P., Johnson, A., & Ke, G.: Evaluation System for a Bayesian Optimization Service. *arXiv preprint arXiv:1605.06170*. (2016b)

[Dewancker et al. 2016c] Dewancker, I., McCourt, M., Clark, S., Hayes, P., Johnson, A., & Ke, G.: A Strategy for Ranking Optimization Methods using Multiple Criteria. In *ICML Workshop on Automatic Machine Learning* (11-20). (2016c).

Dewancker n.d.] Dewancker, I., McCourt, M., Clark, S.: *Bayesian Optimization Primer* [White paper]. Retrieved August 22, 2017, from SigOpt: https://sigopt.com/static/pdf/SigOpt_Bayesian_Optimization_Primer.pdf. (n.d.)

Ding et al. 2013] Ding, S., Li, H., Su, C., Yu, J., & Jin, F.: Evolutionary artificial neural networks: A review. *Artificial Intelligence Review*, (1-10). (2013).

[Elbeltagi et al. 2005] Elbeltagi, E., Hegazy, T., & Grierson, D.: Comparison among five evolutionary-based optimization algorithms. *Advanced engineering informatics*, *19*(1), (43-53). (2005).

[Floreano & Mattiussi 2008] Floreano, D., & Mattiussi, C.: Bio-Inspired Artificial Intelligence - Theories, Methods, and Technologies (2008).

[Girshick 2015] Girshick, R. (2015). Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)*. (1440-1448). (2015).

[Girshick et al. 2014] Girshick, R., Donahue, J., Darrell, T., & Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 580–587). (2014).

[Gong et al. 2014] Gong, Y., Wang, L., Guo, R., and Lazebnik, S.: Multi-scale orderless pooling of deep convolutional activation features. In *Proceedings of the European Conference on Computer Vision* (392–407). (2014).

[Hamze et al. 2013] Hamze, F., Wang, Z., & de Freitas, N. (2013). Self-avoiding random dynamics on integer complex systems. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, *23*(1). (2013).

[Hinton et al. 2012] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R.: Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv Preprint arXiv:1207.0580*. (2012)

[Iandola et al. 2016] Iandola, F. N., Moskewicz, M. W., Ashraf, K., Han, S., Dally, W. J., and Keutzer, K.: SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *arXiv 1602.07360*. (2016).

[LeCun et al. 1998] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P.: Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), (2278-2324). (1998).

[Lessmann et al. 2005] Lessmann, S., Stahlbock, R., & Crone, S. F.: Optimizing hyperparameters of support vector machines by genetic algorithms. In *IC-AI* (74-82). (2005).

[Loshchilov & Hutter 2016] Loshchilov I., & Hutter, F.: CMA-ES for Hyperparameter Optimization of Deep Neural Networks. *arXiv preprint arXiv:1604.07269*. (2016).

[Mallat 2012] Mallat, S.: Group invariant scattering. *Communications on Pure and Applied Mathematics, 65*(10). (2012).

[Martino et al. 2011] Martino, S., Ferrucci, F., Gravino, C., & Sarro, F.: A genetic algorithm to configure support vector machines for predicting fault-prone components. In *International Conference on Product Focused Software Process Improvement* (247-261). (2011).

[Miikkulainen et al. 2017] Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., & Hodjat, B.:. Evolving Deep Neural Networks. *arXiv preprint arXiv:1703.00548*. (2017).

[Mockus et al. 1978] Mockus, J., Tiesis, V., & Zilinskas, A.: The Application of Bayesian Methods for Seeking the Extremum. In *Towards Global Optimization*, vol. 2. (117-128). (1978).

[Orive et al. 2014] Orive, D., Sorrosal, G., Borges, C.E., Martín, C., & Alonso-Vicario, A.: Evolutionary algorithms for hyperparameter tuning on neural networks

models. In *Proceedings of the 26th European modelling & simulation symposium.* (402-409). (2014).

[Pedregosa et al. 2012] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., & Vanderplas, J.: Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, (2825-2830). (2012).

[Ranzato et al. 2007] Ranzato, M. A., Huang, F. J., Boureau, Y., & LeCun, Y.: Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 1-8. (2007).

[Rasmussen & Williams 2006] Rasmussen, C. E., & Williams, C. K.: Gaussian processes for machine learning, (vol. 1). Cambridge: MIT press. (2006).

[Rawat & Wang 2017] Rawat, W., Wang, Z.: Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation, 29*(9), (2352-2449). (2017).

[Shahriari et al. 2016] Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., & de Freitas, N.: Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, *104*(1), (148-175). (2016).

[Snoek et al. 2012] Snoek, J., Larochelle, H., & Adams, R. P.: Practical Bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems, 25* (2951–2959). (2012).

[Snoek et al. 2012] Snoek, J., Larochelle, H., & Adams, R. P.: Practical Bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems, 25* (2951–2959). (2012).

[Suong & Jangwoo 2018] Suong, L.K. & Jangwoo, K.: Detection of Potholes Using a Deep Convolutional Neural Network. *J. UCS*, *24*(9), (1244-1257). (2018).

[Swersky et al. 2013] Swersky, K., Snoek, J., & Adams, R. P.: Multi-task Bayesian optimization. In *Advances in neural information processing systems* (2004-2012). (2013).

[Tao et al. 2007] Tao, J., Wang, N. & Wang, X.: Genetic Algorithm Based Recurrent Fuzzy Neural Network Modelling of Chemical Processes. *J. UCS*, *13*(9), (1332-1343). (2007).

[Vinyals et al. 2015] Vinyals, O., Toshev, A., Bengio, S., & Erhan, D.: Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (3156–3164). (2015).

[Wiatowski & Bolcskei 2015] Wiatowski, T., Bolcskei, H.: A mathematical theory of deep convolutional neural networks for feature extraction. *arXiv 1512.0629*. (2015).

[Xie et al. 2017] Xie, L., & Yuille, A.: Genetic CNN. arXiv preprint. *arXiv preprint arXiv:1703.01513*. (2017).