Genetic Programming and Evolvable Machines (2024) 25:10
https://doi.org/10.1007/s10710-024-09483-5

A genetic programming approach to the automated design of CNN models for image classification and video shorts creation

Rahul Kapoor[1] · **Nelishia Pillay**[1]

Received: 23 April 2023 / Revised: 22 October 2023 / Accepted: 6 February 2024 /
Published online: 14 March 2024

Abstract
Neural architecture search (NAS) is a rapidly growing field which focuses on the automated design of neural network architectures. Genetic algorithms (GAs) have been predominantly used for evolving neural network architectures. Genetic programming (GP), a variation of GAs that work in the program space rather than a solution space, has not been as well researched for NAS. This paper aims to contribute to the research into GP for NAS. Previous research in this field can be divided into two categories. In the first each program represents neural networks directly or components and parameters of neural networks. In the second category each program is a set of instructions, which when executed, produces a neural network. This study focuses on this second category which has not been well researched. Previous work has used grammatical evolution for generating these programs. This study examines canonical GP for neural network design (GPNND) for this purpose. It also evaluates a variation of GP, iterative structure-based GP (ISBGP) for evolving these programs. The study compares the performance of GAs, GPNND and ISBGP for image classification and video shorts creation. Both GPNND and ISBGP were found to outperform GAs, with ISBGP producing better results than GPNND for both applications. Both GPNND and ISBGP produced better results than previous studies employing grammatical evolution on the CIFAR-10 dataset.

Keywords Genetic programming · Iterative structure based search · Genetic algorithm · Automated design · Neural network · Neural architecture search

Area Editor: Sebastian Risi.

✉ Rahul Kapoor
u16034130@tuks.co.za

Nelishia Pillay
npillay@cs.up.ac.za

[1] Department of Computer Science, University of Pretoria, Pretoria, South Africa

1 Introduction

Deep neural networks (DNNs), in particular convolutional neural networks, have been very effective for image, video and text processing and classification [3, 16]. Designing architectures to perform these tasks can be done manually; however, it is quite cumbersome and time-consuming. It can be difficult to determine which architecture would be the best for solving the task at hand, and the process of determining this structure can take time, with multiple attempts involving trial-and-error. It takes experienced researchers to determine the correct architecture for highly accurate neural networks in a reasonable amount of time. Due to this, the process of determining a DNN architecture is now automated resulting in the area of Neural Architecture Search (NAS) [6] which involves automating neural network architecture design. NAS involves employing optimization techniques to determine the most suitable neural network architecture for the problem at hand.

At the inception of the field, reinforcement learning (RL) was used for the automation, however, as the field developed evolutionary algorithms have proven to outperform the RL [8]. Evolutionary Computation (EC) methods exist for NAS [37]. Genetic algorithms, in particular, have been predominantly used for NAS [28]. There also exist one-shot and training-free approaches for NAS [11, 46] Additionally, genetic programming (GP) [19] has been investigated for NAS, but has not been as well researched as GAs for this purpose [27]. GP explores a program space rather than a solution space. In previous work applying GP for NAS, the programs take one of two forms. In some studies the programs represent the neural network directly [42, 43, 47] or a representation of the neural network and parameters [29, 33, 38]. In others the program is comprised of instructions which when executed produce a neural network [5, 25]. The research presented in this paper focuses on the latter.

Grammatical evolution [35] has generally been used to evolve a program of instructions to construct a neural network for the problem at hand. This paper investigates canonical genetic programming for this purpose, namely, genetic programming for neural network design (GPNND). The performance of GPNND is compared to that of GAs which is traditionally used for NAS. GPNND is also compared to a variation of genetic programming, namely, iterative structure-based genetic programming (ISBGP) [17] which takes both fitness and structure into consideration when directing the search in the program space. In this paper ISBGP is improved based on the findings in [17]. Firstly, a more accurate similarity index, that compares all nodes, instead of estimating similarity, is used. Secondly, early stopping is removed, giving the algorithm more time to converge. We refer to this version of the algorithm as ISBGP-II.

A GA, GPNND and ISBGP-II have been implemented for NAS and evaluated on image classification (4 datasets) and video shorts creation problems (4 datasets). For all datasets GPNDD and ISBGP-II outperforms the GA, with ISBGP-II outperforming GPNDD. Furthermore, ISBA-II produces better results than ISBGP. Hence, the contributions of this research include:

- An investigation into canonical GP for NAS, with each program a set of instructions that is executed to produce a neural network.
- A comparative study of the performance of genetic algorithms, canonical genetic programming, and iterative structure-based genetic programming for NAS.
- An improvement over the original ISBA algorithm for NAS.

The remainder of this paper is structured as follows: Sect. 2 briefly goes through some related work on the topic and states the differences between the related work and our work. The genetic algorithm is detailed in Sect. 5. Section 3 describes the GPNND and Sect. 4 the ISBGP-II. The experimental setup is presented in Sect. 6. Section 7 discusses the results of the experiments, and finally the research conclusions and some possible future work is presented in Sect. 8.

2 Related work

This section firstly provides a brief introduction to the application domains, namely, image classification and video shorts creation followed by an overview of NAS. Previous work on GAs and GP for NAS for image processing is then outlined. NAS has not previously been applied for video shorts creation.

2.1 Image classification

Image classification represents a foundational challenge in the domains of artificial intelligence and machine learning [21]. This problem involves the automated assignment of predefined categories or labels to images based on their inherent content. The fundamental objective of image classification is to allow machines to discern and categorize various objects, scenes, or patterns depicted in images, thereby emulating the cognitive prowess of humans.

The principal aim is the development of algorithms and models proficient in recognizing and categorizing objects within images, also accommodating variations in pose, lighting, and occlusion. This allows for the extraction of features and representations from raw pixel values, empowering the technique to comprehend visual attributes that differentiate distinct classes. The process of converting data into semantically rich features allows for the development of accurate image classifiers.

The inherent challenges in image classification, including the management of extensive and diverse datasets, the mitigation of overfitting, and the attainment of high accuracy, propel the exploration of innovative techniques in machine learning. [21]

Some of the most notable approaches to image classification include work from [37], where evolutionary algorithms to perform NAS for image classification, achieving accuracies comparable to recent publications on CIFAR-10 and CIFAR-100 datasets, are used. A fine-tuning of differential architecture search by [44] for performing classifications and also reducing the number of trainable parameters. An approach by [45] employs a semi-supervised learning framework which improves the accuracy of many semi-supervised image classification algorithms. Work by [7] employs a procedure which minimizes both the loss value and loss sharpness and has a great improvement in the accuracy of models.

2.2 Video shorts creation

This area goes by multiple names, for example video highlight detection is also used. Certain names in their contexts may have small nuances, however they all follow the same general principle. The method will be referred to as video shorts creation in this paper.

Video shorts creation is the process whereby a video is condensed into a shorter form of itself. This shortened video should retain the most important parts of the full video, and should still convey the intended message of the full video to the viewer, in its condensed form. A common example of this is sports match or movie highlights. This process was initially performed manually, however AI models are now being developed to perform this task. A model for VSC can take in the original video as input and produce the frames corresponding to the highlight areas as output.

Some examples of Video Shorts Creation are researched in [16, 32, 48]. [48] used CNNs to detect highlights in construction footage, allowing for only the most important parts of the footage to be stored, saving storage space. In the work by [16], soccer match events are detected by making use of CNNs and recurrent neural networks. [32] also used a CNN for video summarisation for Internet of Things devices.

2.3 Neural architecture search

Neural architecture search is the process of finding the appropriate architecture of a neural network to efficiently solve the problem at hand [6]. Initially, NAS was performed manually. As the field of neural networks grew, neural network architecture design became time-consuming and difficult, hence this process was automated and led to the emergence of NAS. Initially reinforcement learning [15] was used for NAS and as the field developed evolutionary algorithms, in particular, genetic algorithms, were shown to outperform those using RL [8]. Currently NAS employs optimization methods, predominately genetic algorithms [28] to find the neural network architecture. More recently GP has been investigated for this purpose. The following sections report on previous work using GAs and GP for NAS respectively. The studies reported on are restricted to image processing and video shorts creation.

