GPCNN: Evolving Convolutional Neural Networks using Genetic Programming

Abigail McGhie, Bing Xue, Mengjie Zhang
School of Engineering and Computer Science
Victoria University of Wellington, P.O. Box 600, Wellington 6140, New Zealand
Email: andrew.mcghie@ecs.vuw.ac.nz; bing.xue@ecs.vuw.ac.nz; mengjie.zhang@ecs.vuw.ac.nz

***Abstract*—Image classification is an important task that has a wide range of applications. Convolutional neural networks (CNNs) are a common approach that can achieve promising performance in image classification. However, using CNNs to address a problem requires in-depth knowledge about CNN architectures and how it relates to the problem domain. Genetic programming (GP) as an evolutionary computation method can used to reduce the amount of knowledge required to design a CNN for a given problem domain by automatically searching for the optimal architecture. This paper proposes a new algorithm named, GPCNN, which encodes graph-based CNN architectures as trees and uses genetic operators, i.e. mutation, crossover and selection, to find better architectures. A more flexible crossover, partial subtree crossover, is also proposed to improve the search performance. As an preliminary work, GPCNN did not manage to achieve better performance than the state-of-the-art methods due to the limit on computational resource, but it is able to achieve better results than the baseline methods. More importantly, the proposed tree-based graph representation of CNN allows CNN architecture of various shapes, which has a great potential for future work in evolutionary automatic neural architecture search.**

***Index Terms*—Genetic Programming, Neural Architecture Search, Image Classification, Convolutional Networks**

I. Introduction

Image analysis is an area of machine learning with many real-world applications. Examples include fruit quality assessment, number plate identification, facial identification, and medical imaging analysis. Image analysis can be achieved more rapidly using computers with machine learning than by humans, and the accuracy levels achieved are often higher.

Convolutional Neural Networks (CNNs) are complex machine learning algorithms that are becoming larger and more difficult to build and understand. Increasing complexity makes it challenging for humans to design improved architectures, and a high level of domain knowledge is required. Automation of the architectural design of CNN's using Neural Architecture Search (NAS) algorithms will potentially overcome these limitations. NAS is an approach that automatically generates the architecture of a neural network.

There are two main ways that NAS can generate the architectural design of a CNN, either by the use of an Evolutionary Computation (EC) algorithm (NeuroEvolution) or by Reinforcement Learning. [1] evaluated five of each type of algorithm and found that EC based algorithms trained faster and were more accurate on average compared with Reinforcement Learning (RL) based algorithms. The Computational time of the final models was also compared, and it was found that models developed by EC had a smaller computational cost than models developed with a RL or Random search approach. NAS has been used with increasing success in recent years due to the increase in computational power available making their use more feasible. A more recent way of finding an architecture for a neural network is differential architecture search (DARTS) [2]. This generally uses weighted connection between parts of the network and a such can calculate a partial derivative with respect to the weight. This means that the gradient descent can be applied to these connections and therefore learned as with along with the weights on the neurons of the network. DARTS based methods typically have the advantage of been able to search faster than EC or RL based methods however due to having to evaluate all possible operations the models require lots of RAM and how the search space gets reduced so that models fit within the ram available becomes very important.

Automatically designing the architecture of a CNN has significant advantages in requiring neither domain knowledge of the problem or an in-depth knowledge about CNNs. It also has the potential to find previously unknown knowledge regarding applying CNNs to the domain.

Genetic Programming (GP) is an EC algorithm which has the advantage of representing individuals in the population as trees. This type of representation has more complex relationships between the parts of each individual and is important as state-of-the-art CNNs are no longer a sequence of layers, but rather Directed Acyclic graphs (DAG). CNNs use architectures with complex connectivity to allow the training of deeper networks. An example is the skip connections used by ResNet [3]. Trees are closer to DAG's than other encoding strategies such as binary strings and sequences of layers. A disadvantage of encoding strategies that use DAGs directly is that doing crossover is much harder due to their potentially been multiple connection point rather than just one. The increased complexity comes from having sections of layers with multiple inputs and outputs. Therefore, tree based GP has the unique position where complex connectivity can be expressed and it is possible to apply crossover operators. Applying GP to image classification is complex as images usually have a large amount of data to process with the individual pixel value combined with the large search space from using trees causes

978-1-7281-2547-3/20/$31.00 ©2020 IEEE

GP on image data to be computationally expensive.

A. Goals

The goal of this paper is to show how GP can be used to evolve the structure of CNN's to do image classification tasks.

