algorithms

MDPI

Article

Grammatical Evolution-Driven Algorithm for Efficient and Automatic Hyperparameter Optimisation of Neural Networks

Gauri Vaidya *,†, Meghana Kshirsagar † and Conor Ryan *

Biocomputing and Developmental Systems Research Group, University of Limerick, Limerick V94 T9PX, Ireland; meghana.kshirsagar@ul.ie

* Correspondence: gauri.vaidya@ul.ie (G.V.); conor.ryan@ul.ie (C.R.)

† These authors contributed equally to this work.

Abstract: Neural networks have revolutionised the way we approach problem solving across multiple domains; however, their effective design and efficient use of computational resources is still a challenging task. One of the most important factors influencing this process is model hyperparameters which vary significantly with models and datasets. Recently, there has been an increased focus on automatically tuning these hyperparameters to reduce complexity and to optimise resource utilisation. From traditional human-intuitive tuning methods to random search, grid search, Bayesian optimisation, and evolutionary algorithms, significant advancements have been made in this direction that promise improved performance while using fewer resources. In this article, we propose HyperGE, a two-stage model for automatically tuning hyperparameters driven by grammatical evolution (GE), a bioinspired population-based machine learning algorithm. GE provides an advantage in that it allows users to define their own grammar for generating solutions, making it ideal for defining search spaces across datasets and models. We test HyperGE to fine-tune VGG-19 and ResNet-50 pre-trained networks using three benchmark datasets. We demonstrate that the search space is significantly reduced by a factor of 90% in Stage 2 with fewer number of trials. HyperGE could become an invaluable tool within the deep learning community, allowing practitioners greater freedom when exploring complex problem domains for hyperparameter fine-tuning.

Keywords: search space pruning; machine learning; grammatical evolution; combinatorial optimisation; computer vision; metaheuristics; hyperparameter tuning; deep learning; neural networks

Citation: Vaidya, G.; Kshirsagar, M.; Ryan, C. Grammatical Evolution-Driven Algorithm for Efficient and Automatic Hyperparameter Optimisation of Neural Networks. *Algorithms* **2023**, *16*, 319. https://doi.org/10.3390/a16070319

Academic Editor: Ioannis Tsoulos

Received: 31 May 2023
Revised: 24 June 2023
Accepted: 25 June 2023
Published: 29 June 2023

1. Introduction

Recent years have seen neural networks (NNs) become increasingly popular in a wide variety of industries and commercial applications, from image classification [1], natural language processing [2], predictions, forecasting [3], and machine translation [4]. As research into NNs continues at an ever-increasing rate, efforts are being made to improve the performance of these models, which are highly impacted by the combination of parameters and hyperparameters. Model parameters (e.g., weights and biases) are learned by the model during training, while hyperparameters (e.g., learning rate) control how well these parameters are learned. Hyperparameters can be divided into two broad categories—those related to model design such as the number of hidden layers or activation functions used, and those related to the learning process such as an optimiser, learning rate, or momentum. The selection of hyperparameters has a significant impact on the performance of a model, as it determines how well it generalises to unseen or new data [5]. Figure 1 illustrates this by showing the sensitivity of the model's performance against hyperparameter configurations of a convolutional neural network (CNN) model pre-trained with VGG-19. The model is trained for five epochs on a benchmark dataset, `cifar10`, with two different batch sizes with varying learning rates. The graph depicts the impact of learning rate and batch sizes on model accuracy. This indicates the need of the optimal selection of hyperparameters

that can enhance the performance of NNs.

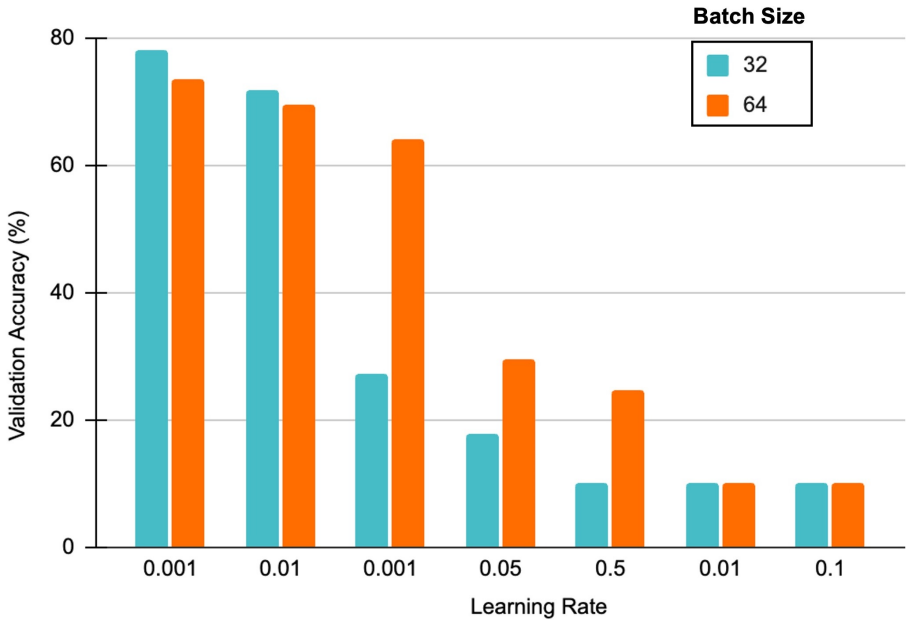


Figure 1. An example illustrating the impact of hyperparameter configurations (batch sizes and learning rates) on model performance. The graph depict the model performance over five epochs with a pre-trained VGG-19 model on `cifar10` dataset.

Hyperparameter tuning, or hyperparameter optimisation (HPO), is the process of selection of hyperparameter values resulting in improving the model performance while reducing the computational resources. Throughout the article, we refer to *computational resources* in HPO as the number of trials, the size of the hyperparameter search space, or computational time. Here, one trial refers to one run of NN with a hyperparameter configuration set. The traditional approach for HPO has been to use the *trial-and-error* method, which is further validated by domain experts [6], i.e., selecting configurations based on the knowledge and experience of domain experts [7]. However, this ad hoc method does not always guarantee optimal results and can become resource-intensive with large NNs.

Automation of HPO follows a structured process as shown in Figure 2. HPO aims at finding the optimal configurations from a given search space by iteratively training NNs on a given dataset while incorporating feedback from previous iterations until either budget is exhausted or desired results are achieved. Traditional HPO methods such as random search [8] and grid search [9] have been successful at obtaining optimal or sub-optimal performance compared to human-derived configurations; however, these approaches become highly cumbersome with larger models and increasing complexity. Recent research has shown that Bayesian optimisation (BO) is one of the most widely used methods for industrial applications; however, it is limited by the curse of dimensionality when dealing with high-dimensional deep learning models [10]. This complexity can be further compounded due to its $O(n^3)$ complexity [11], where n is the search space of hyperparameter sets. To address this issue, evolutionary algorithms (EAs) or population-based training (PBT) has been increasingly used in recent years as an ideal solution for considering such high-dimensional spaces due to their ability to balance exploration (efficient global search) and exploitation (efficient local search).

In this work, we attempt to tackle this problem of reducing the complexity in HPO with *HyperGE*, a grammatical evolution (GE)-driven model. GE is a bio-inspired, machine-learning-based EA, with the flexibility of automatically generating computer programs with the Backus–Naur form (BNF) grammar. We propose HyperGE, a HPO model with

a two-stage grammar-guided search space approach. In Stage 1 of HyperGE, the defined hyperparameter search space is efficiently explored globally by the individuals, while in Stage 2, the intuitions from Stage 1 further guide exploitation within the reduced search space to attain optimal configurations. In addition, we harness parallelisation with HyperGE by allowing for knowledge transfer among individuals within the population. The proposed approach is supported by empirical evidence from extensive testing on three benchmark datasets and comparative analysis with state-of-the-art (SOTA) demosntrates its effectiveness in finding the best possible solutions with minimal resources.