2.4 GAs for NAS

Image classification is probably the most researched subfield in AI and ML. It serves as a good test bed for many AI problems. There is a fair amount of research into using NAS for image classification. This section provides an overview of the studies that are relevant to the study in this paper.

The approach by [4] employs a GA to design a model for facial expression recognition. The algorithm makes use of a novel encoder-decoder model for automatic evolution of CNN models. In [26] a GA for network topology optimisation is used and evaluated on the MNIST dataset. A GA is used for acoustic scene classification in [13]. The model is split into sub-models which are optimized individually and as a combination. These methods work well and achieve good accuracy on the datasets they test on.

2.5 GP for NAS

GP explores a program space rather than a solution space [19]. Thus, each element of the population is a program which when executed will produce a solution. This section provides an overview of previous studies investigating GP for NAS for image processing. There has been no research into NAS for video shorts creation. These studies can divided into two categories based on what each program represents.

The first category includes those studies in which a program represents a neural network directly or a simplified representation of the neural network. One of the earlier studies is that conducted by Ritchie et al. [38] in which each program represented the connections between nodes in a neural network and the corresponding weights. In the study conducted by McGhie et al. [29] each neural network and associated parameters is represented as a tree. Narajan [33] represents each neural network as a directed graph consisting of convolutional and pooling operators and their corresponding parameter values. Various studies [42, 43, 47] have employed Cartesian genetic programming (CGP) [30] in which the neural network is represented as an acyclic graph.

The second category are those studies in which each program is comprised of instructions for creating a neural network, e.g. add a layer. In the study conducted by Diniz et al. [5] grammatical evolution [35] is used to evolve programs that construct neural networks. Similarly, Lima et al. [25] also use grammatical evolution to generate programs that construct neural networks. The study presented in this paper focuses on this second category, i.e. using GP to evolve programs of instructions to create a neural network. Canonical GP is investigated for this purpose. In addition to this, a variation of GP that takes both fitness and structure into consideration when exploring the program space, namely, iterative structure-based GP, will also be evaluated for NAS. The study compares the performance of canonical GP, iterative structure-based GP and genetic algorithms for NAS for image classification and video shorts creation.

3 Genetic programming for neural network design (GPNND)

This section describes the GPNND. The genetic programming algorithm is the generational algorithm depicted in Algorithm 2. The sections that follow describe the processes comprising the algorithm.

Table 1 Function nodes for the GPNND approach

Name	Arity	Description	Input
Layer	2	A layer type for the neural network Can either be: convolution (C), max pooling (MP), average pooling (AP), dropout (Dr), densely connected (D) or flattening (F) layer	- Layer parameter (A 'Shape' terminal, defined in Table 2) - Activation function (A 'LFunc' terminal, defined in Table 2)
AddLayer	2	Adds two layers together	Two layers
AdjustLayer	2	Adjusts the parameters for a layer. Can change the layer size, dimensions or activation function	- Layer - Layer parameter (A 'Shape' terminal, defined in Table 2)
AdjustNet	1	Changes the parameters for the neural network. Can change the learning rates or optimiser	- Neural network parameter (A 'NOpt' or a 'NLRate' terminal, defined in Table 2)
If-Then-Else	3	Performs the function of a standard if-then-else statement.	- Condition - True Branch - False Branch
CheckLayer	1	Checks if a specific layer is in the neural network and returns a boolean value	- Layer
SwapLayers	2	Swaps the order of two layers	- Two layers
ContainsParams	2	Checks if there is a specific parameter defined for a layer. For example, it can check to see if a layer contains a activation function or certain dimensions	- Layer - Parameter (A 'Shape' terminal, defined in Table 2)
Comb2	2	Combiner used to combine two subtrees together	AddLayer or Layer
Comb3	3	Combiner used to combine three subtrees together.	AddLayer or Layer

3.1 Representation and initial population generation

Each element of the population is a parse tree representing a program consisting of instructions to produce a neural network to solve the problem at hand. The function and terminal sets used to create the programs are described in Tables 1 and 2 respectively.

Each program is used to build a neural network. Hence, the function set contains primitives to add different layers and set parameters for the neural network. The terminal set contains elements which are building blocks of a neural network, such as the parameter actual values. Therefore, in this approach, an activation function of a layer in a neural network would be seen as a terminal node, since it is part of the building blocks of the neural network.

Table 2 Terminal Nodes for GPNND approach

Name	Description
Shape	This represents the options which a layer can take as a parameter
	Depending on the layer type, one of these options are randomly chosen
	Convolution layer (C)
	- 016, 032, 048, 064, 128, 256, 512, 1024 (The number of convolutions)
	Max pooling and average pooling layers (MP, AP)
	- 03, 05, 07, 09, 11, 13 (The size of the kernel, for example ’05’ represents a 5x5 kernel)
	Dropout layers (Dr)
	- 0.2, 0.4, 0.5 (The dropout proportion)
	Densely connected layers (D)
	- 0, 16, 32, 48, 64 (The number of neurons in the layer)
	(except for final dense layer which takes on number of classes as its size)
LFunc	The activation function for a layer, which could be one of the following
	- ReLU
	- tanh
	- sigmoid
	- softmax
	One of these values are randomly selected from the oprions specified
NOpt	The Optimiser of the network
	- Adam, AdaGrad, AdaDelta. or SGD
	One of these values are randomly selected from the options specified
NLRate	The Learning Rate of the neural netowrk
	Learning rates can be 0.1, 0.01, 0.001
	One of these values are randomly selected from the range specified
NSize	Size (number of layers) of the neural network.
C	Ephemeral constant in the range -2 and 2

Table 1 details the name, arity, and imputs, and gives a description of the purpose of each primitive.

In Table 2 the names of the terminals and a description of the values they may take are provided. The sets of possible vales of the neural network layer parameters for GPNND are the same as the set of possible values for GA. This is to enforce the same experimental conditions for both approaches. The justification of these values is stated in Sect. 5.1.

An example of an element of the population and the corresponding neural network that it produces is depicted in 1. In order to evaluate the tree in Fig. 1, a pre-order traversal will take place. Starting from the root, moving along the left branch, the left most primitive is the node labelled ’C’, which indicates a convolution layer. The left child of this primitive is a node labelled ’16’ which indicates that the convolution layer will have 16 convolutions. The right child of the ’C’ primitive is labelled

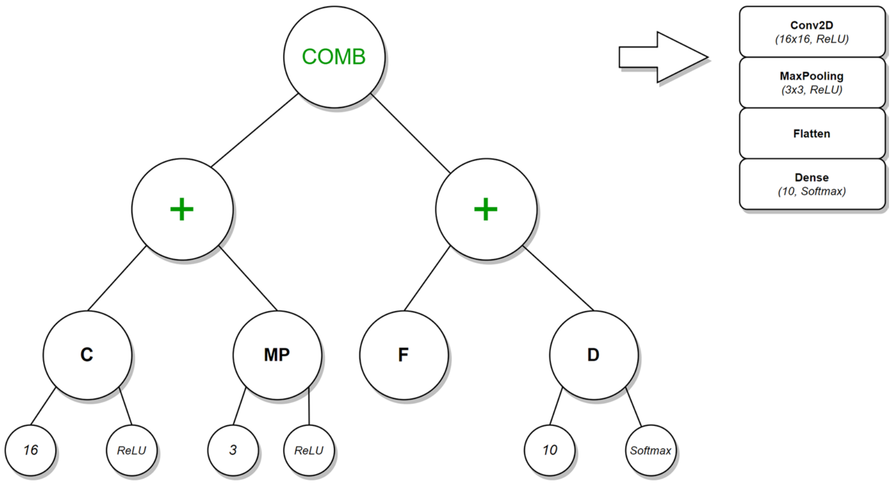

Fig. 1 Example of GP tree and the corresponding neural network derived from the tree