- This project proposes the GPCNN algorithm that uses GP to evolve complex CNN architectures from a range of different types of layers and connections. A new and powerful encoding strategy is used to convert trees into CNN's that use complex connectivity rather than sequential layers.
- This project proposes a novel crossover technique that can exchange random partial subtrees rather than whole subtrees. The main advantage of this new crossover technique is that to change layers at the end of the network can require changing all layers that come before that layer. Exchanging partial subtrees makes it easier to search the space close to the fittest individuals.

II. Background

A. Convolutional Neural Networks and Genetic Programming

CNNs are a form of neural networks that were inspired by the connectivity of neurons in the visual cortex of cats [4]. They are a higher order neural network that has layers that use cross-correlations between different filters and images in the dataset to calculate feature maps. CNNs provide a way to construct localized features from an instance where the features are location invariant.

GP [5] is a form of Evolutionary Algorithm that represents individuals in its population as trees. For each generation, it applies a selection operator to the population to select high performing individuals before creating new individuals by mating (through a crossover operation) or mutation. The selection operator uses the fitness value associated with each individual to select high performing individuals. The crossover and mutation operators are applied randomly to the population. Individuals are typically represented as trees. The nodes in the individuals represent functions, and the leaf nodes represent constant values called terminals.

An extension of GP is Strongly Typed Genetic Programming (STGP) [6]. It adds constraints to the children of nodes so that they have to match types defined by the node. Each child to a node has to match a given type which can allow constraints to be put on the structure of the individuals.

B. Genetic Programming for Image Analysis

GP has been used to solve image classification problems in several ways. Two Tier GP has been developed by [7] to classify images. The lowest tier used aggregation functions that calculate metrics over part of an image parameterized by terminals on the node. The higher tier used classification functions combining the values from the lower tier to create the predicted class value.

[8] used three-tier GP for image analysis. The bottom tier used a function set made of convolution and pooling functions and where the terminals were specific convolutional filters. The middle tier applied aggregation to the image to find one feature. The top tier applied threshold functions to classify the instances to a class. Convolutional operators were learnt as part of the evolutionary process of this three-tier GP. There has been further work on this way of using GP by [9]. Multi-layer GP was proposed where region detection, feature extraction, feature construction and classification were performed by multiple layers on images.

C. Neural Architecture Search

Neural architecture search (NAS) is the name for a family of techniques that attempt to automate the design of neural networks. NAS is an alternative to handcrafting the architecture. The main drawback of NAS is the significant computation time required to evaluate a neural network architecture.The main advantage of NAS is that it requires less domain knowledge than handcrafted networks. NAS methods have been successfully applied to neural networks for the last two decades [10].

More recently handcrafted networks have fallen behind NAS methods. A number of the top algorithms on CIFAR-10 [11] use architecture search to some degree. GPipe [12] achieved 99% with 557 million parameters and applied NAS on much larger networks than had previously been possible. EfficientNet [13] used a grid search to scaled up a base network and achieved 98.9% accuracy with 64 million parameters. ProxylessNAS [14] used NAS with a hypernetwork to predict the weights on each individual to speed up searching and achieved 97.92% accuracy with just 5.7 million parameters.

D. Related Work

There are two families of encoding strategies: direct encoding and indirect encoding. Direct encoding is where the representation of the individuals are the layers and connection of the neural network. The operators in an EC algorithm would have to operate directly on individuals. Indirect encoding uses a separate data structure to represent the individuals to ensure that the evolutionary operators are compatible with the individuals. In the case of GA [15] and PSO [16] based methods, binary strings or sequences are used. In GP [17] trees are typically used.

EvoCNN [18] is an EC approach where each individual was represented as a sequence of layers. The mutation operator was used to add, delete or modify layers in the individuals allowing for variable length representations and the crossover operator exchanged parameters between the layers in the parent individuals. This limits the power of the crossover operator as genetic information about the architecture of the networks cannot be exchanged. Similarly, CNN-GA [19] used a sequence of layers where chosen from pooling or skip connection blocks rather than individuals convolutional layers. Building networks in this way will result in a linear sequence of blocks that have non-linear connections inside them. The blocks themselves are not optimized, so the search space of the algorithm is limited.

A Cartesian GP based approach was proposed in [20]. This approach creates an acyclic graph from the function sets by

connecting earlier layers with layers later in a sequence of layers. These individuals are modified only using a mutation operator and not a crossover operator, which limits the ability for an individual with high fitness improving other individuals.

In GP-CNAS [21] tree-based GP was used to design CNN architectures. The terminals were made up of predefined blocks of layers. The function set defined the hyperparameters of the blocks, such as the number of filters and strides. It also used dynamic crossover operation which favoured smaller changes in the second half of the generations. Similar to CNN-GA using predefined blocks of layers limits the search space of the algorithm.

