VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Genetic Programming for Evolutionary Deep Learning for Image Classification

Ben Evans

Supervisors: Mengjie Zhang, Bing Xue, Harith Al-Sahaf

Submitted in partial fulfilment of the requirements for
Bachelor of Science with Honours in Computer Science.

Abstract

Image classification is used for many tasks such as recognising handwritten digits, identifying the presence of pedestrians for self-driving cars, and even providing medical diagnosis from cell images. The current state-of-the-art solution to image classification typically uses convolutional neural networks, however, there are limitations in this approach such as the need for manually crafted architectures and low interpretability. A genetic programming solution is proposed in this study that aims to overcome these limitations, while also taking advantage of useful operators in convolutional neural networks such as convolutions and pooling. The new approach is tested on four widely used image datasets, both as a classification method and as a form of feature construction, and has provided competitive results for both. The resulting programs are highly interpretable, and visualisations of the evolved programs revealed interesting patterns within the programs. The constructed features resulted in a significant reduction in the original dimensionality of the images, and drastically speed up both training and testing time compared with using the raw pixels.

Acknowledgments

Thank you to my supervisors Mengjie Zhang, Bing Xue and Harith Al-Sahaf.

Contents

Figures

Chapter 1

Introduction

Computer vision enables computers to interpret and understand digital images, which is something which humans are able to do autonomously with high levels of accuracy.

Image analysis is an interesting area within computer vision, with important real-world applications such as face recognition [4], medical diagnosis [5], and autonomous vehicles [6]. There are many tasks in image analysis, such as segmentation, transformation, and classification. Segmentation aims to divide an image into multiple related segments, transformation aims to convert an image to a new representation, and classification aims to assign a label to an image. The focus of this project will be on image classification.

1.1 Motivations

The current state-of-the-art solution to image classification is using convolutional neural networks, for example, ImageNet [7] has held the "Large Scale Visual Recognition Challenge" (ILSVRC) since 2010 and from 2012 all top performing methods have used a convolutional neural network. However, there are limitations with this approach such as low interpretability, the need for manually crafted architectures and the need for large amounts of training data. The aim of this project is to overcome some of these limitations.

Convolutional neural networks can often appear like a black box, so visualising the solutions (weights and filters learned) is an important but challenging area of research. Simonyan et al. [8] presented a method to visualise the class saliency for a given image. This method visualises the contribution of each individual pixel, so gives a good idea of regions of interest within an image. However, it still has the limitation that the intermediary steps are not visualised. DeconvNets [9] are a step in the right direction to overcome this limitation, however deeper nets still remain relatively uninterpretable.

The success of a convolutional neural network is largely dependant on the underlying architecture of the network, and these architectures are increasingly complex and often manually crafted via a trial and error approach. For example, LeCun's [10] original convolutional neural network architecture was 7 layers deep, and the newer ResNet [11] was 152 layers deep. Suganuma et al. [12] showed Genetic Programming (GP) can be used to automatically construct a convolutional neural network architecture which can outperform existing manually crafted architectures, however the time taken to do so is prohibitively slow for most users (this took several weeks to learn).

Another factor determining the success of a convolutional neural network is the amount of training data available. Deep neural networks require a large amount of training data to effectively learn, and when there is only a small amount of data available other techniques such as data augmentation [13] are often utilised. Data augmentation involves modifying the original image in some way to artificially enlarge the training data, for example by scaling or cropping the image. Data augmentation can be useful, however still requires a modest amount of data to adequately generalise.

Other promising approaches to image classification are ones which utilise evolutionary principles to automatically evolve a program to perform image classification, such as GP. GP has several advantages over convolutional neural networks such as the high interpretability of the evolved programs [14], the automatically evolved architecture, and ability to learn from a relatively small number of instances [15]. However, most pure genetic programming approaches do not currently achieve accuracy as high as convolutional neural networks. It would also be interesting to investigate whether the extracted and constructed features learnt from GP can be useful for other classification methods.