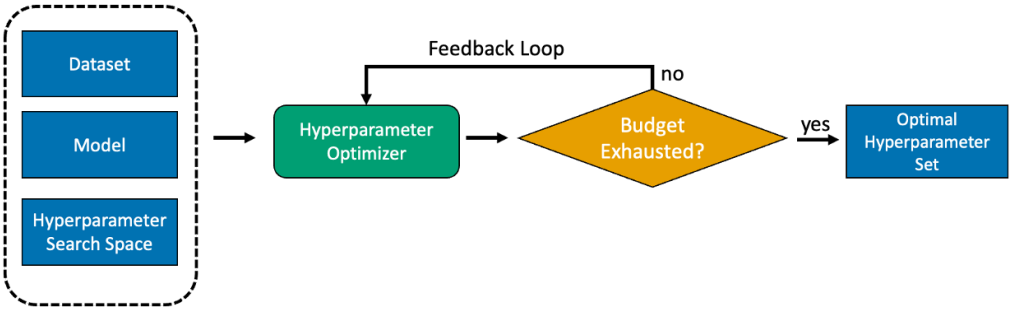


Figure 2. Process flow of hyperparameter optimisation.

The rest of the article is organised as follows. We introduce the theoretical background of CNNs and mathematical optimisation in Section 2. Subsequently, we discuss the recent approaches used in the literature to reduce the computational complexity in Section 3. Our proposed technique is presented in detail under Section 4, while Sections 5 and 6 present our experimental setup with findings for our hypothesis. Finally, we conclude by highlighting the future scope of research related to this topic in Section 7.

2. Background

In this section, we discuss some theoretical foundations and introduce the hyperparameter optimisation problem.

2.1. Convolutional Neural Networks

CNNs have revolutionised the field of image-related tasks, such as image classification [12], segmentation, and object detection [13]. A typical CNN architecture consists of convolutional layers that are able to learn features from the input dataset followed by fully connected layers for learning a specific task . The number of layers and their corresponding parameters may vary depending on the dataset size and budget constraints. Designing custom CNNs for a given task has been extensively researched in recent years; however, with the introduction of transfer learning in 2012 [14], this research area has seen an increased acceleration due to its ability to use knowledge transfer obtained from one task onto another, called *transfer learning*, and the models used for transfer learning are called *pre-trained models*. Transfer learning is inspired by the working of human brains which can use acquired knowledge from one domain into another related domain without having to start again from scratch each time. Figure 3 further illustrates this concept in detail. Larger models (with layers ranging from as small as 19 to 500) are trained on a large dataset (e.g., ImageNet [15] with 14 million instances). These pre-trained models can then be used for a different dataset with fewer instances by sharing the common convolutional layers and by tuning their fully connected layers. Thus, the process can be summarised in two steps: (1) knowledge transfer from an already trained model; and (2) customising the fully connected layers for the given dataset, known as *fine-tuning*. Using such pre-trained models is highly beneficial in real-world applications due to their ability to perform well even with smaller datasets thus saving computational resources. Transfer learning also significantly reduces the carbon footprint as compared to training models from scratch. Hence, fine-tuning the pre-trained models is more in practice for real-world applications, thus highlighting fine-tuning as a significant research direction.

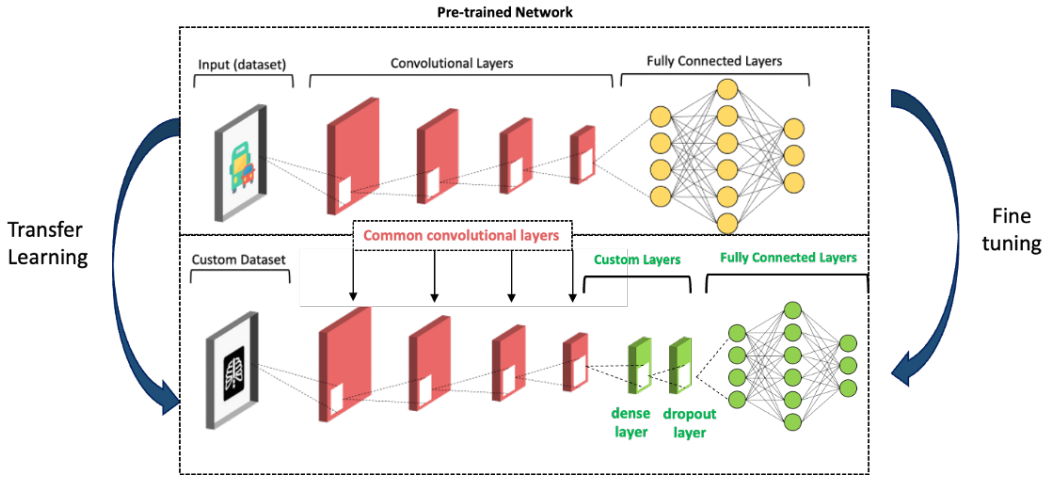


Figure 3. Common topology of transfer learning in a convolutional neural network. The features learned from a dataset can be transferred to a new dataset by sharing the convolutional layers and by fine-tuning the fully connected layers.

2.2. Mathematical Optimisation

Hyperparameter optimisation falls under the umbrella of mathematical optimisation, which can be broadly defined as the process of maximising or minimising an objective function given a set of candidates (solution alternatives). The search space of the objective function is determined by the decision variables, x, which can take any value between a given range for all real numbers, $\mathbb{R}$ in one-dimensional space. Unconstrained optimisation is then defined as finding the maximum/minimum value of an objective function without imposing any restrictions on its solution (i.e., within all possible values). Generally, unconstrained optimisation can be defined as:

$$\min_{x \in \mathbb{R}} f(x), \tag{1}$$

where $f(x)$ is the objective function to minimise.

However, in real-world applications, the constraints of the decision variables are generally known, and hence, Equation (1) then becomes:

$$\min_{x \in \mathbb{X}} f(x), \tag{2}$$

$$a_i(x) \leq 0,\ i = 1, 2, \ldots, m$$
$$b_j(x) = 0,\ j = 1, 2, \ldots, n$$

where a_i are the inequality constraint functions and b_j is the equality constraint functions over the constrained domain X of x. The constraints define the boundaries for the search space of hyperparameters and thus lead to a feasible region, F, where the solution exists, which can be denoted by:

$$F = \{x \in X | a_i(x) <= 0,\ b_j(x) = 0\} \tag{3}$$

Thus, the goal of mathematical optimisation is to find a global optimum of $f(x)$ given the set of X in the feasible region F. However, a global optimum of an objective function can only be found if the objective function, $f(x)$, is a convex function and X is a convex set over the domain F as follows:

$$\min_{x \in \mathbb{X}} f(x), \tag{4}$$

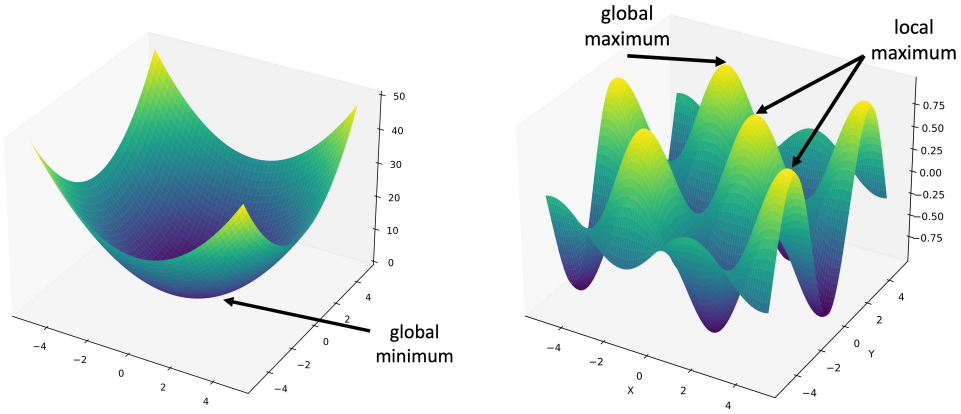
For time t,

$$\text{as } t \to \infty, \tag{5}$$

$$f(x_t) - f(x^*) \to 0 \tag{6}$$

where $f(x_t)$ is the objective function at time t and $f(x^*)$ is the optimal objective value.