The limitations of previous work involve reducing the search space to allow the evolutionary algorithm to be able to search the space efficiently or keeping a large search space but not allowing information to be passed between individuals.

III. Proposed Algorithm

A. Overview

In the proposed method, GPCNN, CNNs are encoded as trees. This encoding allows the genetic operators (mutation and crossover) to be applied to individuals allowing genetic information to be shared and created during evolution. Trees, however, are not able to represent the DAG structure of neural networks directly so an encoding strategy needs to be used.

The implementation of the various form of complex connections that make CNN architectures DAGs general. This allows the algorithm evaluate and find new motifs that allow CNN's to achieve high accuracy. However, this also means the search space will be large.

The tree structure consists of leaves representing the inputs. The root of the tree represents the output of the CNN. Each tree has a backbone which is a branch where the leaf node represents the input to the whole network. In the other branches off this backbone the leaves represent the input to a subsection of the graph.

B. Encoding/Representation

The function set contains two types of tree nodes. Ones that represent layers in the neural networks and ones that represent how the layers are connected.

C. Terminal/Function set

The function set contained 8 nodes:

- Layers:
  - Convolutional node
  - Activation node
  - Pooling node
  - Batch Normalization node
  - Dropout node
- Connections:
  - Skip connection
  - Densely connected
  - Auxiliary output.
  - Branching

The convolutional layers were represented by a convolutional node where the parameters for the number of filters and the size of the filters were represented as children that were enforced with Strongly typed GP. The number of filters that could be chosen were 8, 16, 32, 64, 128 and the filter size options were 3x3, 5x5, 7x7 and 9x9. Inputs to the convolutional layer was padded with zeros such that the size of the feature activation maps in the output of the layer was the same as input to the convolutional layer.

The activation layers used a Relu activation function as this is typically used in deep CNN architectures due to not having a vanishing gradient problem.

The pooling layer applied either max or average pooling. The pooling layers used a pool of size 2x2 and stride of 1x1. For each pooling layer the width and height of the activation maps were halved. If the dimensions of the feature maps was already 4x4 then the pooling nodes were ignored.

Batch normalization is used to reduce internal covariate shift and has been shown to have many positive effects on the training of neural networks, such as allowing the use of higher learning rates and regularizing the network making the network more generalized [22].

Dropout was used as another possible way to regularize individuals. This works by randomly setting some values to zero for an epoch effectively changing the architecture of the neural network each epoch [23]. Dropout will likely reduce the rate at which the neural network converge but will increase the accuracy which it converges to.

The ResNet architecture introduced the idea of skip connections which is where a point in the network is connected to a point earlier in the network. Doing this allows the loss to be propagated through the network without being distorted by weights near the output of the network. Skip connections were implemented by a node with two children. The first child represents the network before the skip connection node. The second child represents the layers that are in parallel with the skip connection, as represented in Fig. 1. The input node in the second child represents the output of the first child. The output of both children are concatenated together and used as the output from the skip connection node. Both the children of the skip connection node are required to contain a convolutional layer. The depth of the second child is limited to 5 nodes.

Densely connected blocks were proposed by [24]. This is where the output of convolutional layers is concatenated with the input of following convolutional layers. The encoding strategy of dense connectivity blocks used a node where the input to that node is concatenated with the inputs in a number of following convolutional layers. The number of convolutional layers that are connected to this is determined by a parameter which is a number between 2 and 6. An example of this can be seen in Fig. III-C.

GoogLeNet [25] introduced two notable motifs for neural network architecture. The first was auxiliary outputs where a neural network has multiple outputs. The network is then trained through joint back propagation where the loss of the network is propagated from each output. Propagating the loss

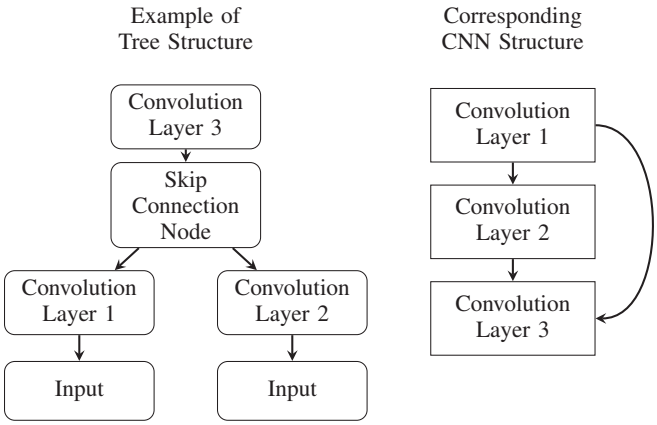

Fig. 1. Example of a Skip connection node.