1.2 Goals

A classification method which could achieve the accuracy of convolutional neural networks, and interpretability and flexibility of genetic programming would be ideal. The goal of this project is to incorporate various key design characteristics of convolutional neural networks (such as the convolution and pooling operators) into a deep genetic programming structure. The expectation is that this new model outperforms existing evolutionary and deep learning classification methods. The specific objectives of the project are the following

1. Develop a novel deep genetic programming structure that can be used to correctly classify images by automatically detecting interesting regions from the images, performing feature extraction on these regions, and then automatically constructing higher level features from these extracted features.

2. Visualise and interpret the features automatically extracted, constructed and learned by the evolutionary learning process.

3. Investigate whether the extracted/constructed features automatically learned can be useful for other classification algorithms, such as Nearest Neighbour or Decision Trees.

1.3 Major Contributions

In this project, a novel approach utilising the advantages of two promising methods, GP and convolutional neural networks, has been developed to address binary image classification problems. The major contributions are:

1. This project has shown how to design and develop a deep program structure for GP to automatically evolve/learn genetic programs for image classification. This was achieved by using a tiered structure enforced by strongly-typed GP, and featuring a tier for convolutional operators on the input image, a tier for aggregating regions of the image, and a tier for classification of the image. With the developed architecture, high level features were able to be built from raw images using the convolution operators, and informative regions of these images were automatically detected. The automatically evolved programs were able to achieve comparable average testing accuracy to general and GP based classification methods. While the evolved programs did not always outperform convolutional neural networks in terms of classification accuracy, they achieved faster classification on the testing set.

2. This project has shown how to interpret the learned (deep) programs with an intuitive representation so that humans are able to understand the deep structures. This was achieved by visualising the evolved programs as tree structures, and representing

nodes of the trees as the intermediary steps of the algorithm. For example, convolutional nodes are represented as the outcome of the convolution, and region selection nodes are represented by highlighting the regions used. This has proved useful for interpretability of the programs, and is an advantage over convolutional neural networks where it is difficult or even impossible to visualise each intermediary step of the program due to the large number of parameters in such networks.

3. This project has shown the evolved programs can also be used for automatic feature construction, by utilising the output of the aggregation tier as a set of constructed features. The automatically constructed features on average were able to achieve equivalent or in some cases even slightly improved classification accuracy over using the raw pixels of the image, while offering drastically reduced dimensionality and significantly faster training and testing times.

Part of the research is under preparation to be submitted to 2018 IEEE World Congress on Computational Intelligence/IEEE Congress on Evolutionary Computation (WCCI/CEC2018) or 2018 Genetic and Evolutionary Computation Conference (GECCO 2018).

1.4 Report Organisation

The remainder of this report is structured in the following way. Chapter 2 presents background to the work, including a general overview of key areas and related methods. Chapter 3 presents the newly developed solution and comparison results on four datasets. Chapter 4 visualises the best performing individuals for each dataset, and gives an insight to what filters have been evolved. Chapter 5 looks at the evolved trees as not just a method of classification, but also at the potential of using the learned features with other classifiers. Chapter 6 concludes the report by examining the major contributions made and outlining potential future work.

Chapter 2

Background

An overview of current work in machine learning and computer vision is given, placing particular focus on the use of genetic programming and deep learning for the purpose of image classification.

2.1 Computer Vision

The goal of computer vision is to allow computers to interpret images [16], something which humans are naturally very good at. For example, viewing an image of a boat in the sea, it would be immediately clear to a human what this image was and where the boat was positioned within the image. For a computer, the task is much more difficult, as the image is simply represented as pixels.

There are many tasks within computer vision such as image classification, object recognition, and object detection [17]. Image classification aims to assign a label to the image as a whole, object recognition aims to determine all of the objects within an image and their positions, and object detection is similar to object recognition except only dealing with a single class of objects.

The focus of this project will be on image classification due to the multitude of real-world applications. Image classification is the process of assigning a class label to an image. For example, determining if a given image contains a pedestrian or not. This is an example of binary classification, which will be the focus of this project. Another type of image classification exists, which is multi-class image classification. An example of multi-class classification is given a set of animal images, assign a label to each image such as "monkey", "lion", and "dog" based on what animal was photographed. Binary classification was chosen as a starting point to evaluate the usefulness of the proposed solution, this could be extended in the future to deal with multi-class classification.