’ReLU’, which indicates that the layer will make use of a ReLU activation function. These three nodes, together represents a convolutional layer of a neural network which perform 16 convolutions and uses a ReLU activation function, indicated by the first block in the neural network representation on the right hand side of the figure. The ’C’ node is the left child of the Add (+) primitive. The same process takes place with the right child of the Add primitive. It is labelled ’MP’ which represents a max pooling layer. The ’MP’ primitive has two children. The left child is labelled ’3’. This indicates that the max pooling layer has a 3x3 kernel size. The left child is labelled ’ReLU’, indicating a ReLU activation function. These three nodes, together represents a max pooling layer of a neural network which perform 16 convolutions and uses a ReLU activation function. The ’MP’ node is the right child of the Add (+) primitive. The + primitive adds its children together, and therefore adds the convolution layer and max pooling layer together. The neural network representation on the right thus shows the max pooling layer underneath the convolution layer. The left hand side of the tree has now been traversed. The same process occurs with the right child of the root. The right child of the root is another Add primitive. Its left child is a ’F’ node which represents a flattening layer. Flattening layers do not have parameters. The right child of the Add primitive is labelled ’D’ which represents a densely connected layer. Its left and right children are labelled ’10’ and ’Softmax’, which indicate a size of 10 and a softmax activation function respectively. These three nodes represent a densely connected layer with size 10 which uses a softmax activation function. The Add primitive then adds these layers together. The right hand side of the tree has been traversed. The root is labelled ’Comb’ which represents a combiner primitive, which combines two subtrees together. The left hand side of the tree which contains the convolution layer and the max pooking layer is combined with the right hand side of the tree which contains a flattening and densely connected layer, resulting in the neural network depicted on the right hand portion of the figure.

The population is created using the ramped half-and-half method [19]. A default min and max depth is chosen and an equal number of trees in the range of this depth are generated. Half of these will be generated using the grow method (trees of variable shape adhering to depth bounds) and the other half are generated using the full method (trees of full shape, no missing nodes).

3.2 Genetic operators

GPNND uses the mutation and crossover operators to create offspring. These operators are described in the sections that follow.

3.2.1 Mutation

An individual is selected, a mutation point is randomly selected. The subtree rooted at the point is deleted and a new subtree is generated using the grow method and inserted at this point. Figure 2 illustrates an example of mutation. In Fig. 2, the top of the figure represents the chosen tree before mutation, along with the corresponding neural network the tree evaluates to, and the bottom of the figure depicts the tree after mutation with its corresponding neural network. The mutation point is circled in red. The subtree rooted at this point is replaced with a newly generated subtree, and the bottom tree below shows the newly generated subtree along with the neural network which this offspring will create.

3.2.2 Crossover

Two individuals are selected, and a crossover point in both individuals is selected. The subtrees rooted at each of these points are swapped with each other. An example of crossover is shown in Fig. 3. In Fig. 3, the top shows two trees, along with the neural networks they evaluate to, before crossover takes place and the bottom of the figure shows the trees and corresponding neural networks after crossover has taken place. Crossover points are shown in red and blue. The subtrees rooted at these points are interchanged between the two program trees.

3.3 Fitness evaluation and selection

The fitness function is multi-objective. The objective functions are:

1. Maximise the model accuracy
2. Minimise the training time
3. Minimise the size of the model

The accuracy of the neural network is the most important objective. However, it cannot be that the size of the model grows to be too large, or that it takes too long for the GP to evolve to produce the neural network. Therefore, these values have been log scaled and summed as follows:

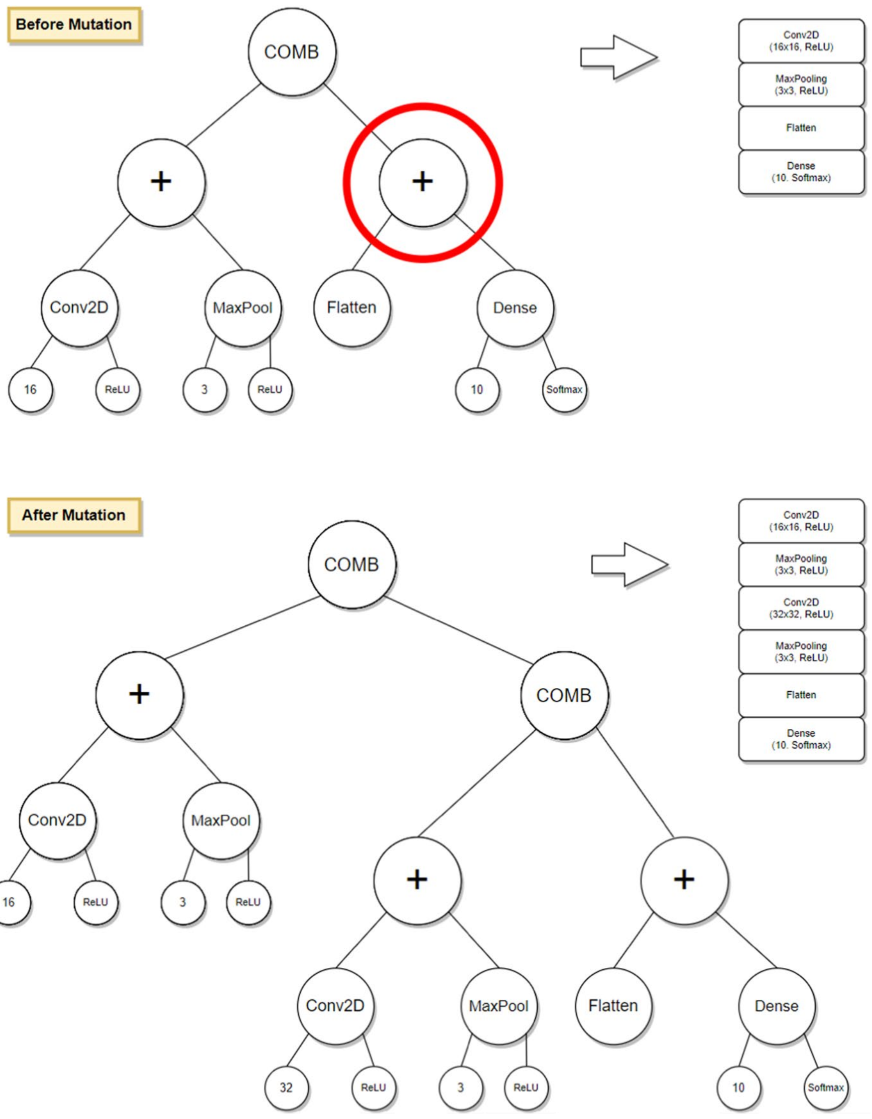

Fig. 2 Example of mutation (Mutation point circled in red)

$$F(i) = 100acc(i) - 10log(train(i)) - 5log(size(i))$$

Where $F(i)$ refers to the fitness of the individual, $acc(i)$ the accuracy, $train(i)$ the training time in seconds, and $size(i)$ the size of the model in Kilobytes after training. GPNND uses tournament selection. A tournament size is chosen, and a random selection is made from the population, comprising the chosen number of individuals. Thereafter, the individual with the best fitness within the tournament is selected for the application of a genetic operator.

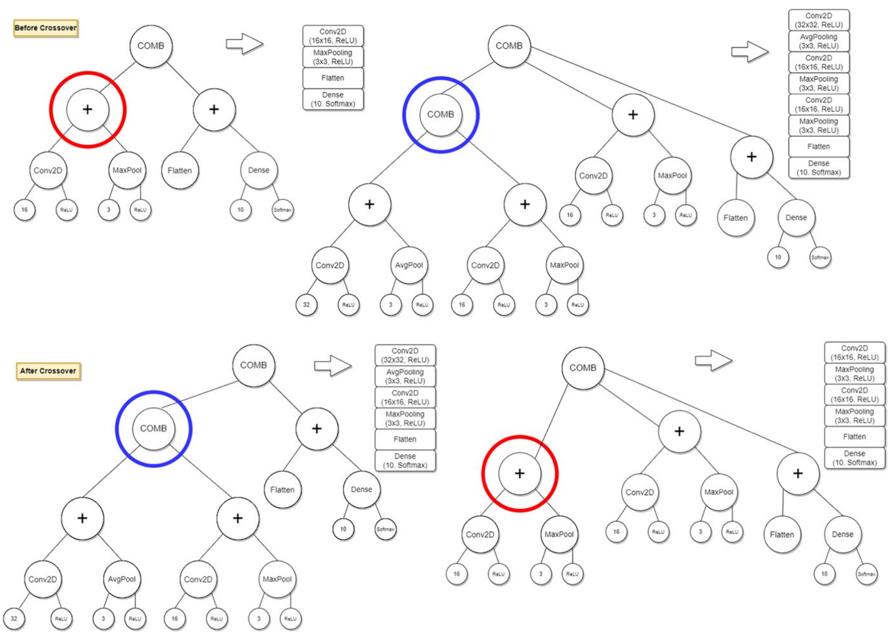

Fig. 3 Example of crossover (Mutation points circled in red and blue)

4 Iterative structure based genetic programming (ISBGP-II)

This section describes the ISBGP-II algorithm. ISBGP-II uses both fitness and structure to direct the search. Structure is taken into consideration by preventing the algorithm from exploring areas that previously resulted in local optima. A similarity index is used to compare new elements of the population to elements in areas of the search space already visited. This done at both a local level to promote exploitation, as well as a global level to promote exploration.