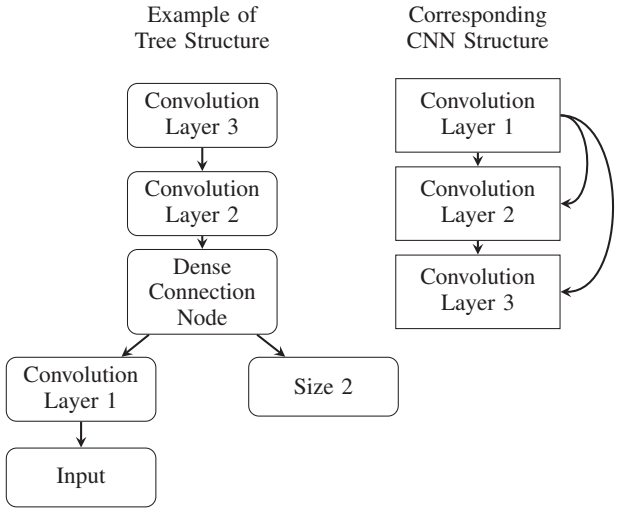

Fig. 2. Example of a Dense node.

from each output means that layers close to the input layer are much closer to an output so there will be a less distorted update to their weights. They are represented by a node with a single child that passes the input to the next layer in the network and also connects the input to that node to an output. The structuring of this can be seen in Fig. 3.

The second motif that GoogLeNet used was inception modules. These are a number of layers such a 1x1 convolution (Feature Pooling), 3x3 convolution, 5x5 convolution and 3x3 max pooling layers that are all connected to the same input. The output of each of these layers is concatenated together to form the output of the inception module. To implement a general version of this in the tree-based GP encoding strategy, branching nodes were introduced which are nodes with 3 children. The first child represented the network before the branching node. The second and third child represented the two branches whose outputs were concatenated together to form the output of the branch node. Fig. 4 gives an example of a branching. Branching nodes could be nested leading to more complex branching such as having more than two branches.

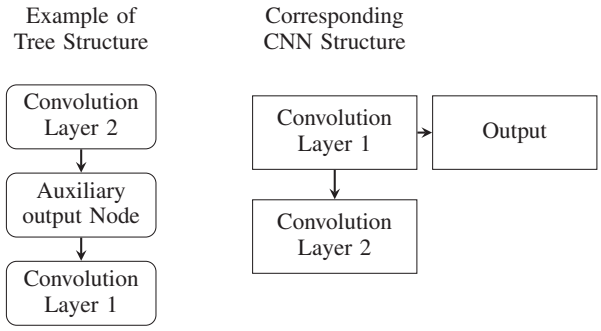

Fig. 3. Example of an Auxiliary output node.

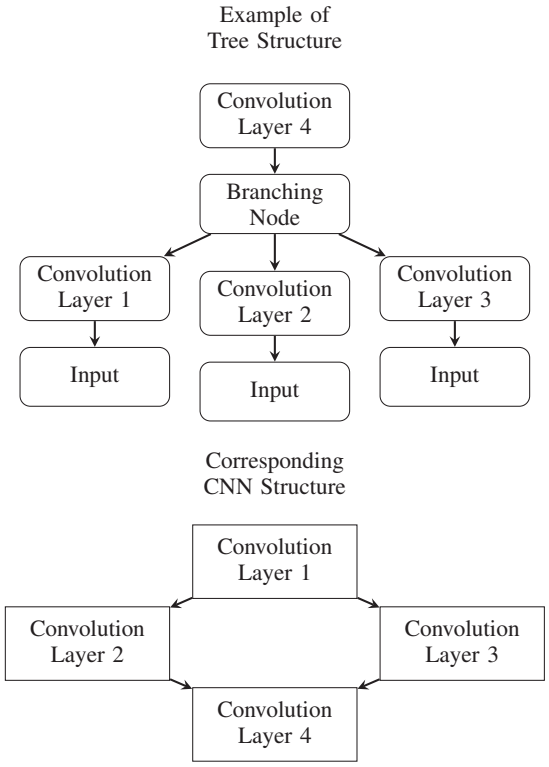

Fig. 4. Example of a Branching node in an individual.

The maximum depth of the branches was set to 5 to restrict the tree size and to avoid wide shallow networks. Both branches were also forced to contain at least one convolution layer. This was to avoid the cases where both branches did not perform any meaningful operation on the input.