However, NNs have an objective function that is non-convex and therefore has multiple local optimums. As illustrated in Figure 4, the optimiser still can get stuck in one of these local optima. As such, numerical optimisation approaches based on gradient descent are used to minimise the objective function within a given search space; however, this can become computationally intensive when dealing with large search spaces. To address this issue, derivation-free algorithms such as heuristics have been developed to help discover optimal solutions more efficiently.



(**a**) (**b**)

Figure 4. Nature of optimal solutions in (**a**) convex functions with a single optimum; (**b**) non-convex functions with multiple local optimums.

2.3. Hyperparameter Optimisation

Let H be the set of all possible values for n hyperparameters $\{h1, h2, h3, \ldots hn\}$, then the search space for the hyperparameters would be n-dimensional with different domains for each dimension (e.g., real-valued numbers for learning rate, choices for optimisers and so on). Given a hyperparameter configuration set $h*$ over the domain H, a NN model, m, is trained for a given dataset d, and its prediction performance is measured on the unseen dataset. Thus, the objective f for HPO is to maximise the accuracy of the m on unseen data with a given d. H in minimal trials and can be formulated as:

$$h* = \arg\max_{h \in \mathbb{H}} f(h, d, m), \tag{7}$$

subject to

$$\arg\min(trial)$$

3. Related Works

Since 2010, there has been an upward trend of exploring HPO techniques as illustrated in Figure 5. Until 2015, traditional techniques such as random search and grid-based were widely used, leading to BO as one of the most prominent techniques. After 2015, there was an increase in the research trend involving more advanced methods such as EA while still maintaining the use of BO, which has been improved upon over time. There are now a number of Python libraries supporting HPO, including Hyperopt [16], Optuna [17], Ray [18], Scikit-Optimise, and Sigopt [19], which support various types of HPO techniques such as random search, grid-search, BO, Hyperband, tree structured Parzen estimators (TPE) [17], Hyperband [20], PBT [21], etc. Hyperband is an extension to random search,

with a unique approach and a restricted time budget to the search configurations, which has been proven as effective as BO. TPE is also a form of sequential modeling, like BO, which works on the principle of approximation based on the previous results. All of these techniques have their own pros and cons as discussed in this review article in detail [22], but the research question that the research community is still focusing on is: *how can we make these techniques less resource-extensive and more efficient?*

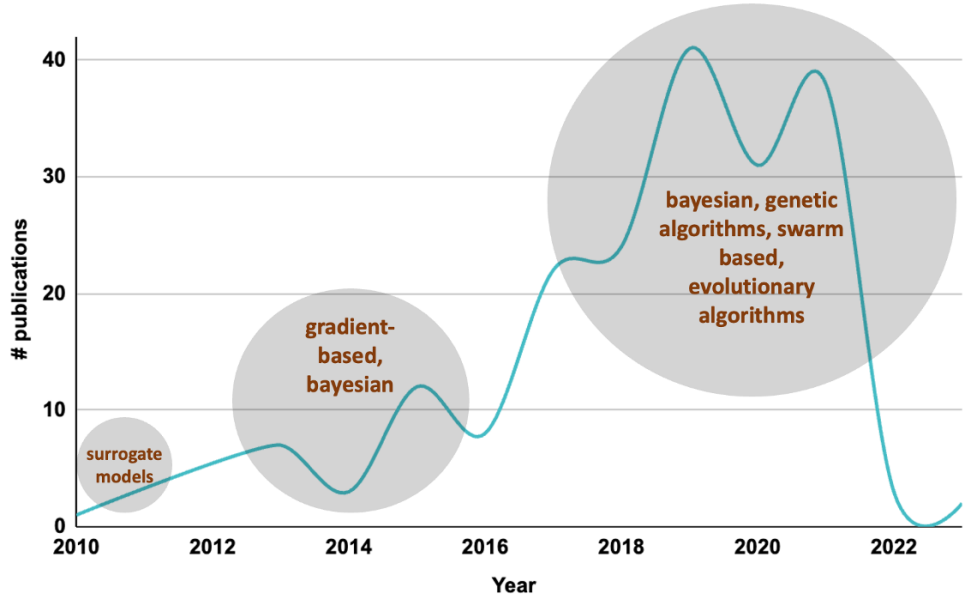


Figure 5. Research trends of hyperparameter optimisation of neural networks from 2010. Results are based on Google Scholar with keywords "hyperparameter, optimisation, neural, networks".

In recent years, there has been extensive research to answer the above research questions. In 2016, Springenberg et al. [23] proposed an efficient way of reducing the complexity of BO by introducing a single and multi-task optimiser with a stochastic gradient Hamiltonian Monte Carlo method. This approach was found to be scalable and efficient on the `cifar10` dataset when applied on 32-layer Resnet network architecture with a test error of 7.4%. Similarly, in the same year, Lévesque et al. [24] presented one such similar approach for multi-task optimisation which included hyperparameter tuning transferring knowledge learned during HPO from one task to another while achieving an 18% generalisation error on the `cifar10` dataset for fine-tuning tasks.

Albeit these techniques have shown improvements in resource usage, they still suffer from limitations in time as well as performance because of their sequential nature. A number of parallelization approaches to improve performance in terms of time have also been proposed. However, the idea of EAs, such as learning together as a population, adds an extra advantage. One such technique was proposed in 2017 with the introduction of the PBT [21]. In PBT, the hyperparameters are tuned while the model is being trained simultaneously across the population, leading to an effective distribution of parallelization. This technique has been proven effective, especially for reinforcement learning, and can be effectively used for other NNs as well. Another study in 2017 proposed derivation-free implementation with a radial basis function [5] and it has proven to be impactful in tuning the hyperparameters on a CNN model with two layers on the MNIST digit recognition dataset. In the following years, EAs have been explored further for hyperparameter tuning [7,25], specifically genetic algorithms [26,27] and swarm optimisation [28].

In 2020, Basha et al. proposed AutoFCL [29], a BO framework for fine-tuning pre-trained neural networks for image classification which was limited to fully connected layers. This limitation was addressed in 2021 by the same research group with AutoTune [30]

which considered dynamic layers and proved more effective than existing methods on some datasets. The results of this study highlight the importance of considering different layer types when using pre-trained models and demonstrate how performance can be improved through such an approach.

To this effect, in 2022, we [31] proposed HyperEstimator [31], a HPO model driven by GE, machine learning, and BO. However, the search space of the HyperEstimator was only restricted to three hyperparameters which are not optimal in the real world [30]. Moreover, using a suite of algorithms (regression and BO with EA) makes it a complex model. Hence, we extend HyperEstimator by incorporating all the hyperparameters for fine-tuning and propose a two-stage technique for HPO by eliminating the BO and regression model from the HyperEstimator. GE, due to its flexibility in defining the search space with grammar, supports the definition of all types of hyperparameters including integer, float, and string values, making it suitable for HPO tasks. Moreover, HPO with GE applied to NNs is a novel approach to the best of the author's knowledge.

4. Materials and Methods

In this section, we briefly introduce our approaches to reduce the computational complexity of HPO by incorporating GE.

4.1. Grammatical Evolution

GE [32,33] is a bio-inspired machine learning technique that generates computer programs by using formal grammar to map binary strings into program structures. This mapping separates the genotype (binary string) from the phenotype (executable program or expression tree), making it possible to incorporate grammar constraints in the search procedure. GE has been used in various applications, such as parameter optimisation for time series modeling [34], cryptography [35], etc., due to its flexibility in defining search spaces using BNF grammar.