The ISBGP-II algorithm is depicted in algorithm 1.

Algorithm 1 ISBGP-II

```
1: while termination criterion has not been met do
2:     for $i = 1$ to $m$ do
3:         Perform local level search for global area $g_i$
4:         Identify a global area different from previous $g_i$s
5:     end for
6: end while
```

The algorithm operates by firstly executing the GPNND algorithm over multiple iterations and analyzing nodes in the tree up to a predefined cutoff depth parameter, denoted as the global level search. The global level search identifies various global areas, with each iteration revealing a new area by comparing it with those found in previous iterations. If the similarity of the new global area is below the defined threshold, it is accepted; otherwise, it is discarded. The similarity is quantified using the global index, defined in Sect. 4.1. Within each global level search, a local level search occurs.

In the local level search, nodes from the tree root to the cutoff depth are fixed, initiating the local level search of the algorithm. In this phase, nodes above the cutoff depth cannot be modified. If the similarity of the new local area is below the defined threshold, the search continues. If not, the search moves to a new global search. The similarity is quantified using the local index, defined in Sect. 4.1

This iterative process continues until the termination criterion is satisfied. Termination occurs either when the algorithm achieves a fitness surpassing that of GPNND for the same dataset or when the design time matches that of the GPNND approach.

These search strategies encourage the algorithm to both exploit and explore more of the search space, thereby minimizing the risk of converging to a local optimum. The utilization of similarity indices prevents excessive exploration of a single area in the search space. This control was not possible when solely relying on genetic operators.

Population generation, genetic operators, and the fitness function for ISBGP-II mirror those used in the GPNND approach.

4.1 Similarity indices

Two similarity indexes are used for this purpose:

- Global index (GI) - The global similarity index is a count of the number of function nodes that the individuals being compared have in common from the root to a specified cut-off depth.
- Local index (LI) - The local similarity index counts the number of function and the number of terminal nodes that both the individuals being compared have in common, after the cut-off depth.

5 Genetic algorithm for NAS (GA)

Genetic algorithms have predominantly been used for NAS. To get an idea of the contribution of GPNND and ISBGP-II, a genetic algorithm for NAS is also applied to the same datasets used to evaluate GPNND and ISBGP-II. The genetic algorithm is described in this section. The GA is based on that by employed by Klos et al. [18] for NAS. Both GPNND and the GA presented in this section employ the standard generational algorithm depicted in Algorithm 2.

Algorithm 2 Generational algorithm

Table 3 Layer types and parameters for chromosomes

Name	Parameters	Parameter values
C - Convolution	Number of convolutions	16, 32, 48, 64, 128, 256, 512, 1024
MP - Max pooling	Kernel size	3x3, 5x5,7x7, 9x9, 11x11, 13x13
AP - Average pooling	Kernel size	3x3, 5x5,7x7, 9x9, 11x11, 13x13
F - Flattening layer	–	–
D - Dense	Number of neurons	10, 16, 32, 48, 64. 128, 256, 521, 1024
Dr - Dropout	Dropout proportion	0.2, 0.4, 0.5

```
1: Generate an initial population
2: while termination criteria are not met do
3:     Evaluate the fitness of each individual in the population
4:     Select individuals to apply genetic operators to
5:     Form new population by applying the genetic operators to the selected
   individuals
6: end while
7: After the termination criteria are met, the individual with the best fitness
   is returned
```

This section firstly describes the representation used for the chromosome, followed by an overview of each of the processes that make up this algorithm.

5.1 Chromosome representation

Each chromosome represents a neural network. Each gene corresponds to a layer in the neural network, its corresponding parameters and activation function. The parameters and the options for the parameter values for each layer type is depicted in Table 3. The table lists the set of parameter values for a layer, one of which are randomly chosen when a chromosome is generated. The reason for choosing the values in this table as possible values for the algorithm to make use of is that high performing neural network designs tend to use multiples of 16 for convolutions and multiples of odd numbers for kernel sizes for pooling [14, 40]. The values for dropout were chosen as they help to reduce overfitting, the range was chosen as having dropout values which are too high could lead to underfitting, slow convergence or loss of information [10, 41]. One of the following three activation functions is randomly selected for each layer: ReLU, TanH, Softmax.

In Fig. 4, the left hand side depicts the chromosome of the GA, and the right hand side depicts the neural network this chromosome represents. Reading from the chromosome on the left, the first block is labelled ’C016R’. This represents the first gene of this chromosome. C indicates that this gene represents a convolution layer,

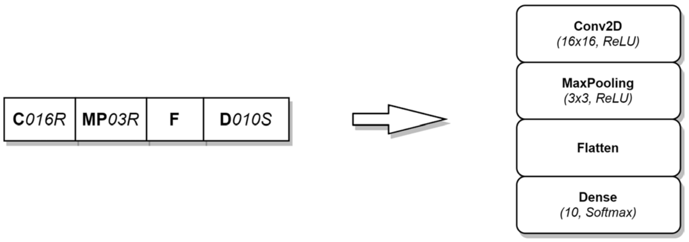

Fig. 4 A basic GA chromosome and the corresponding neural network derived from this chromosome

016 indicates that this convolution layer has 16 convolutions as its parameter, and R indicates that this layer uses a ReLU activation function. In the neural network on the right hand side, the top most block depicts the layer that the ’C016R’ gene represents, which is shown as a convolution layer with 16 convolutions and a ReLU activation function. The second block in the chromosome, representing the second gene, is labelled ’MP03R’. M indicates that this gene represents a max pooling layer, 03 indicates that this layer has a 3x3 kernel as its parameter, and R indicates that this layer uses a ReLU activation function. The layer in the neural network which this gene represents is depicted in the second block in the neural network on the right, that being a max pooling layer with a 3x3 kernel and a ReLU activation function. The third block in the chromosome, representing the third gene, is labelled ’F’. F indicates that this gene represents a flattening layer. Fattening layers do not have any parameters. Looking again at the neural network on the right, the third block depicts the representation of this layer. The fourth and final blocks of the chromosome on the left and the neural network on the right represent the final layer. The gene is labelled ’D010S’. D indicates that this gene represents a densely connected layer, 010 indicates that this layer has a size of 10 as its parameter, and S indicates that this layer uses a Softmax activation function.

The initial population is created by randomly creating each chromosome. The value for each gene is randomly selected from the set of options described above. The size of the population and the length of the chromosome are parameters of the GA.

5.2 Fitness evaluation and selection

The GA employs the same fitness function as GPNND, outlined in Sect. 3.3. The GA employs tournament selection. In tournament selection, a specific tournament size is chosen, and a random selection is made from the population, comprising the chosen number of individuals. Subsequently, the individual with the highest fitness within the tournament is selected for the application of a genetic operator.

5.3 Genetic operators

Mutation and crossover are used to create the new population of each generation. These operators are described in the sections that follow.