2.2 Machine Learning

Machine learning algorithms are concerned with learning and improving from experience [18]. Machine learning has a huge range of applications in many industries, for example, machine learning is used for fraud detection, determining credit scores, optimization, and is also used extensively in computer vision.

There are various types of learning which can be broadly split into four categories: supervised, semi-supervised, unsupervised and reinforcement learning. Supervised learning is where we have a set of instances with desired outputs and the aim is to predict the output for some unseen data (e.g. classification and regression). In unsupervised learning there are no labelled outputs and the aim is to create groups of related data (e.g. clustering).

Semi-supervised learning is where only a subset of the training data contains labels. Reinforcement learning is based around the idea of rewarding actions which lead to a desirable state. Image classification is an example of supervised (or semi-supervised) learning.

In supervised learning, the data is split into two (potentially three) datasets: the training, test, and optional validation set. The training set is used to train the model, and the test set is used to evaluate the performance of the learned model. It can be easy to train a model to achieve a high training accuracy, but doing so can often lead to overfitting (the testing performance decreasing). The generalisation of a model is important, where the model should be general enough that it still performs well on unseen data (the test set) and not just the training data. The validation set can be used to monitor the training process, and help protect against overfitting to the training set.

There also exists multiple learning paradigms [19]: evolutionary learning, connectionist learning, symbolic learning, bayesian learning and analogical learning. The two paradigms of concern here are connectionist (i..e convolutional neural networks) and evolutionary learning (i.e. GP), however, methods from other paradigms will also be tested against the developed solution.

2.2.1 Deep Learning

Deep learning is a subfield of machine learning following the connectionist paradigm. Deep learning approaches try to mimic what occurs in a human brain. The key component of deep learning is neural networks. These networks are composed of neurons and connections between these neurons (similar to synapses and axons in a human cell). Neurons are organised into layers, and neural networks which have multiple hidden layers are considered "deep".

Convolutional Neural Networks

A typical type of neural network for image processing is convolutional neural networks (ConvNets). The difference between "standard" neural networks and ConvNets is that ConvNets use convolutions in place of standard matrix multiplication in at least one of the layers [20].

The convolution operation applies a filter to an image. For example, consider the image shown in Figure 2.1, this shows the application of a filter on a single pixel, this is then repeated across the entire image. Using stacked layers of these convolutions means we can build up multiple activation layers.

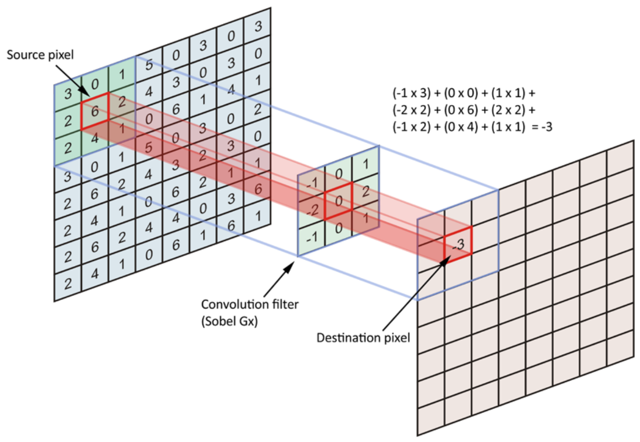

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Figure 2.1: Convolution of a Pixel [1].

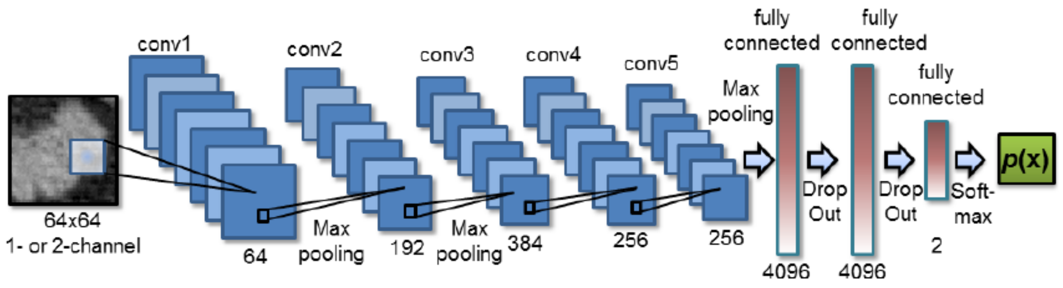


Figure 2.2: Sample ConvNet Architecture [2].