D. Terminal Set

The terminal set for the tree-based GP algorithm was made up of parameters for the nodes and an input node. Each output in the networks used either global pooling or flatten operations to turn the output of the convolutional section into a one dimensional form.

E. Fitness Function

The fitness function used was an estimate of the accuracy of the individual. This was calculated by training the CNN for less epochs than is required to converge. This assumes networks that are more accurate at fewer epochs will be more accurate at convergence. This, however, is often not the case due to overtraining where some network converge much faster than other networks regardless of the accuracy after a few epochs.

F. Genetic Operators

In this paper two crossover were tested. The first one was the traditional subtree crossover. This is done by selecting a node in each individual and using the nodes as the root of the subtree and the place where to copy the subtree to. The other crossover method called partial subtree crossover is proposed to improve the crossover operator that GPCNN uses by increasing its flexibility so that it can exploit higher fitness individuals. Exchanging partial subtrees allows the crossover to make small changes on any part of the individual within one generation rather than requiring it to take several generations to make small changes. Taking several generations will require the individual to be reasonably accuracy at every step in the process so that the individual will get chosen by the selection operator. Allowing small changes to individuals will enable GPCNN to more easily evaluate the search space around the individuals, i.e. stronger local search, as a complementary ability of GP's global search ability.

One potential issue with exchanging partial subtrees is the vast number of partial subtrees that are possible for the typical size of the individuals that perform well on the CIFAR 10 dataset. The number of partial subtrees in a tree scales at a rate of 10^n where n is the number of nodes in the tree. Iterating through all these partial subtrees would be infeasible. However, a random signature can be chosen and then two partial subtrees that match these signatures can be extracted using little computation time.

Instead of iterating over all possible options, the process was broken down into six steps, where each one scaled better than 10^n. They are:

- Compress parameters
- Count frequency of signatures
- Filter signatures
- Choose signature
- Find roots which contain the chosen signature
- Turn the subtree into a partial subtree

The compress step compressed the trees so that the parameters on each node where included with the node. This will simplify the computation required. Since parameters are always leaves they can be independently changed with mutation rather than crossover.

To calculate the number of partial subtrees based on their signature. Signatures can be represented by the type of the root node and the number of each type of connection points. Trees are recursive data structures therefore signatures that use a node as the root can be counted by counting the signatures of the node's children and combining them. These frequencies can then be aggregated for all nodes in the tree to generate the frequencies for the signatures of any root.

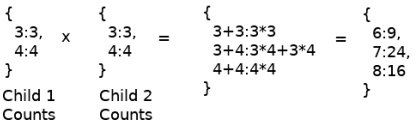

Fig. 5. Example combining frequencies from children.

As shown in Fig. 5, the frequencies of the children can be combined by multiplying the frequencies and summing the signatures from each child with the frequencies from all the other children. Calculating the frequencies for a tree can be achieved with Algorithm 1.

Algorithm 1: Count the frequencies of signatures in a tree

```
1  CountSignatures (R)
2      if ||children|| == 0 then
3          return {0:1}, {0:1} // there is only one
             possible subtree with 0 connection points
4      end
5      if ||children|| > 1 then
6          current = {} all = {}
7          foreach child in children do
8              current = combine(current,
                 CountSignatures(child)[0] + {1:1})
9              all += CountSignatures(child)[1] + {0:1}
10         end
11         return current, all + current
12     end
13
```

In Algorithm 1 Line 3 handles the case where the root is a leaf so the only possible partial subtree is the subtree that contains only the root. The algorithm has two return values because to calculate the frequencies of the subtrees that have a particular node as the root, the partial subtree frequencies of only the immediate children are needed, however, the final result of the subtree is the aggregation of this value for all nodes. Line 8 combines the frequencies of all the children with the extra subtrees that contain only the root node.

The filter step removes all signatures that are not present in both trees. If was no signatures in common with the two individuals, then the crossover was aborted. However, this case is extremely unlikely.

To choose a signature is a weighted random choice based on the number of subtrees with that signature. The index of the signature is a random number between 1 and the total number of partial subtrees with that signature. The root is found by utilizing depth-first search. The number of partial subtrees that match the signature at each node is summed. When the summation matches or exceeds the index of the signature, the search stops and that node is returned. This node

is the root of the partial subtree for the individual.