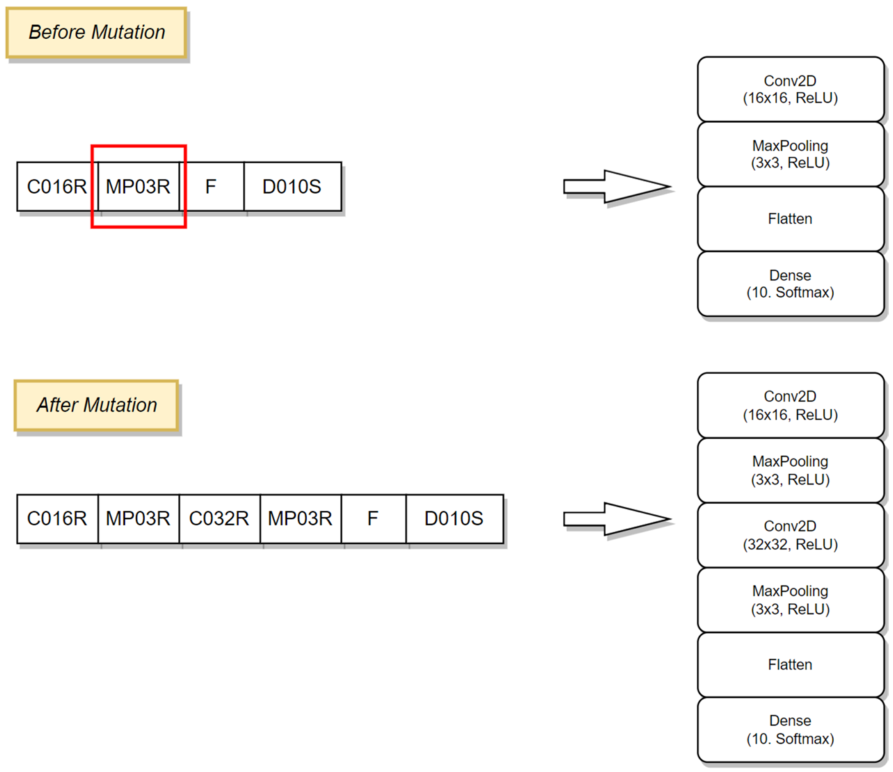

Fig. 5 An example of the Mutation operator for GA

5.3.1 Mutation

A mutation point is chosen at random in the parent selected using tournament selection. The mutation point corresponds to a gene in the chromosome. One of the following operations is applied to the gene:

- Replace the gene
- Add genes before this gene
- Add genes after this gene
- Remove the gene

An example of mutation is shown in Fig. 5.

Figure 5 depicts a chromosome and the neural network this chromosome represents, firstly before mutation occurs and then after mutation has occurred. The chosen mutation point is indicated by a red square, in this case highlighting the gene labelled ’MP03R’. From here, this mutation has made the choice to generate new genes and place them after its position in the chromosome. Two new genes were generated, namely ’C032R’ and a ’MP03R’. The neural network on the left

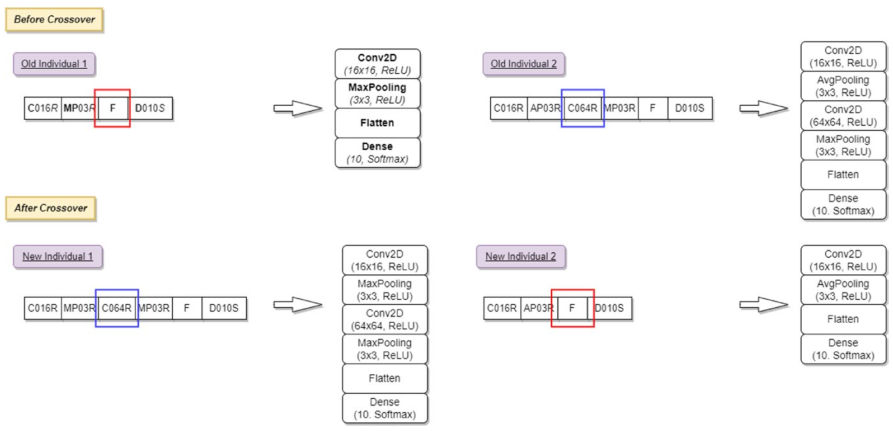

Fig. 6 An example of the Crossover operator for GA

shows the effect of the mutations before and after they take place, where two new layers have been added.

5.3.2 Crossover

Two parents from the population are selected using tournament selection. Crossover points are randomly selected in each of the parents. The fragments defined by the crossover points are swapped to produce two offspring. Figure 6 shows an example of the operator for GA The crossover points are indicated by red and blue squares.

In Fig. 6, The two parent chromosomes which have been selected are shown at the top of the figure, the two crossover points are indicated by red and blue squares. The section, beginning at the crossover points and ending at the final gene of the chromosomes, are interchanged with one another. The two offspring are shown at the bottom of the figure with the interchanged areas. If the crossover points chosen produce invalid chromosomes i.e chromosomes which do not correctly represent a correct neural network architecture, the new chromosome is discarded and crossover is retried. If crossover produces an invalid chromosome five consecutive times for the same two parent chromosomes, mutation is performed instead.

6 Experimental setup

This section describes the experimental setup used to evaluate GA, GPNND and ISBGP-II. Section 6.1 describes the datasets that were used. The performance metrics used to report the performance of the approaches are presented in Sect. 6.2. Parameters are listed in 6.3 and section 6.4 presents the technical specifications of the hardware and software for the experiments.

6.1 Datasets

This section describes the datasets used for this research. These datasets were chosen because the state-of-the-art approaches mentioned in Sect. 7.3 make use of these datasets, allowing for a comparison to be made between this research and state-of-the-art methods. Additionally, they contain a large number of samples which can be adequately split into training and testing sets. The large number of samples will help to reduce neural networks from overfitting. The following datasets were used to test the proposed approaches for image classification:

- **CIFAR-10** - This dataset contains 60,000 32x32 colour images. There are 10 classes. Each class contains 6000 images. 50,000 images are used for training and the remaining 10,000 for testing. [20]
- **Fashion MNIST** - A dataset containing images of various clothing items. There are 70,000 28x28 greyscale images in total, with 60,000 being used for training and 10,000 for testing. There are 10 classes. [49]
- **Street View House Numbers (SVHN)** - This dataset consists of 600,000 32x32 RGB images of digits on house number plating. Of these images, 73,257 are used for training and 26,032 are for testing with 10 classes. [34]
- **EuroSat** - A data set containing satellite images. In total there are 27,000 images and 10 classes. [34]

The following datasets were used for video shorts creation:

- **PHD2: Personalized Highlight Detection** - This dataset contains a selection of various YouTube video IDs, and the timestamp of the highlight of each video, along with some user-specific information. There are 12,972 training examples and 850 testing examples. [31]
- **Video2Gif** - A dataset containing 100,000 gifs along with the respective videos and related data which the gifs were extracted from. [12]
- **YouTube-8 M Segments Dataset** - This dataset contains semgments date from YouTube videos, of which there are about 237,000 segments on 1000 classes. There are 5 segments per video. [1]
- **QVHighlights** - This dataset consists of over 10,000 YouTube videos, each with annotated highlight information. [22]

6.2 Performance metrics

The following metrics will be used to evaluate the performance of the GPNND system:

- **Model Accuracy** - A measure of the correctness of the predictions of the neural network, represented as a decimal or percentage.

Table 4 Parameters

Name	Value	Justification
Population size	15	Allows for enough genetic variation and reasonable execution times
Minimum Tree Depth	2	This depth is enough for a minimal working tree
Maximum Tree Depth	20	Allows for ample room for trees to grow and promote variation without memory and computational issues
Number of Generations	25	Allows for enough time for the programs to sufficiently evolve
Number of NN Epochs	30	Allows for models to train sufficiently without overfitting
Crossover Probability	0.35	Promotes sufficient genetic variation and genetic collaboration
Mutation Probability	0.45	Introduces sufficient genetic variation without losing good changes
Cut-off depth	4	This depth is before which there are the most similarities between trees
Global index threshold	6	Allows for ISBGP to move the search space effectively for global areas
Local index threshold value	8	Allows for ISBGP to move the search space effectively for local areas

- **Design Time** - The time taken for the GPNND system to evolve a suitable architecture for the given dataset. This is measured in hours.