The typical ConvNet architecture is shown in Figure 2.2. The input is the raw image, the first layer (after the input) is then a convolution layer. There can be multiple convolution layers, and in between these convolution layers there are often pooling and activation layers. Pooling layers are used as a form of dimensionality reduction, and the most common pooling operation used is max pooling. Activation layers are used to introduce non-linearity into the network, a common activation function used is ReLU, which is simply $max(0, x)$, where x is the input value. Other activations such as the sigmoid function can also be used.

The final layer is a fully-connected layer (there can potentially be several fully connected layers). The fully connected layer works on the previous layer (of high-level features), and converts these features to class probabilities. A softmax function is often used in the fully connected layer, which means the output of the fully connected layer is a K-dimensional vector, where K = number of classes, containing the normalised probabilities of the image belonging to a particular class.

Convolutional neural networks have had much success in the area of image classification, which is likely because the stacking effect of the convolution layers can be used to represent high level features (or abstractions) of the image at each step. For example, the first layer may find the edges, the next layer groups these edges into shapes, and the layer after this then recognizes these shapes as belonging to a particular class. Having multiple layers can be particularly useful, as instead of dealing with raw pixels at each step, we can build off the resulting features from the previous step. However, ConvNets are not without limitations. The limitations of ConvNets (more widely deep learning) are that

1. Deep learning approaches can often be seen as a "black box". This means that while the output may be correct, it is difficult to see how this output was reached.

2. The architecture of the model (e.g. number of layers, number of nodes) is difficult to construct. This is often manually crafted via a trial and error approach, which is both time consuming and requires domain knowledge. Finding experts in the field can also be difficult and their services are costly.

3. Deep learning requires a large number of instances to learn effectively, for example, the AlphaGo system used 30 million instances to learn the game go. These types of resources are often not available as many problems simply do not have enough data to train the classifier.

4. There is a high computational cost associated with deep learning, so even if there is enough data available, the time it takes to train the classifier can be too long.

2.2.2 Evolutionary Computation

Algorithms based on the theory of evolution can be broadly referred to as evolutionary computation (EC) methods, which follow the evolutionary paradigm of learning. The idea is to mimic Darwin's theory of evolution, a (random) initial population of individuals is created, these individuals are then evaluated and updated, with the best individuals making it through to the next generation. The process is then repeated, and the "best" individuals should progress over time [21]. The main EC methods are evolutionary programming, evolutionary strategies, genetic algorithms and genetic programming. Genetic programming will be the focus of this project, due to the flexible representation interpretability of the model, and ability to hybridise with deep neural network structures.

Genetic programming

Genetic programming (GP) is an evolutionary computation method where individuals are typically represented as a tree based structure (referred to as programs). For example, the mathematical formula $1 + 2 * 5$ could be represented as (ADD 1 (MUL 2 5)). This is called prefix notation and the notation lends itself naturally to trees (shown in Figure 2.3).

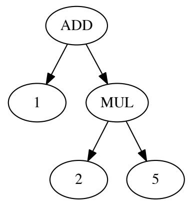


Figure 2.3: Sample tree representing the formula ADD 1 (MUL 2 5).

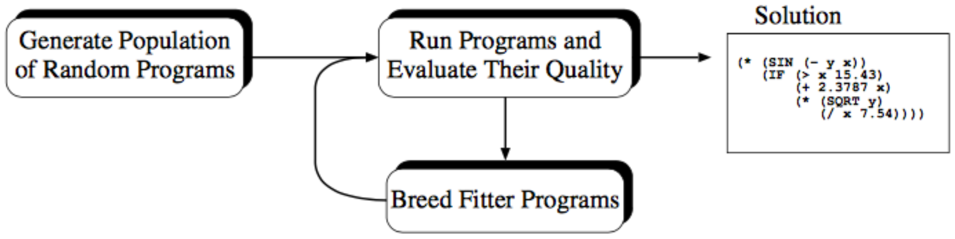
Some key terminology for programs is that leaf nodes (nodes without any children) are referred to as terminals, and internal nodes (nodes with children) are referred to as functions. A terminal set and function set must be defined by the user for use in the programs, a key consideration is that the terminals and functions should satisfy both closure (any function can accept the output of any other function or terminal) and sufficiency (a combination of the functions/terminals can solve the problem).