In GE, the boundaries of the search space are defined with BNF grammar, and the genetic algorithm (GA) is used to traverse the search space. The evolutionary process begins with the initialization of a population. In each iteration, known as a *generation*, each individual in the population is mapped into a program or expression using the BNF grammar and is tested using an objective function (fitness function). The population is then evaluated based on fitness score, and genetic operators of selection, crossover, and mutation are applied to produce offspring for the next generation. This process continues until one of the termination criteria is satisfied, e.g., the population has evolved for the defined number of generations or the desired fitness is achieved. The best individual is the one that achieves the maximum/minimum fitness. Figure 6 provides an overview of the workings of GE with GA.

4.2. Mapping of Genotype to Phenotype

GE uses an eight bit integer codon string to guide its mapping process in GE by determining the choice of appropriate production rule from the BNF grammar. The mapping of the integer codons begins with a start symbol **S**, with the expansion of the non-terminal symbols, **N**, into terminal symbols, **T**, dictated by production rules **P**, resulting in a complete syntax. Thus, BNF grammar can be represented by a tuple $< S, N, T, P >$. Figure 7 shows the mapping process with a sample BNF grammar consisting of some terminals and non-terminals. The genotype-to-phenotype mapping is controlled by the following equation:

$$\text{next choice} = (\text{integer codon}) \bmod (\#\text{choices}) \tag{8}$$

The value from the equation dictates the selection of the next production rule. For example, when considering S, `<finetuning>`, as it has only one choice, the mapping module will choose the next first rule `<batch_size>`. For mapping `<batch_size>`, the integer codon is 204, and the number of available choices is four. The result would be 0 by Equation (8), which results as `<batch_size>::= 32`, the first value from the available choices. Note that

the beginning integer is 0, and hence, the first choice is mapped. The process repeats until all the non-terminals are mapped, and an expression composed entirely of terminals remains.

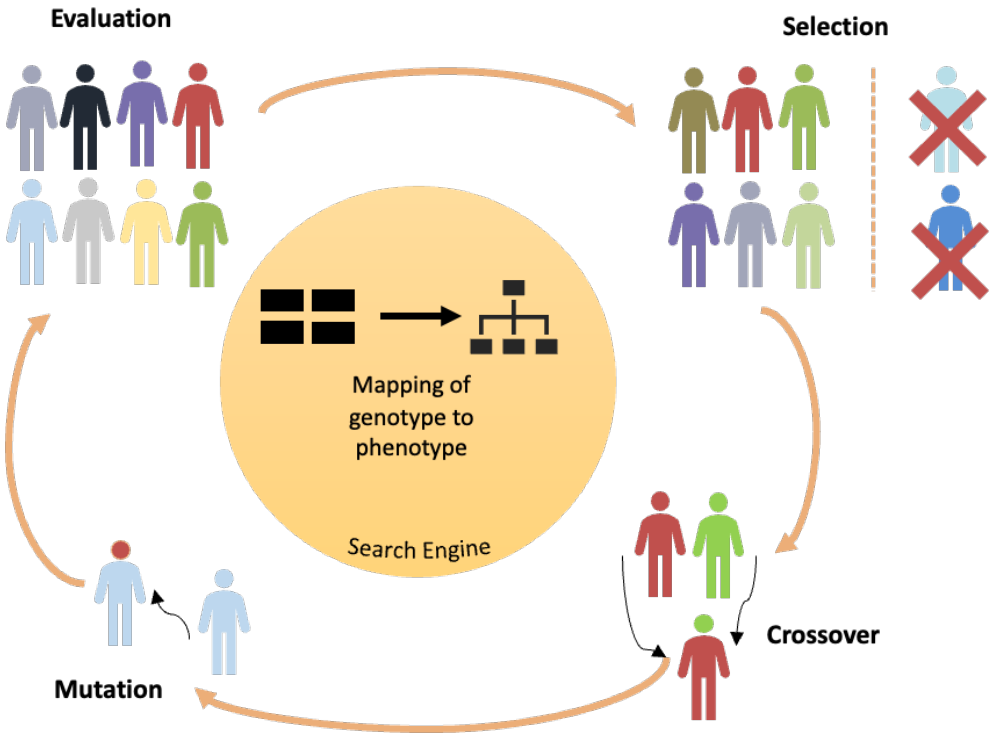


Figure 6. Working of GE. The chromosome encodes the genotype, which is mapped to a derivation tree, which in turn becomes either a program or expression, depending on the BNF used.

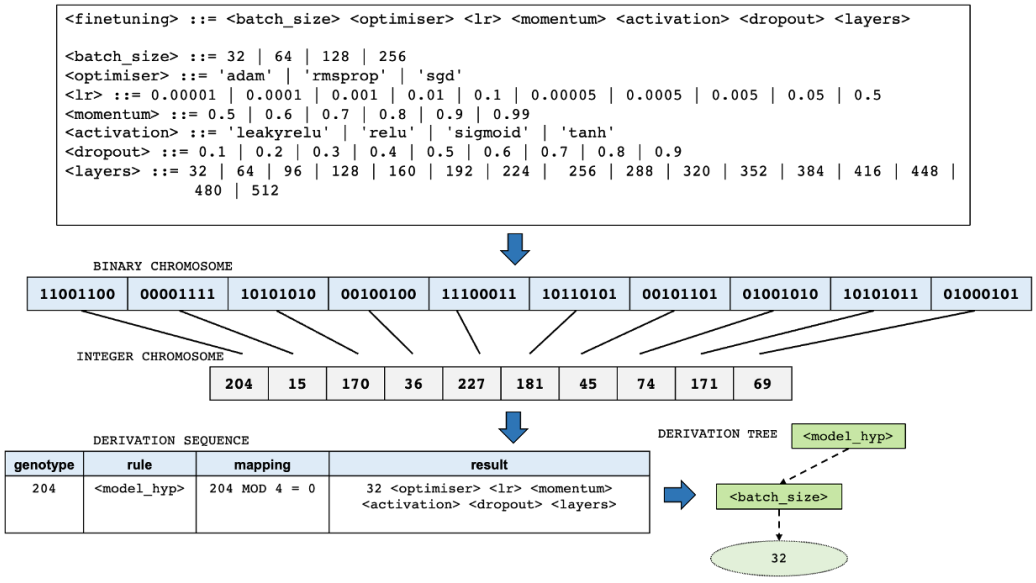


Figure 7. Mapping of genotype to phenotype with BNF grammar with an example.

4.3. Overview of HyperGE

HyperGE is a two-stage GE-driven HPO model which automatically fine-tunes the hyperparameters for a given NN architecture and dataset. In Stage 1, the model and dataset are fed to the HyperGE framework. The hyperparameter search space is defined in the BNF grammar. The population of individuals then evolves until the number of generations are reached. The statistics of the individuals obtained from Stage 1 are then analysed to reduce the search space. The refined grammar is then used again to evolve the individuals until

the termination criteria is met. This data-driven approach leads to a significant reduction in search space. obtaining optimal hyperparameter configurations in a minimal number. of trials. Moreover, as GE is a population based technique, the learned information of the individuals is carried over to the next generations (Figure 6), exploiting the promising regions while also encouraging the exploration thus improving the performance over the generations. An overview of HyperGE is illustrated in Figure 8.