6.3 Parameters

The determination of parameter values for the GPNND and ISBGP approaches were guided by a review of related work [2, 4, 13], wherein initial values from previous studies were adopted as a foundational starting point. Subsequently, an approach was taken for each parameter, involving the exploration of values within ranges slightly smaller and larger than the initial settings. Through experimentation and testing, the algorithm's performance was evaluated across various parameter values. The decision to use the current parameter values emerged from this iterative process, as they demonstrated the best results for this research. A table detailing all the parameters and their justification is given in Table 4.

6.4 Technical specification

GA, GP and ISBGP-II are designed using Python 3. The TensorFlow, Keras and DEAP libraries are used. The experiments were evaluated on the Google Colab[1] suite, making use of a GPU runtime.

[1] https://colab.research.google.com/

Table 5 Table of results for image classification

Approach	Metrics	Dataset			
		Fashion MNIST	CIFAR-10	SVHN	EuroSat
GPNND	Best Design Time (hrs)	4.6	4.7	4.6	5.3
	Best Accuracy	0.9602	0.9531	0.9648	0.9296
	Average Accuracy (Train)	0.9774	0.9693	0.9785	0.9382
	Average Accuracy (Test)	0.9517	0.9396	0.9514	0.9127
GA	Best Design Time (hrs)	5.2	5.6	5.6	5.7
	Best Accuracy	0.9429	0.8916	0.9098	0.9106
	Average Accuracy (Train)	0.9619	0.9086	0.9191	0.9286
	Average Accuracy (Test)	0.9351	0.8831	0.8896	0.8963

Table 6 Table of results for video short creation

Approach	Metrics	Dataset			
		PHD2	Video2GIF	8 M	QVHighlights
GPNND	Best Design Time (hrs)	9.1	9.6	9.2	9.3
	Best Accuracy	0.8912	0.8783	0.8707	0.8376
	Average Accuracy (Train)	0.9105	0.8914	0.8973	0.8649
	Average Accuracy (Test)	0.8827	0.8679	0.8596	0.8297
GA	Best Design Time (hrs)	10.8	10.7	10.4	9.8
	Best Accuracy	0.7376	0.6891	0.7719	0.8202
	Average Accuracy (Train)	0.7587	0.7114	0.7936	0.8449
	Average Accuracy (Test)	0.7279	0.6825	0.7604	0.8164

7 Results

This section discusses the performance of the GA, GPNND and ISBGP-II approaches. For each of these approaches, 30 runs were performed for each dataset. Average accuracy and design times were collected and *P* values were calculated. The following sections document the results of the approaches.

Table 7 *P* values

Data set	*P* value	
	Accuracy	Design time
Fashion MNIST (Image)	0.0435	0.0163
CIFAR-10 (Image)	0.0232	0.0388
SVHN (Image)	0.0164	0.0249
EuroSAT (Image)	0.0242	0.0311
PHD2 (Video)	0.0329	0.0302
Video2GIF (Video)	0.0337	0.0226
QVHigilights (Video)	0.0253	0.0219
YouTube 8 M (Video)	0.0104	0.0168

Table 8 Table of results for image classification

Approach	Metrics	Dataset			
		Fashion MNIST	CIFAR-10	SVHN	EuroSat
GPNND	Best design time (hrs)	4.6	4.7	4.6	5.3
	Best accuracy	0.9602	0.9531	0.9648	0.9296
	Average accuracy (Train)	0.9774	0.9693	0.9785	0.9382
	Average accuracy (Test)	0.9517	0.9396	0.9514	0.9127
ISBGP-II	Best design time (hrs)	4.2	4.2	5.1	5.1
	Best accuracy	0.9775	0.9786	0.9815	0.9549
	Average accuracy (Train)	0.9847	0.9863	0.9885	0.9692
	Average accuracy (Test)	0.9637	0.9615	0.9753	0.9395

7.1 GPNND and GA approaches

The performance comparison for GPNND and GA for image classification and video shorts creation is documented in Tables 5 and 6 respectively.

A statistical comparison between GPNND and GA was performed. The null hypothesis states that the accuracy of GPNND is equal to that of GA and the design time of GPNND is equal to that of GA. The alternative hypothesis states that GPNND has a higher accuracy and lower design time than GA. The hypotheses were tested with a Welch's t-test, with a significance level of 0.05. Table 7 shows the *P* values for the accuracy and design time for this comparison. In the table, all *P* values are less than the significance level of 0.05 implying that the null hypothesis can be rejected in favour of the alternative hypothesis, thus GPNND has higher accuracy and lower design time than GA.

Table 9 Table of results for video short creation

Approach	Metrics	Dataset			
		PHD2	Video2GIF	8 M	QVHighlights
GPNND	Best design time	9.1	9.6	9.2	9.3
	Best accuracy	0.8912	0.8783	0.8707	0.8376
	Average accuracy (Train)	0.9105	0.8914	0.8973	0.8649
	Average accuracy (Test)	0.8827	0.8679	0.8596	0.8297
ISBGP-II	Design time	9	9.2	9.4	9.0
	Accuracy	0.9167	0.8949	0.8985	0.8682
	Average accuracy (Train)	0.9264	0.9162	0.9095	0.8892
	Average accuracy (Test)	0.9016	0.8877	0.8894	0.8573

Table 10 *P* values

Data set	*P* value	
	Accuracy	Design time
Fashion MNIST (Image)	0.0435	0.0163
CIFAR-10 (Image)	0.0232	0.0388
SVHN (Image)	0.0164	0.0249
EuroSAT (Image)	0.0242	0.0311
PHD2 (Video)	0.0329	0.0302
Video2GIF (Video)	0.0337	0.0226
QVHigilights (Video)	0.0253	0.0219
YouTube 8 M (Video)	0.0104	0.0168

7.2 ISBGP-II and GPNND approaches

The performance comparison for the ISBGP-II approach and GPNND for image classification and video shorts creation is documented in Tables 8 and 9 respectively.

A statistical comparison between ISBGP-II and GPNND was also performed. The null hypothesis states that the accuracy of ISBGP-II is equal to of GPNND and the design time of ISBGP-II is equal to that of GPNND. The alternative hypothesis states that ISBGP-II has a higher accuracy and lower design time than GPNND. The hypotheses were also tested with a Welch's t-test, with a significance level of 0.05. Table 10 shows the *P* values for the accuracy and design time for this comparison. In the table, all *P* values are less than the significance level of 0.05 implying that the null hypothesis can be rejected in favour of the alternative hypothesis, thus ISBGP-II has a higher accuracy and lower design time than GPNND.

Table 11 Comparison to state of the art methods for image classification

Approach	Dataset		
	Fashion MNIST	CIFAR-10	SVHN
GPNND	0.9602	0.9531	0.9646
ISBGP	0.9775	0.9786	0.9815
Other Methods	0.9691	0.9891	0.9901
	[44]	[45]	[7]
	0.9641	0.972	0.99
	[7]	[50]	[36]
	0.9444	0.0653	0.989
	[9]	[23]	[24]

7.3 Comparison with state of the Art

This section compares the performance of GPNND and ISBGP-II with state of the art approaches applied to the same datasets. Please note that this is only included for the sake of completeness. The aim of the research is not to obtain a performance improvement over the state of the art approaches but rather to investigate the benefit of using GP rather than GA for NAS. The comparison is shown in Table 11.

7.4 Evolved designs

This section showcases evolved designs which the GA, GPNND and ISBGP-II approaches produce for the image classification and video shorts creation. The figures depict a graphic representation of the evolved neural networks where each block represents a layer of the neural network. The GP trees produced and evolved in practice grow to be too large to plot effectively, thus making them not interpretable. The trees are therefore treated as a black box, and future work will investigate methods for making the classifiers more readable. The final neural network architectures, which are created by evaluating the evolved trees are therefore shown.

7.4.1 Evolved designs for image classification

Figure 7 shows the architecture of three neural networks, each evolved using the GA, GPNND and ISBGP approaches respectively for image classification:

In Fig. 7, graphic representations of three neural networks for image classification are shown. These are labelled ’GA’, ’GP’ and ’ISBGP’, referring to the three approaches. The best performing neural network is evolved by ISBGP and achieves an accuracy of 97.75%.