Another key part of GP is selecting or designing an appropriate fitness function. The fitness function is used to evaluate individuals, so it is important that the fitness function can determine the "goodness" of a program. For example in a regression problem, the fitness function could simply be the difference between the expected value and the predicted value, this way programs with a lower fitness are deemed more desirable, with a fitness of 0 being ideal and infinite being the worst. For a classification problem, the fitness function could be the percentage of instances correctly classified (classification accuracy), where 100 is the ideal and 0 is the worst.

One special type of GP is strongly typed GP, proposed by Montana in [22]. With strongly typed GP, constraints are added to the types of data which a specific function inputs and outputs, this restriction allows a structure to be enforced in the programs.

Once the terminal set, function set, fitness function, and optional structure have been created, the process of generating a good solution is fully automatic by using the genetic operators. The general process of GP is shown in Figure 2.4

The initial population of GP is created randomly. There exist several methods for generating this random population such as full, grow and ramped half-and-half. With the full method, functions are selected until the tree reaches a given maximal depth, then terminals are selected. This results in a complete and full tree. With the grow method, each node can either be a function or a terminal regardless of the depth, which means the resulting tree is not necessarily going to be full as there will potentially be branches of different lengths. Ramped half-and-half combines the two methods, generating half of the population from the full method and half from the grow method. Programs are then evolved using crossover, mutation and elitism operators [23]. Crossover is combining parents to produce two children, mutation is randomly changing parts of a program, and elitism (or reproduction) is directly copying a good program into the next population.


Generate Population of Random Programs
Run Programs and Evaluate Their Quality
Breed Fitter Programs
Solution

```
(* (SIN (- y x))
   (IF (> x 15.43)
       (+ 2.3787 x)
       (* (SQRT y)
          (/ x 7.54))))
```

Figure 2.4: GP Process [3].

GP will continue to evolve (breed) a population of programs until a stopping criterion has been met (such as a particular fitness value is achieved, or a number of generations passed), the best program can then be returned.

GP can help overcome some of the limitations associated with deep learning

1. GP solutions are commonly represented as a tree structure, which means they can often be visualised and interpreted.
2. The architecture of the solution evolves automatically using crossover and mutation operators on the selected well performing individuals
3. Depending on the fitness function used, GP can effectively learn from a very small number of instances.

However GP also has there own set of disadvantages. Unfortunately, due to the evolutionary nature, there is still a high computational cost associated with GP so this limitation persists. Since there is an element of randomness associated with GP, there is also no guarantees of finding the global optima and depending on the parameter values (particularly the mutation rate), and we could end up getting stuck in local optima in early generations. Selecting an appropriate fitness function and parameter values are important for producing good solutions. Furthermore, the function and terminal sets must be carefully considered to meet the closure and sufficiency requirements.

2.3 Related Work

The accuracy of ConvNets with the interpretability of GP would be an ideal solution, so a hybrid approach combining the two will be developed. This project will focus on the use of GP for binary image classification, utilising the structure seen in convolutional neural networks.

2.3.1 Genetic Programming for Image Classification

Before discussing related work, it is important to understand how a "traditional" image classification pipeline works. A common approach would be to split the process into four stages:

Preprocessing: In this stage useful regions of images could be identified, either manually or automatically. For example, for an image of a face, key regions such as the nose, eyes and mouth regions could be selected as these are likely useful discriminators between a face and non face image.

Feature Extraction: In this stage, the regions identified in the preprocessing step could now be converted to a "feature", or set of features. A region is represented as a collection of pixel values, whereas for a feature it is often useful for classification to have a single value (either continuous or discrete). To convert these regions to a single value, a mathematical operator can be applied over the region which works on the pixel values, such as taking the minimum value of the region. This step has two important properties. Firstly, the original dimensionality of the image is greatly reduced which reduces the computational time required for the classification stage. Secondly, with this new representation many redundant/irrelevant regions of the image have been removed, and only features which are representative of key regions are kept, which can improve the classification accuracy.