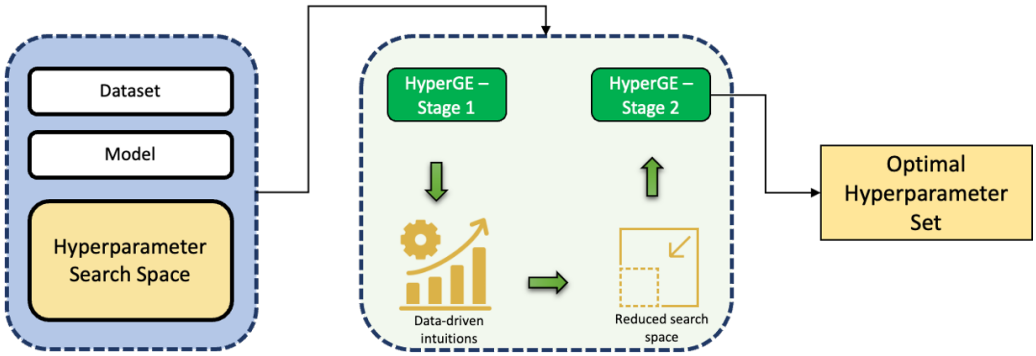


Figure 8. Overview of HyperGE framework.

4.4. Hyperparameter Search Space

The hyperparameter search space in our proposed approach consists of six hyperparameters: *{batch size, optimiser, learning rate, momentum, dropout rate, and number of hidden units in the dense layer}* for fine-tuning the last layer of the pretrained models. Table 1 shows the six hyperparameters and all the possible values they can take. All the possible values for the hyperparameters are referred to from the standard recommendations [36]. The number of hidden units in denser layers is in the range [32, 512] with an increment of 32. Dropout rates vary within [0.1, 0.9] with an increment of 0.1. The choices for optimisers are from three different families of optimisers, viz., stochastic gradient (sgd), Ada (adam), and root mean square propagation (rmsprop). The learning rate can take values of [0.00001, 0.1] with an exponential step of 10. The range of momentum varies of [0.5, 0.9] while batch sizes are in the range [32, 256]. Column 4 of Table 1 shows the possible combinations of each of these hyperparameters. The size of the hyperparameter search would thus be 102,680 (product of number of combinations from Column 4). This search space is represented with the BNF grammar defining all the possible choices for each hyperparameter as illustrated in Figure 9. Every dataset and model starts with Stage 1 grammar which is then further refined according to the insights obtained after Stage 1 exploration. Thus, Stage 2 grammar would be unique for each dataset and model combination.

```
<finetuning> ::= <batch_size> <optimiser> <lr> <momentum> <dropout> <layers>

<batch_size> ::= 32 | 64 | 128 | 256

<optimiser> ::= 'adam' | 'rmsprop' | 'sgd'

<lr> ::= 0.00001 | 0.0001 | 0.001 | 0.01 | 0.1 | 0.00005 | 0.0005 | 0.005 | 0.05 | 0.5

<momentum> ::= 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 0.99

<dropout> ::= 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9

<layers> ::= 32 | 64 | 96 | 128 | 160 | 192 | 224 |  256 | 288 | 320 | 352 | 384 | 416
           | 448 | 480 | 512
```

Figure 9. BNF grammar of Stage 1 in HyperGE.

Table 1. Hyperparameters employed in NNs with their scopes: *choice/categorical* type indicates the options that the hyperparameter can take, while ranges define any possible value between the given range.

Hyperparameter	Type	Scope	#Combinations
# dense layers	Integer	{32, 64, 96, 128, 160, 192, 224, 256, 288, 320, 352, 384, 416, 448, 480, 512}	16
dropout rate	Decimal	{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}	9
optimiser	Categorical	{adam, sgd, rmsprop}	3
learning rate	Decimal	{0.00001, 0.0001, 0.001, 0.01, 0.1, 0.00005, 0.0005, 0.005, 0.05, 0.5}	10
momentum	Decimal	{0.5, 0.6, 0.7, 0.8, 0.9, 0.99}	6
batch size	Choice	{32, 64, 128, 256}	4
Total combinations			**103,680**

5. Experimental Setup

We performed extensive experimentation to test our hypotheses and present an empirical analysis in this section.

5.1. NN Architectures

We tested HyperGE on two widely used pre-trained networks, viz., VGG-19 [37] and ResNet-50 [6]. VGG-19 has 19 layers, while ResNet-50 is a larger network with 50 layers of convolutions. Both of these models are pre-trained on the ImageNet benchmark dataset for image classification with 14 million instances, making them an ideal choice for transfer learning tasks. The intuition behind choosing these models was twofold: (1) they both are from different families of pre-trained networks; (2) they have a different number of layers, specifically ResNet-50 having almost double the amount of layers as that of VGG-19.

5.2. Datasets

We present the empirical analysis on three benchmark datasets for image classification, viz., `cifar10` [38], `cifar100` [38], and `caltech101` [39]. These datasets have varying sizes, classes, instances, and domains amongst them. All the classes in each of the datasets were balanced and were used without any preprocessing or augmentation. For all datasets, the ratio of train–test–validation was 80–20–20%, respectively. While `cifar10` and `cifar100` are subsets of ImageNet, `caltech101` has significantly different classes compared to ImageNet, validating how well the approach generalises.

5.3. Evolutionary Hyperparameters

All the evolutionary hyperparameters used in the experimentation are presented in Table 2. The budget of the experiments, viz., the number of trials, is controlled by the number of generations and population size as $total(trial) = size(pop) * size(gen)$. In our experiments, the budget was restricted to 50 and 30 trials, respectively, in Stage 1 and Stage 2, with a total budget of 80 trials as observed from Table 2. The exploration and exploitation are controlled by the crossover rate (high) and mutation rate (low). For image classification tasks, top 1%-accuracy is an appropriate method to measure the performance of a model when datasets are balanced. This metric measures the number of correct predictions from the total predictions on a dataset. When dealing with unbalanced datasets, other fitness functions such as F1 score, precision, or recall can be used instead. In our empirical study, we opted for the top 1%-accuracy as our fitness score for validating data (as the chosen datasets are balanced) as shown in Equation (9).

$$\text{top 1\% accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions}} \tag{9}$$

Table 2. Evolutionary hyperparameter settings for Stage 1 and Stage 2 in HyperGE.

Evolutionary Hyperparameters	Values	
	HyperGE-Stage 1	**HyperGE-Stage 2**
Population size	10	10
Number of generations	5	3
Search Engine	Genetic Algorithm	Genetic Algorithm
Crossover Type	Variable One-Point	Variable One-Point
Mutation Type	Integer Flip Per Codon	Integer Flip Per Codon
Crossover probability	0.95	0.95
Mutation probability	0.01	0.01
Selection Type	Tournament	Tournament
Initilialisation method	Position Independent grow	Position Independent grow

6. Results and Discussion

This section discusses the comparative analysis of our proposed approach with SOTA and addresses our research questions with empirical analysis. We adapted PonyGE2 [40], a GE implementation in Python, to run all the experiments with the Pytorch framework. All experiments were conducted simultaneously on high-performance computing platforms-Intel Xeon Silver 4215R CPU @ 3.20 GHz with Quadro RTX 8000 GPU.

6.1. Refining the Search Space with HyperGE

As the objective of HyperGE is to reduce the computational complexity of the HPO, we aim at reducing the resources used in the experimental settings. Baldominas et al. [41] and Vaidya et al. [31] have shown that while fine-tuning the models, a small subset of the dataset is enough to learn the hyperparameters. Hence, we followed the same settings and used 5% of the training set of the dataset during HPO. However, the validation accuracy, viz., the fitness function, was calculated on the entire validation set, making it a robust hyperparameter learner. For each trial, the subset of 5% of data was trained over five epochs with the configuration, and validation accuracy over the entire validation set was measured as metrics.

With the budget of Stage 1 (50 trials), the insights of the individuals' performance were obtained and plotted in Figure 10. The plot shows the distribution of performance of individuals across the hyperparameters for VGG-19 and ResNet-50 across all three datasets. It is evident that although the search space and dataset are the same, the performance of the hyperparameters is still very different across the models (e.g., batch size for `cifar10`). Hence, the choice of model, hyperparameters, and datasets is equally significant in HPO.