In order to turn the subtree into a partial subtree, a number of branches are chosen equal to the number of outputs in the signature. Each is assigned either the pre convolution or post convolution types in order to match the signature. If a branch is assigned the post convolution type, then the pre convolution section is removed. By cutting the chosen branches between the lowest common ancestor of any other chosen branch and the lowest viable node, the subtree can be turned into a partial subtree matching the signature. From here, the partial subtrees can be exchanged without violating any typing.

G. Voting

The GP algorithm creates a large number of individuals that do a reasonable job at classifying the images in the test set. Since evaluating individuals is computationally expensive only a small number of individuals will be evaluated. As only a small portion of the search space is explored there will be a high level of variability in the final population. Assuming that individuals that are diverse correspond to neural networks that have diverse behaviour, using a voting system that predicts class values based on the most commonly predicted class from a pool of the best individuals to provide better results than using only the best individual. The pool was found by adding the individuals that were most accurate on the evolutionary training set to the pool until the accuracy on the evolutionary training set started to decrease.

IV. Experiment Setup

A. Dataset

The CIFAR-10 dataset was used as the problem to test the GPCNN algorithm on. This dataset contains 60,000 instances split between 10 classes of images. The dataset was split into three sections; the gradient training set; the evolutionary training set; and the test set. The gradient training set consisted of 45,000 instances and was used to train the CNNs that were generated from the individuals in the population. The evolutionary training set was made up of 5,000 instances. The split was chosen so that there was as much as possible in the gradient training set while still having an adequate number of individuals to assess the accuracy of the 10 classes of the CIFAR-10 dataset. Stratified sampling was used so that the class ratios were all kept consistent. Fitness of the individuals was the accuracy on this dataset. The final split is the test set which was only used to evaluate the final solution after training.

B. Benchmark Algorithms

Three benchmark algorithms were chosen Support Vector Machine (SVM), Decision Tree (DT) and GP. These methods used each pixel of the image as independent features. The GP used a function set of standard maths functions such as +, -, * and protected division. It predicted the class value as a number with 0 to 1 being one class and 1 to 2 being the next class. The fitness function was the accuracy that the individual gained on the evolutionary training set. Each of the algorithms used the same train and test set as GPCNN was evaluated on to ensure the results are not dependant on the data split or the amount of data. This means that for the SVM and DT algorithms 5,000 of the instances were not used.

TABLE I
Parameter Settings Used for the Evolutionary Process

Parameter	Value	Justification
Crossover	0.35	Due to a low population size their is little genetic information at each generation
Mutation	0.65	To introduce genetic information to makeup for low crossover rate
Elitism	0.1	Ensure best solutions are not lost
Generations	20	Give time to evolve solutions
Population size	50	Set as high as possible with computation resources available
Initial Tree Depth	15-20	Favour small networks in early generations
Max Depth	30	Allow for larger networks

C. Diversity

During the evaluation of the algorithm, the diversity of the population was measured using the distance metric proposed by [26]. This method calculated the distance between two trees by adding dummy nodes to a pair of trees until the structures of both trees matched. Then comparing the trees node by node. The diversity in the population was measured by the average of the distances between all trees in the population.

D. Parameter Settings

While evaluating the fitness values networks were trained with two early stopping criteria. The main form of early stopping was a patience criteria where the training was stopped if the accuracy on the evolutionary training set did not increase within a certain number of epochs [27]. The other form of early stopping was a time limit of 20 minutes. All the training on the networks was done using Stochastic Gradient Descent (SGD) [28] with a learning rate of 0.01, momentum of 0, and weight decay of 0.

[22] found larger batch sizes worked better with SGD on CIFAR-10 and MNIST datasets. They used batch sizes ranging from 16 to 1024 and found that using a batch size of 1024 achieved the highest accuracy on both datasets. The drawback to using a larger batch size is the increased memory requirement of the GPU. Therefore, in this project, smaller networks were able to use a larger batch size whereas larger networks needed to use a smaller batch size to be able to be run on the GPUs available. The batch size for each network was started at 1024 and if the GPU ran out of memory the batch size was halved until either the network was able to be trained or the batch size got smaller than 32 where the network was deemed untrainable and the fitness value was set to -1. After the GP search is completed, the individuals were trained for a maximum of 60 epochs on the gradient training set.

V. Results and Discussions

The experiment results are shown in Table II, where T-test with 95% confidence interval was used to compare the performance of different methods.