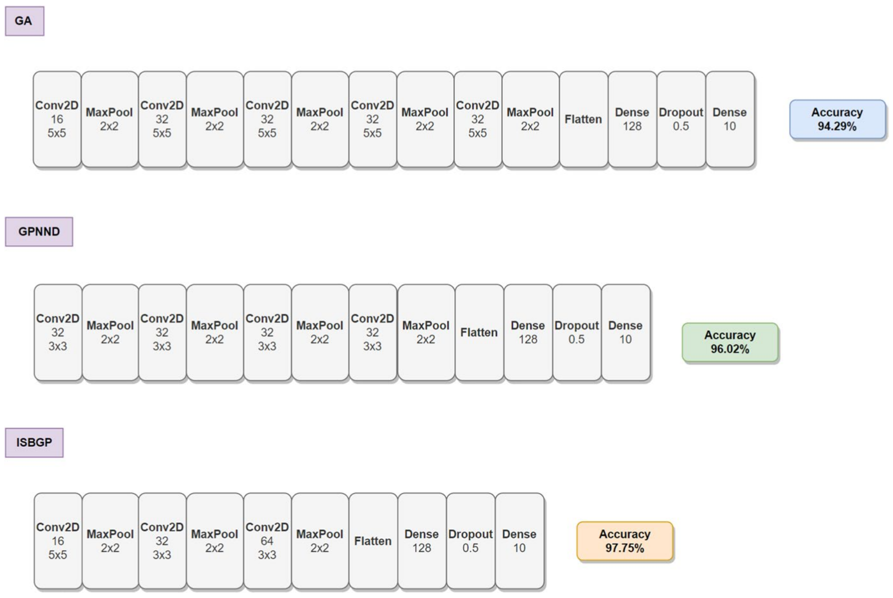

Fig. 7 Neural network architectures evolved by GA, GP and ISBGP-II for Image Classification

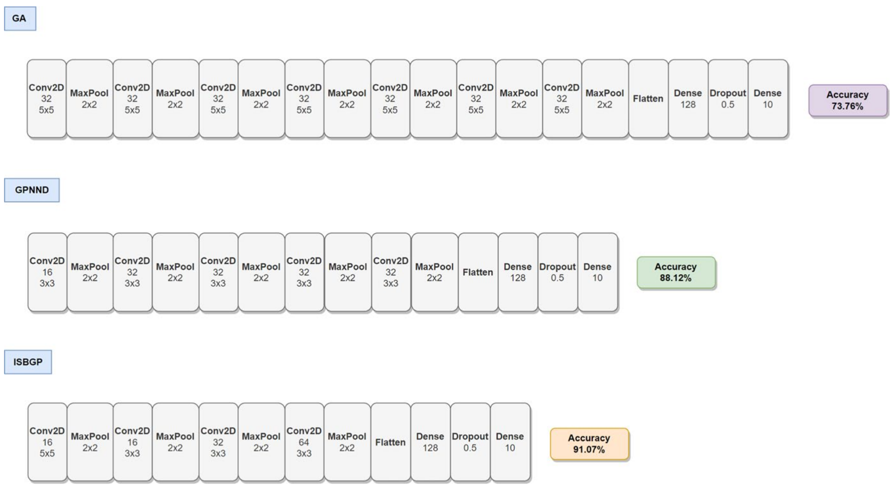

Fig. 8 Neural network architectures evolved by GA, GP and ISBGP-II for Video Shorts Creation

7.4.2 Evolved designs for video short creation

Figure 8 shows the architecture of three neural networks, each evolved using the GA, GPNND and ISBGP approaches respectively for video shorts creation:

In Fig. 8, graphic representations of three neural networks for video shorts creation are shown. These are labelled ’GA’, ’GP’ and ’ISBGP’, referring to the three approaches. The best performing neural network is evolved by ISBGP and achieves an accuracy of 91.07%.

For both image classification and video shorts creation, the neural netowrk for GPNND contains fewer layers than the neural network for GA, while achieving a higher accuracy. The same can be seen for ISBGP-II where the neural network contains fewer layers but achieves a higher accuracy. The layer parameter values are also different in the approaches. GPNND and ISBGP-II use different numbers of convolutions and kernel sizes for pooling in their respective layers.

8 Conclusions and future work

The main aim of the research presented in this paper was to study the use of genetic programming for NAS. Both canonical GP (GPNND) and a variation of GP which takes both structure and fitness into consideration when directing the search, ISBGP-II, were examined for NAS for image processing and video shorts creation. The performance of GPNND and ISBGP-II was also compared to a genetic algorithm (GA) for NAS. Both GPNND and ISBGP-II outperformed the GA for NAS for image classification and video shorts creation. ISBGP-II was found to perform better than GPNND as well as a previous version of the approach.

Given the effectiveness of transfer learning in GP [39], future work will examine the use of transfer learning in ISBGP-II for NAS. In addition to this, fitness approximation techniques will also be examined to reduce the computational cost of ISBGP-II. Future work will also investigate combination operators that will form part of the GP function set to combine layers in a neural network architecture.

Author Contributions R. Kapoor wrote the main manuscript. N. Pillay acted as supervisor and reviewed the manuscript.

Funding Open access funding provided by University of Pretoria. This work was funded as part of the Multichoice Research Chair in Machine Learning at the University of Pretoria, South Africa. This work is based on the research supported in part by the National Research Foundation of South Africa (Grant Number 138150). Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

Declarations

Conflict of interest The authors have no competing interests as defined by Springer, or other interests that might be perceived to influence the results and/or discussion reported in this paper.

References

1. S. Abu-El-Haija, N. Kothari, J. Lee, et al, Youtube-8m: A large-scale video classification benchmark. CoRR abs/1609.08675. http://arxiv.org/abs/1609.08675, (2016)
2. Y. Bi, B. Xue, M. Zhang, An evolutionary deep learning approach using genetic programming with convolution operators for image classification. In: 2019 IEEE Congress on Evolutionary Computation (CEC), pp 3197–3204 (2019). https://doi.org/10.1109/CEC.2019.8790151
3. P. Covington, J. Adams, E. Sargin, Deep neural networks for youtube recommendations. In: Proceedings of the 10th ACM Conference on Recommender Systems. Association for Computing Machinery, New York, NY, USA, RecSys '16, pp 191–198 (2016). https://doi.org/10.1145/2959100.2959190
4. S. Deng, Y. Sun, E. Galvan, Neural architecture search using genetic algorithm for facial expression recognition. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. Association for Computing Machinery, Boston, Massachusetts, GECCO '22, pp 423–426 (2022). https://doi.org/10.1145/3520304.3528884
5. J. Diniz, F. Cordeiro, P. Miranda, et al, A grammar-based genetic programming approach to optimize convolutional neural network architectures. In: Proceedings of the XV National Meeting of Artificial and Computational Intelligence, https://doi.org/10.5753/eniac.2018.4406 (2018)
6. T. Elsken, J.H. Metzen, F. Hutter, Neural architecture search: a survey. J. Mach. Learn. Res. **20**(1), 1997–2017 (2019)
7. P. Foret, A. Kleiner, H. Mobahi, et al, Sharpness-aware minimization for efficiently improving generalization. In: International Conference on Learning Representations, (2021) https://openreview.net/forum?id=6Tm1mposlrM
8. G. Franchini, V. Ruggiero, F. Porta et al., Neural architecture search via standard machine learning methodologies. Math. Eng. **5**(1), 1–21 (2022). https://doi.org/10.3934/mine.2023012www.aimspress.com/article/doi/10.3934/mine.2023012
9. P. Gavrikov, J. Keuper, Cnn filter db: An empirical investigation of trained convolutional filters. In: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp 19,044–19,054 (2022). https://doi.org/10.1109/CVPR52688.2022.01848
10. I.J. Goodfellow, Y. Bengio, A. Courville, *Deep learning* (MIT Press, Cambridge, MA, USA, 2016)
11. Z. Guo, X. Zhang, H. Mu et al., Single path one-shot neural architecture search with uniform sampling, in *Computer Vision - ECCV 2020*. ed. by A. Vedaldi, H. Bischof, T. Brox et al. (Springer International Publishing, Cham, 2020), pp.544–560
12. M. Gygli, Y. Song, L. Cao, Video2gif: Automatic generation of animated gifs from video. pp 1001–1009 (2016). https://doi.org/10.1109/CVPR.2016.114
13. T. Hassanzadeh, D. Essam, R. Sarker, Evodcnn: an evolutionary deep convolutional neural network for image classification. Neurocomputing (2022). https://doi.org/10.1016/j.neucom.2022.02.003
14. K. He, X. Zhang, S. Ren, et al, Deep residual learning for image recognition. 1512.03385 (2015)
15. Y. Jaafra, J. Luc Laurent, A. Deruyver et al., Reinforcement learning for neural architecture search: A review. Image Vis. Comput. **89**, 57–66 (2019). https://doi.org/10.1016/j.imavis.2019.06.005www.sciencedirect.com/science/article/pii/S0262885619300885
16. H. Jiang, Y. Lu, J. Xue, Automatic soccer video event detection based on a deep neural network combined cnn and rnn. In: 2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI), pp 490–494 (2016). https://doi.org/10.1109/ICTAI.2016.0081
17. R. Kapoor, N. Pillay, Iterative structure-based genetic programming for neural architecture search. In: Proceedings of the Companion Conference on Genetic and Evolutionary Computation. Association for Computing Machinery, New York, NY, USA, GECCO '23 Companion, pp 595–598 (2023). https://doi.org/10.1145/3583133.3590759
18. A. Klos, M. Rosenbaum, W. Schiffmann, Neural architecture search based on genetic algorithm and deployed in a bare-metal kubernetes cluster. Int. J. Netw. Comput. **12**(1), 164–187 (2022)