Using the insights obtained from these plots, the least-performing hyperparameters (with validation accuracy < 25%) were eliminated and a refined search space was designed for each model across the datasets as illustrated in Figure 11. In Stage 2 of HyperGE, the models are further fine-tuned using the refined grammar for the budget of 30 trials. The best-performing individual among these 30 trials (the individual with the highest validation accuracy) is then trained for 50 epochs using the hyperparameters.

Figure 12 illustrates the performance of individuals across Stage 1 and Stage 2 against the number of trials. From the graphs, it can be seen that in Stage 1, the individuals have more variance and have a global explorative nature. This is helpful for them not to get stuck in local minima and to learn all the hyperparameters. In Stage 2, with the refined search space, the variance of the performance is significantly reduced, and it can be seen that the accuracy of the individuals lies within a certain range thus guiding the search. The values of mean and standard deviation of the fitness values are presented in Table 3. The average reduction in search space with Stage 2 of HyperGE is 90%, as presented in the last column of Table 3, showing the potential of HyperGE in reducing the computational resources.

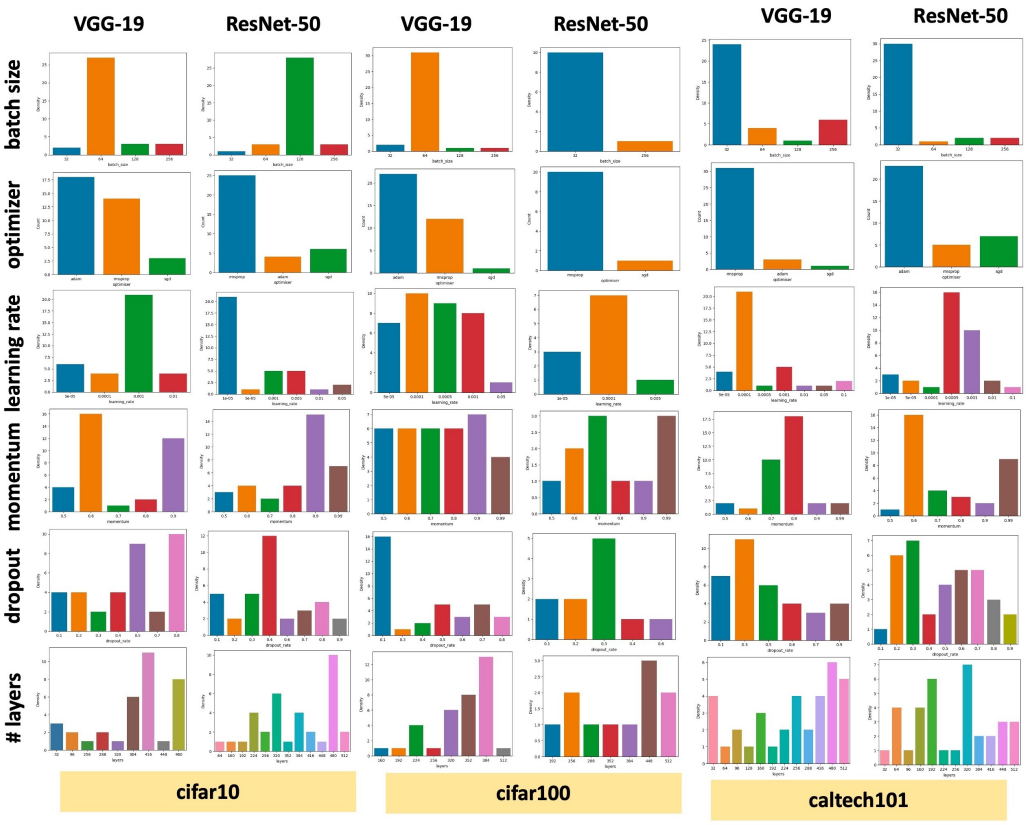


Figure 10. Data-driven analysis of exploration in Stage 1 of HyperGE across all datasets.

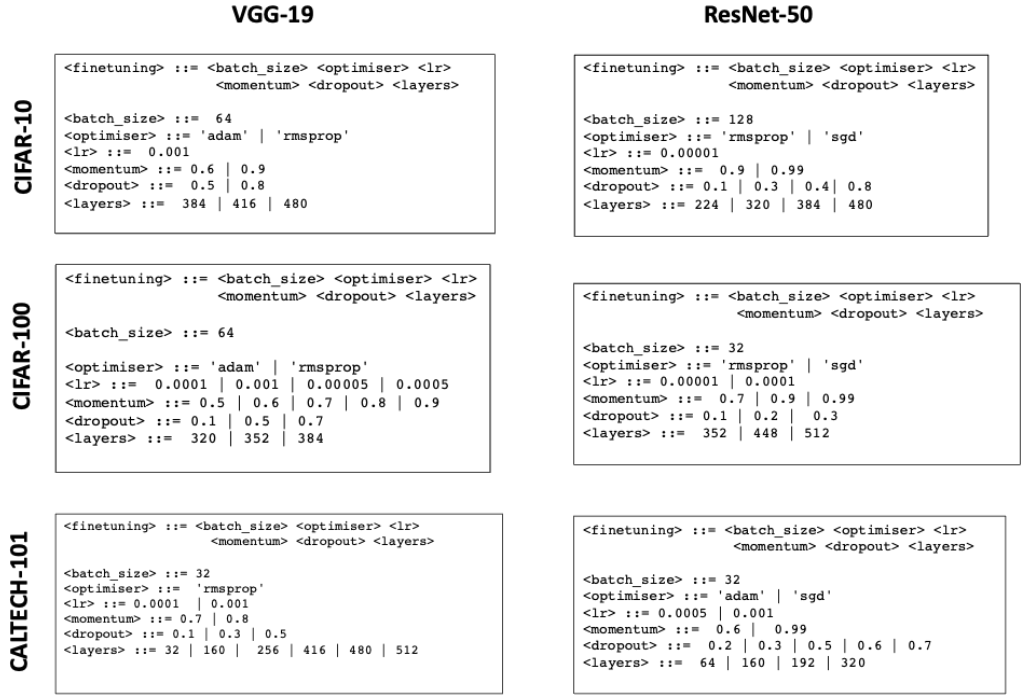


Figure 11. Refined search space of VGG-19 and ResNet-50 models across datasets in Stage 2 of HyperGE.

Table 3. Statistical analysis of HyperGE stages with mean and standard deviation.

Dataset	VGG-19				ResNet-50				Reduction in Search Space
	Fitness (Stage 1)		Fitness (Stage 2)		Fitness (Stage 1)		Fitness (Stage 2)		
	Mean (VA %)	std	Mean (VA %)	std	Mean (VA %)	std	Mean (VA %)	std	
`cifar10`	59.02	13.93	76.75	9.27	53.29	25.05	65.50	8.52	∼90%
`cifar100`	13.05	15.45	41.83	6.57	10.18	11.57	22.72	6.81	∼90%
`caltech101`	41.60	23.27	62.79	6.95	26.07	13.93	23.60	9.27	∼90%