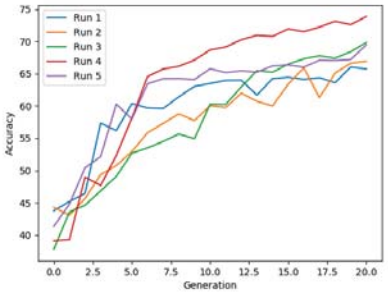

Fig. 6. Accuracy on test set of top 5 fittest individuals

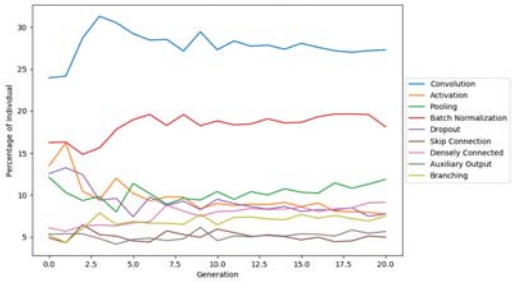

Fig. 7. Composition of 5 fittest individuals each generation

A. GPCNN with Traditional Crossover

The tree-based GP algorithm was biased towards models that reached high accuracy in under 10 epochs regardless of how much the accuracy will increase after 10 epochs. This caused the increase in accuracy between 10 epochs and 60 epochs to be small at 5.3% on average.

Voting was as effective at increased the accuracy of the algorithm as training for 60 epochs rather than 10. The average accuracy of the top individual was 71.4% at 10 epochs and 76.7% at 60 epochs while the average accuracy from voting was 81.2% as shown in Table II. These results indicate that there is significant diversity in the classification outputs of the neural networks created by the best individuals.

The diversity between the top 5 individuals of each run was 258.5 where as the average diversity between the top 5 in each run was only 53.3. The difference in diversity indicates that the starting population of the algorithm had a large effect on the outcome of the algorithm at 20 generations. It was expected that running the algorithm for more generations would decrease the diversity between runs as the different runs would overlap more as they searched throughout the search space. However, this did not occur in practice because of the small portion of the search space that was evaluated. As only 655 individuals were evaluated on average which is extremely small compared to the total search space.

None of the five runs of the algorithm converged to a local minimum. The lack of convergence is shown in Fig. 6, where the accuracy of the top individuals in each generation steadily increases over the 20 generations. Lack of convergence indicates that there is further potential to learn better classifiers. However, this would take significant computation time.

The difference in the composition of the fittest 5 individuals during the evolutionary process was small as seen in Fig. 7. The densely connected and branching nodes became more common the more generations were evaluated. The nodes that decreased the most were the activation nodes and the dropout nodes, showing that networks with more complex connectivity performed better than networks that were more sequential. The decrease in activation and dropout nodes was expected as activation layers with no convolutional layer between them is equivalent to having only one activation layer and networks with too much dropout train slowly.

TABLE II
CLASSIFICATION RESULTS ON THE CIFAR-10 DATASET.

Algorithm	Accuracy on CIFAR-10	Parameters	Seach cost (GPU days)
GPipe (AmoebaNet)	99%	557M	5250
EfficientNet	98.9%	64M	-
ProxylessNAS	97.92%	5.7M	8
DenseNet	96.54%	27.2M	-
NAS with RL	96.4%	7.1M	22400
GP-CNAS	94.57%	9.7M	180
CGP-CNN	94.02%	1.7M	27
GPCNN	81.2± 2.5%	8.9±9.5M	2.0±1.2
GPCNN with Partial Subtree Crossover	82.2±3.4%	4.7±3.5M	1.0±0.9
Decision Tree	26.33±0.8	-	-
SVM	53.89±0.2%	-	-
GP	10±0%	-	-

B. GPCNN with Partial Subtree Crossover

The size of the subtrees exchanged by crossover was much larger using partial subtree crossover than traditional crossover. This is because nodes close to the root of the tree have a greater number of partial subtrees causing these nodes to be chosen as the root of the subtree more often than nodes closer to the leaves of the tree. Traditional crossover is more likely to choose a nodes closer to the leaves than partial subtree crossover would and therefore choose smaller subtrees.

Using partial subtree crossover did not significantly increase the accuracy of the GPCNN algorithm with an average accuracy of 82.2% compared to the base GPCNN which had an average accuracy of 81.2%. Partial subtree crossover was more destructive but made better improvements. These effects cancelled each other out, leading to a very small change in the performance of the algorithm. Partial subtree crossover did not effect the lack of convergence seen in GPCNN as all 5 runs of GPCNN with partial subtree crossover did not converge.

GPCNN did not achieve an accuracy close to that of existing algorithms when evaluating them on the CIFAR 10 dataset. The main reason was the lack of runs due to constraints in computational power available. The algorithms developed were run for significantly less time than competing algorithms and it was only feasible to do 5 runs for each algorithm.