19. J.R. Koza, R. Poli, Genetic programming, in *Search methodologies*. (Springer, Berlin, 2005), pp.127–164
20. A. Krizhevsky, Learning multiple layers of features from tiny images. Tech. Rep (2009)
21. A. Krizhevsky, I. Sutskever, G. Hinton, Imagenet classification with deep convolutional neural networks. Neural Inf. Process. Syst. (2012). https://doi.org/10.1145/3065386
22. J. Lei, T.L. Berg, M. Bansal, Qvhighlights: Detecting moments and highlights in videos via natural language queries. (2021). https://doi.org/10.48550/ARXIV.2107.09609,
23. X. Li, W. Wang, X. Hu, et al, Selective kernel networks. pp 510–519, https://doi.org/10.1109/CVPR.2019.00060 (2019)
24. S. Lim, I. Kim, T. Kim, et al, Fast autoaugment (2019)
25. R. Lima, D. Magalhães, A. Pozo et al., A grammar-based gp approach applied to the design of deep neural networks. Genet Program Evol Mach (2022). https://doi.org/10.1007/s10710-022-09432-0
26. S. Litzinger, A. Klos, W. Schiffmann, Compute-efficient neural network architecture optimization by a genetic algorithm, in *Artificial Neural Networks and Machine Learning - ICANN 2019: Deep Learning*. ed. by I.V. Tetko, V. Kůrková, P. Karpov et al. (Springer International Publishing, Cham, 2019), pp.387–392
27. Y. Liu, Y. Sun, B. Xue et al., A survey on evolutionary neural architecture search. IEEE Trans Neural Netw Learn Syst (2021). https://doi.org/10.1109/TNNLS.2021.3100554
28. Y. Liu, Y. Sun, B. Xue et al., A survey on evolutionary neural architecture search. IEEE Trans. Neural Netw. Learn. Syst. **34**(2), 550–570 (2023). https://doi.org/10.1109/TNNLS.2021.3100554
29. A. McGhie, B. Xue, M. Zhang, Gpcnn: evolving convolutional neural networks using genetic programming. In: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp 2684–2691 (2020). https://doi.org/10.1109/SSCI47803.2020.9308390
30. J.F. Miller, *Cartesian Genetic Programming* (Springer, Berlin Heidelberg, Berlin, Heidelberg, 2011), pp.17–34. https://doi.org/10.1007/978-3-642-17310-3_2
31. A. Garcia del Molino, M. Gygli, PHD-GIFs: personalized highlight detection for automatic GIF creation. In: Proceedings of the 2018 ACM on Multimedia Conference. ACM, New York, NY, USA, MM ’18 (2018)
32. K. Muhammad, T. Hussain, M. Tanveer et al., Cost-effective video summarization using deep cnn with hierarchical weighted fusion for iot surveillance networks. IEEE Internet Things J. **7**(5), 4455–4463 (2020). https://doi.org/10.1109/JIOT.2019.2950469
33. M.H.T. Najaran, A genetic programming-based convolutional deep learning algorithm for identifying covid-19 cases via x-ray images. Artif. Intell. Med. **142**(102), 571 (2023). https://doi.org/10.1016/j.artmed.2023.102571www.sciencedirect.com/science/article/pii/S0933365723000854
34. Y. Netzer, T. Wang, A. Coates, et al, Reading digits in natural images with unsupervised feature learning. In: NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011 (2011). http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf
35. M. O’ Neill, C. Ryan, Grammatica Evolution Evolutionary Automatic Programming in an Arbitrary Language. Springer (2003)
36. N.H. Phong, B. Ribeiro, Rethinking recurrent neural networks and other improvements for image classification (2020)
37. E. Real, S. Moore, A. Selle, et al, Large-scale evolution of image classifiers. In: Precup D, Teh YW (eds) Proceedings of the 34th International Conference on Machine Learning, Proceedings of Machine Learning Research, vol 70. PMLR, pp 2902–2911 (2017). https://proceedings.mlr.press/v70/real17a.html
38. M.D. Ritchie, B.C. White, J.S. Parker et al., Optimizationof neural network architecture using genetic programming improvesdetection and modeling of gene-gene interactions in studies of humandiseases. BMC Bioinform. **4**(1), 28 (2003). https://doi.org/10.1186/1471-2105-4-28
39. J. Russell, N. Pillay, A selection hyper-heuristic for transfer learning in genetic programming. Association for Computing Machinery, New York, NY, USA, GECCO ’23 Companion, pp. 631–634 (2023). https://doi.org/10.1145/3583133.3590686
40. K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition. 1409.1556 (2015)
41. N. Srivastava, G. Hinton, A. Krizhevsky et al., Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
42. M. Suganuma, S. Shirakawa, T. Nagao, A genetic programming approach to designing convolutional neural network architectures. In: Proceedings of the Twenty-Seventh International Joint

Conference on Artificial Intelligence, IJCAI-18. International Joint Conferences on Artificial Intelligence Organization, pp 5369–5373 (2018). https://doi.org/10.24963/ijcai.2018/755,

43. M. Suganuma, M. Kobayashi, S. Shirakawa et al., Evolution of deep convolutional neural networks using cartesian genetic programming. Evolut. Comput. **28**(1), 141–163 (2020). https://doi.org/10.1162/evco_a_00253
44. M. Tanveer, M.K. Khan, C. Kyung, Fine-tuning darts for image classification. In: 2020 25th International Conference on Pattern Recognition (ICPR). IEEE Computer Society, Los Alamitos, CA, USA, pp 4789–4796 (2021). https://doi.org/10.1109/ICPR48806.2021.9412221
45. X. Wang, D. Kihara, J. Luo et al., Enaet: a self-trained framework for semi-supervised and supervised learning with ensemble transformations. IEEE Trans. Image Process. (2020). https://doi.org/10.1109/TIP.2020.3044220
46. M.T. Wu, C.W. Tsai, Training-free neural architecture search: a review. ICT Express (2023). https://doi.org/10.1016/j.icte.2023.11.001www.sciencedirect.com/science/article/pii/S2405959523001443
47. X. Wu, X. Zhang, L. Jia, et al, Neural architecture search based on cartesian genetic programming coding method (2021)
48. B. Xiao, X. Yin, S.C. Kang, Vision-based method of automatically detecting construction video highlights by integrating machine tracking and cnn feature extraction. Autom. Constr. **129**(103), 817 (2021). https://doi.org/10.1016/j.autcon.2021.103817
49. H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. cs.LG/1708.07747 (2017)
50. Z. Zhang, H. Zhang, L. Zhao et al., Nested hierarchical transformer: towards accurate, data-efficient and interpretable visual understanding. Proc. AAAI Conf. Artif. Intell. **36**, 3417–3425 (2022). https://doi.org/10.1609/aaai.v36i3.20252