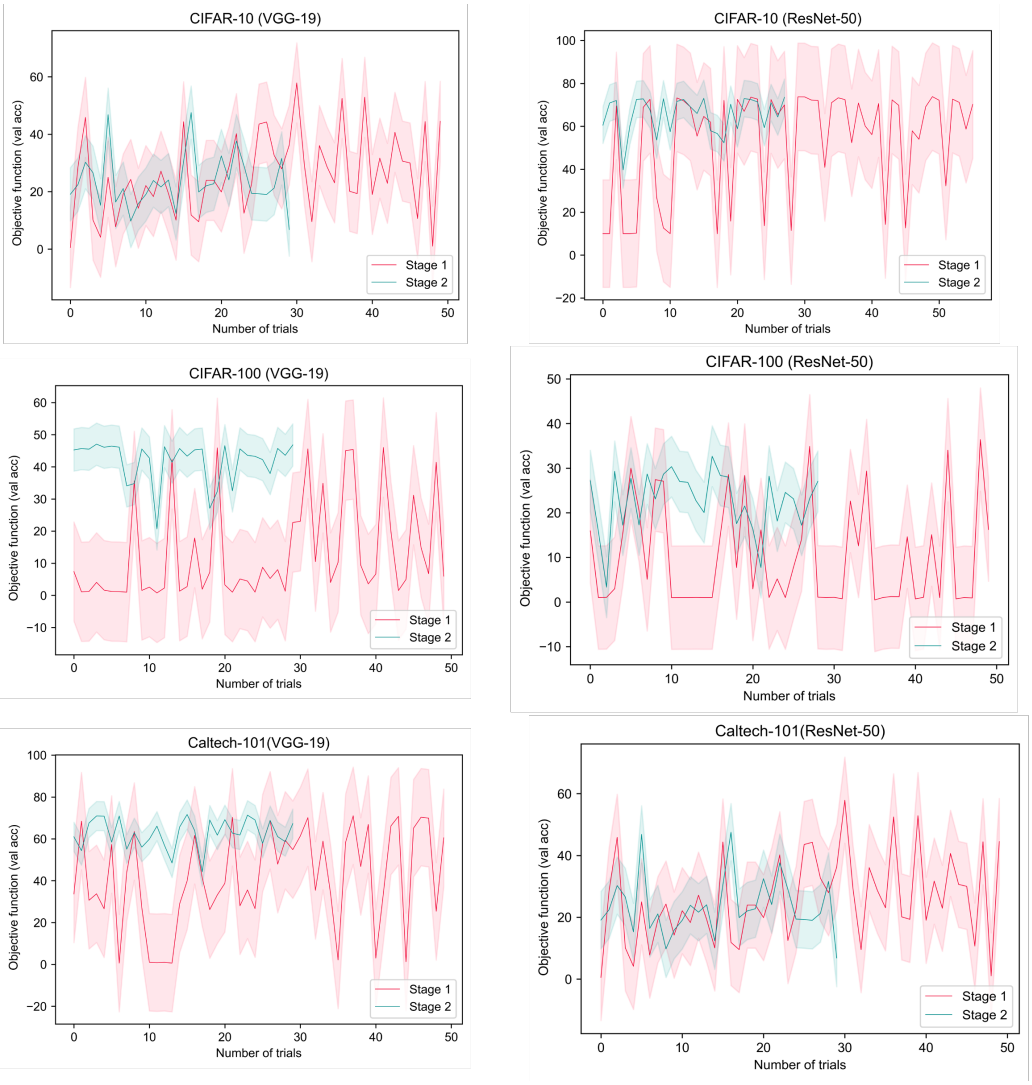


Figure 12. Data-driven analysis of exploration in Stage 1 of HyperGE across all datasets.

6.2. Comparitive Analysis with SOTA

The comparative analysis in terms of the number of trials and models employed and their respective test accuracy are presented in Table 4. The test accuracy of HyperGE across the models is competitive with the SOTA models. *However, HyperGE has learned the hyperparameters using only 80 trials trained over 5% of the dataset over five epochs.* HyperGE achieves near-optimal performance within minimal trials, with less time using a subset of data. Moreover, the flexibility of GE to adapt and scale hyperparameters and their choices

are added advantages. An example of a BNF grammar with more hyperparameters is presented in Figure 13 which can be a possible future direction for HyperGE.

Table 4. Comparative analysis of HyperGE against SOTA. The bold results indicate our approach outperforms SOTA.

Dataset	Approach	Model	# Trials	# Epochs/Time	Test Accuracy (%)
CIFAR-10	Springenberg et al. [5]	ResNet-32	104	6 h	93
	Lévesque et al. [24]	cuda-convnet		250	88
	HyperGE	**VGG-19**	**80**	**50/3 h**	**84.98**
	HyperGE	**ResNet-50**	**50**	**30/2 h**	**82.80**
Caltech-101	Autotune(BO) [30]	ResNet-50	-	50	92.01
	Autotune(RS) [30]	ResNet-50	-	50	92.94
	HyperGE	**VGG-19**	**80**	**50**	**88.75**
	HyperGE	**ResNet-50**	**80**	**30**	**91.89**

```
<finetuning> ::= <batch_size> <optimiser> <lr>
                 <momentum> <activation> <dropout> <layers>

<batch_size> ::= 32 | 64 | 128 | 256
<optimiser> ::= 'adam' | 'rmsprop' | 'sgd' | 'Levenberg-Marquardt'
<lr> ::= 0.00001 | 0.0001 | 0.001 | 0.01 | 0.1 | 0.00005 | 0.0005
        | 0.005 | 0.05 | 0.5
<momentum> ::= 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 0.99
<activation> ::= 'leakyrelu' | 'relu' | 'sigmoid' | 'tanh'
<dropout> ::= 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9
<layers> ::= 32 | 64 | 96 | 128 | 160 | 192 | 224 |  256 | 288 | 320 | 352
             | 384 | 416 | 448 | 480 | 512
```

Figure 13. An example illustrating flexibility of BNF grammar in scaling hyperparameters.

7. Conclusions and Future Scope

HPO is an important challenge for the efficient performance of NNs. In this research work, we propose HyperGE, a two-staged GE-driven model to reduce computational resources while fine-tuning NNs on three benchmark image classification datasets and two NN architectures: VGG-19 and ResNet-50. The findings indicate the capabilities of HyperGE in reducing the hyperparameter search space by 90% in Stage 2. The immediate future scope of this research could be to test HyperGE across different NNs for different tasks. Additionally, reducing fitness evaluation complexity can be explored through the use of approximate neural networks and surrogate or approximate functions. This could provide a more efficient method for evaluating complex problems, allowing us to gain insights into how best to optimise our solutions in order to maximise performance and efficiency.

Author Contributions: Conceptualization, M.K.; Data curation, G.V.; Funding acquisition, C.R.; Methodology, M.K.; Project administration, C.R.; Software, G.V.; Supervision, C.R.; Visualization, G.V.; Writing—original draft, G.V. and M.K.; Writing—review editing, C.R. M.K. and G.V. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Science Foundation Ireland Grant #16/IA/4605.