VI. CONCLUSIONS

The work presents the first research into the use of tree-based GP to automate the architecture design of CNNs. While

the approach did not outperform state-of-the-art hand-crafted CNN architectures, it showed that using tree-based GP is a viable approach. The results can be used as a baseline for future research in this area.

In the proposed method, partial subtree crossover looked like it had promise however it was much more destructive on average compared to the traditional crossover. This hid any benefits that exchanging partial subtree had. Using limited hardware such as only one GPU per run means that the algorithm takes significant time to get results similar to the state-of-the-art methods, since organisations such as Google that have a large amount of resources at their disposal and can therefore use many GUPs to run their algorithms. The results that GPCNN achieved on the CIFAR-10 dataset shows that it is a viable approach to automating the design of CNN architecture and can be used as a baseline for further improvements. GPCNN, however, did not perform well compared with existing state-of-the-art algorithms, as expected, being the first research in this area. There are two causes for the lower performance. The first, that GPCNN was not run for as long as the existing algorithms. GPCNN showed that after 20 generations it had not reached a local minimum. However, it was still making steady improvements. The second reason, related to the first, is that the search space for the GPCNN was very large due to the use of trees for encoding the CNN architectures.

VII. Future Work

As the learning capacity of GPCNN has not been fully explored in this work due to computational restrictions, it would be valuable to run the base GPCNN algorithm and the modifications until they converge. The learning capacity of each would then be established and can be compared. Then the effect of partial subtree crossover will be able to be evaluated with respect to the learning capacity and, furthermore, potentially validate the expectation that it would increase the learning capacity by making exploitation easier.

Partial subtree crossover could be improved by reducing the average size of the partial subtrees that are exchanged and so reduce how destructive the operator is. A proposed method could be to choose a random node as the root of the partial subtree then choose a random partial subtree. This would ensure that the partial subtree crossover is less destructive than the traditional subtree crossover as the maximum size of the partial subtrees would be smaller than or equal to the size of the complete subtree.

References

[1] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," *arXiv preprint arXiv:1802.01548*, 2018.

[2] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[4] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[5] J. R. Koza and J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*, vol. 1. MIT press, 1992.

[6] D. J. Montana, "Strongly typed genetic programming," *Evolutionary computation*, vol. 3, no. 2, pp. 199–230, 1995.

[7] H. Al-Sahaf, A. Song, K. Neshatian, and M. Zhang, "Two-tier genetic programming: Towards raw pixel-based image classification," *Expert Systems with Applications*, vol. 39, no. 16, pp. 12291–12301, 2012.

[8] B. Evans, "Genetic programming for evolutionary deep learning for image classification," 2017.

[9] Y. Bi, B. Xue, and M. Zhang, "An automatic feature extraction approach to image classification using genetic programming," in *International Conference on the Applications of Evolutionary Computation*, pp. 421–438, Springer, 2018.

[10] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[11] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," tech. rep., Citeseer, 2009.

[12] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q. V. Le, and Z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *arXiv preprint arXiv:1811.06965*, 2018.

[13] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.

[14] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018.

[15] J. Sadeghi, S. Sadeghi, and S. T. A. Niaki, "Optimizing a hybrid vendor-managed inventory and transportation problem with fuzzy demand: an improved particle swarm optimization algorithm," *Information Sciences*, vol. 272, pp. 126–144, 2014.

[16] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, pp. 1942–1948 vol.4, Nov 1995.

[17] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and computing*, vol. 4, no. 2, pp. 87–112, 1994.

[18] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Transactions on Evolutionary Computation*, vol. 24, pp. 394–407, 2019.

[19] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Automatically designing cnn architectures using genetic algorithm for image classification," *arXiv preprint arXiv:1808.03818*, 2018.

[20] M. Suganuma, M. Kobayashi, S. Shirakawa, and T. Nagao, "Evolution of deep convolutional neural networks using cartesian genetic programming," *Evolutionary Computation*, vol. 28, no. 1, pp. 141–163, 2020.

[21] Y. Zhu, Y. Yao, Z. Wu, Y. Chen, G. Li, H. Hu, and Y. Xu, "Gp-cnas: Convolutional neural network architecture search with genetic programming," *arXiv preprint arXiv:1812.07611*, 2018.

[22] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[24] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[26] A. Ekárt and S. Z. Németh, "A metric for genetic programs and fitness sharing," in *European Conference on Genetic Programming*, pp. 259–270, Springer, 2000.

[27] L. Prechelt, "Early stopping-but when?," in *Neural Networks: Tricks of the trade*, pp. 55–69, Springer, 1998.

[28] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*, pp. 421–436, Springer, 2012.