Data Availability Statement: All the source and code files are available at: https://github.com/gauriivaidya/GE-HPO (accessed on 24 June 2023).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kshirsagar, M.; More, T.; Lahoti, R.; Adgaonkar, S.; Jain, S.; Ryan, C. Rethinking Traffic Management with Congestion Pricing and Vehicular Routing for Sustainable and Clean Transport. In Proceedings of the 14th International Conference on Agents and Artificial Intelligence—Volume 3: ICAART, Online, 3–5 February 2022; pp. 420–427. [CrossRef]
2. Bahja, M. Natural Language Processing Applications in Business. In *E-Business-Higher Education and Intelligence Applications*; BoD–Books on Demand: Norderstedt, Germany, 2020. [CrossRef]
3. Hewamalage, H.; Bergmeir, C.; Bandara, K. Recurrent Neural Networks for Time Series Forecasting: Current status and future directions. *Int. J. Forecast.* **2021**, *37*, 388–427. [CrossRef]
4. Xiao, Y.; Wu, L.; Guo, J.; Li, J.; Zhang, M.; Qin, T.; Liu, T.Y. A Survey on Non-Autoregressive Generation for Neural Machine Translation and Beyond. *IEEE Trans. Pattern Anal. Mach. Intell.* **2023**, 1–20. [CrossRef]
5. Diaz, G.I.; Fokoue-Nkoutche, A.; Nannicini, G.; Samulowitz, H. An effective algorithm for hyperparameter optimisation of neural networks. *IBM J. Res. Dev.* **2017**, *61*, 9:1–9:11. [CrossRef]
6. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26–30 June 2016; pp. 770–778.
7. Bochinski, E.; Senst, T.; Sikora, T. Hyper-parameter optimisation for convolutional neural network committees based on evolutionary algorithms. In Proceedings of the 2017 IEEE International Conference on Image Processing (ICIP), Beijing, China, 17–20 September 2017; pp. 3924–3928. [CrossRef]
8. Bergstra, J.; Bengio, Y. Random Search for Hyper-Parameter Optimisation. *J. Mach. Learn. Res.* **2012**, *13*, 281–305.
9. Bergstra, J.; Bardenet, R.; Bengio, Y.; Kégl, B. Algorithms for Hyper-Parameter Optimisation. In *Advances in Neural Information Processing Systems*; Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., Weinberger, K., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2011; Volume 24.
10. DeCastro-García, N.; Castañeda, Á.L.M.; García, D.E.; Carriegos, M.V. Effect of the Sampling of a Dataset in the Hyperparameter Optimisation Phase over the Efficiency of a Machine Learning Algorithm. *Complexity* **2019**, *2019*, 6278908. [CrossRef]
11. Hensman, J.; Fusi, N.; Lawrence, N.D. Gaussian Processes for Big Data. *arXiv* **2013**, arXiv:1309.6835.
12. Zhang, L.; Zhang, L.; Zhang, L. Application research of digital media image processing technology based on wavelet transform. *EURASIP J. Image Video Process* **2018**, *2018*, 138. [CrossRef]
13. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Commun. ACM* **2017**, *60*, 84–90. [CrossRef]
14. Han, X.; Zhang, Z.; Ding, N.; Gu, Y.; Liu, X.; Huo, Y.; Qiu, J.; Yao, Y.; Zhang, A.; Zhang, L.; et al. Pre-trained models: Past, present and future. *AI Open* **2021**, *2*, 225–250. [CrossRef]
15. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei, L. ImageNet: A large-scale hierarchical image database. In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 20–25 June 2009; pp. 248–255. [CrossRef]
16. Bergstra, J.; Komer, B.; Eliasmith, C.; Yamins, D.; Cox, D.D. Hyperopt: A Python library for model selection and hyperparameter optimisation. *Comput. Sci. Discov.* **2015**, *8*, 014008. [CrossRef]
17. Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; Koyama, M. Optuna: A Next-generation Hyperparameter Optimisation Framework. *arXiv* **2019**, arXiv:1907.10902.
18. Liaw, R.; Liang, E.; Nishihara, R.; Moritz, P.; Gonzalez, J.E.; Stoica, I. Tune: A Research Platform for Distributed Model Selection and Training. *arXiv* **2018**, arXiv:1807.05118.
19. Hayes, P.; Anderson, D.; Cheng, B.; Spriggs, T.J.; Johnson, A.; McCourt, M. *SigOpt Documentation*; Technical Report SO-12/14 – Revision 1.07; SigOpt, Inc.: San Francisco, CA, USA, 2019.
20. Li, L.; Jamieson, K.; DeSalvo, G.; Rostamizadeh, A.; Talwalkar, A. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimisation. *J. Mach. Learn. Res.* **2017**, *18*, 6765–6816.
21. Jaderberg, M.; Dalibard, V.; Osindero, S.; Czarnecki, W.M.; Donahue, J.; Razavi, A.; Vinyals, O.; Green, T.; Dunning, I.; Simonyan, K.; et al. Population Based Training of Neural Networks. *arXiv* **2017**, arXiv:1711.09846.
22. Yu, T.; Zhu, H. Hyper-Parameter Optimization: A Review of Algorithms and Applications. *arXiv* **2020**, arXiv:2003.05689.
23. Springenberg, J.T.; Klein, A.; Falkner, S.; Hutter, F. Bayesian Optimisation with Robust Bayesian Neural Networks. In *Advances in Neural Information Processing Systems*; Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2016; Volume 29.
24. Levesque, J.; Gagné, C.; Sabourin, R. Bayesian Hyperparameter Optimization for Ensemble Learning. *arXiv* **2016**, arXiv:1605.06394.
25. Stang, M.; Meier, C.; Rau, V.; Sax, E. *An Evolutionary Approach to Hyper-Parameter Optimisation of Neural Networks*; Springer: Cham, Switzerland, 2020; pp. 713–718. [CrossRef]
26. Han, J.H.; Choi, D.J.; Park, S.U.; Hong, S.K. Hyperparameter Optimisation Using a Genetic Algorithm Considering Verification Time in a Convolutional Neural Network. *J. Electr. Eng. Technol.* **2020**, *15*, 721–726. [CrossRef]
27. Xiao, X.; Yan, M.; Basodi, S.; Ji, C.; Pan, Y. Efficient Hyperparameter Optimisation in Deep Learning Using a Variable Length Genetic Algorithm. *arXiv* **2020**, arXiv:2006.12703.
28. Yeh, W.C.; Lin, Y.P.; Liang, Y.C.; Lai, C.M.; Huang, C.L. Simplified swarm optimisation for hyperparameters of convolutional neural networks. *Comput. Ind. Eng.* **2023**, *177*, 109076. [CrossRef]

29. Basha, S.; Vinakota, S.K.; Dubey, S.R.; Pulabaigari, V.; Mukherjee, S. Autofcl: Automatically tuning fully connected layers for transfer learning. *arXiv* **2020**, arXiv:2001.11951.
30. Basha, S.S.; Vinakota, S.K.; Pulabaigari, V.; Mukherjee, S.; Dubey, S.R. AutoTune: Automatically Tuning Convolutional Neural Networks for Improved Transfer Learning. *Neural Netw.* **2021**, *133*, 112–122. [CrossRef]
31. Vaidya, G.; Ilg, L.; Kshirsagar, M.; Naredo, E.; Ryan, C. HyperEstimator: Evolving Computationally Efficient CNN Models with Grammatical Evolution. In Proceedings of the 19th International Conference on Smart Business Technologies, Lisbon, Portugal, 14–16 July 2022; pp. 57–68.
32. Ryan, C.; Collins, J.; Neill, M.O. Grammatical evolution: Evolving programs for an arbitrary language. In *Genetic Programming*; Banzhaf, W., Poli, R., Schoenauer, M., Fogarty, T.C., Eds.; Springer: Berlin/Heidelberg, Germany, 1998; pp. 83–96.
33. O'Neill, M.; Ryan, C. Grammatical evolution. *IEEE Trans. Evol. Comput.* **2001**, *5*, 349–358. [CrossRef]
34. Ryan, C.; Kshirsagar, M.; Chaudhari, P.; Jachak, R. GETS: Grammatical Evolution based Optimisation of Smoothing Parameters in Univariate Time Series Forecasting. In Proceedings of the 12th International Conference, ICAART, Valletta, Malta, 22–24 February 2020; pp. 595–602. [CrossRef]
35. Ryan, C.; Kshirsagar, M.; Vaidya, G.; Cunningham, A.; Sivaraman, R. Design of a cryptographically secure pseudo random number generator with grammatical evolution. *Sci. Rep.* **2022**, *12*, 8602. [CrossRef] [PubMed]
36. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*; Curran Associates, Inc.: Red Hook, NY, USA, 2019; pp. 8024–8035.
37. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv* **2015**, arXiv:1409.1556.
38. Krizhevsky, A.; Hinton, G. *Learning Multiple Layers of Features from Tiny Images*; Technical Report 0; University of Toronto: Toronto, ON, USA, 2009.
39. Li, F.F.; Andreeto, M.; Ranzato, M.; Perona, P. *Caltech 101*; CaltechDATA: Pasadena, CA, USA, 2022. [CrossRef]
40. Fenton, M.; McDermott, J.; Fagan, D.; Forstenlechner, S.; O'Neill, M.; Hemberg, E. PonyGE2: Grammatical Evolution in Python. *arXiv* **2017**, arXiv:1703.08535.
41. Baldominos, A.; Saez, Y.; Isasi, P. Evolutionary Convolutional Neural Networks: An Application to Handwriting Recognition. *Neurocomput.* **2018**, *283*, 38–52. [CrossRef]