Transformation: The original images would be transformed to instead be represented by the newly constructed features, rather than the raw pixel values.

Classification: Now we have a new representation of the image, where instead of having raw pixel values, we have a collection of aggregated statistics over key regions of the image (features). The goal is then to use these features to predict the class of the image, using the same example as earlier this would be to predict whether or not a given image contained a face. Classification is a complex task, the key idea behind classification is to use a set of inputs/instances of which we already know the class (labelled instances), and to try and identifies patterns in these instances which can help predict the class for other instances of which we do not know what class they belong to (unseen instances).

For each of these stages, a separate system could be used. For example, a domain expert could identify key discriminative regions of an image using domain knowledge, and decide appropriate aggregations over these regions to generate the features. The images could then be passed through a transformation process which converts the raw pixels to the newly identified features. These features could then be passed into a classification algorithm, such as a Decision Tree, which could be trained on the labelled images to build a classifier in order to predict the class label for for unseen data.

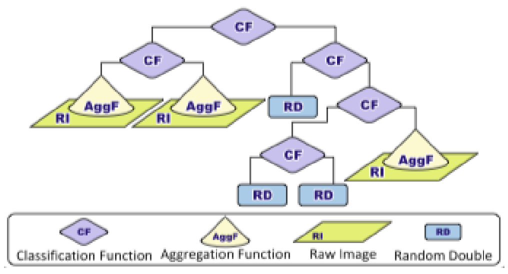
The problem with the above method, is it relies on having a domain expert available to manually extract the features. Finding domain experts can both be difficult (as with some domains there may be very few or even no experts) and their services costly. Furthermore, humans are not necessarily the best deciders of such features.

Therefore, another approach is to remove the dependence of a domain expert. Instead of manually identifying important features from the images, the process could be automated using a feature selection or feature construction algorithm. Hence, the first stage of the above method has now been automated, however, we still need to transform the images using these new features, and then classify the images based on this transformation.

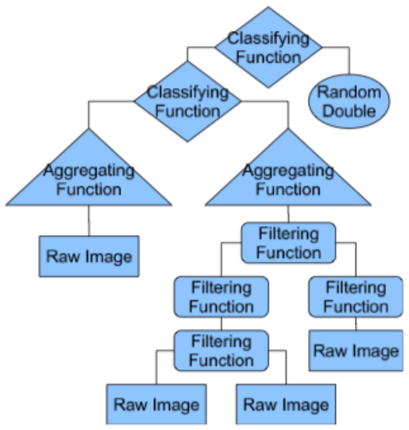
Both of the previous methods still require various disjoint components to achieve the task of classifying an image. Another solution would combine all four stages into a single system, where a raw image can be fed in, features automatically extracted based on the key regions of the image, the image transformed based on these features, and then the newly transformed image classified. GP can be utilised to achieve this goal, and this has been explored with three-tier GP (3TGP) [24], and wo-tier GP (2TGP) [25].

3TGP [24] 3TGP has three distinct tiers. The first tier (above the terminals) is the filtering tier. The filtering tier is able to do things such as add a constant to every pixel in the image, or add the pixel values of two images together. The filtering tier does not change the size of an image, rather just modifies the image pixel values in some way. These modifications were limited to simple arithmetic and statistical operators over the pixel values (such as addition, subtraction, multiplication, mean and square root). The second tier is the aggregation tier, which is only a single layer high. The aggregation tier works by taking a square "window" of the raw image pixels and producing a single double output based on some function such as min, max, mean or standard deviation of the pixel values within this window. The aggregation tier corresponds to the first three stages outlined above (preprocessing, feature extraction, transformation). The second tier is the classification tier, which can contain one or more layers. The classification functions are basic arithmetic operators: addition, subtraction, multiplication and protected division, which work on the extracted features.

2TGP [25] As the name suggests, 2TGP has a two-tiered architecture. 2TGP was based on 3TGP. The key difference is that the filtering tier was removed, and the aggregation tier was improved to work on various shapes (i.e. ellipses and rectangles) rather than being restricted to squares. 2TGP was able to outperform 3TGP, which shows the filtering tier used in 3TGP did not contribute much to the classification accuracy. 2TGP was the inspiration for this projected (specifically the architecture referred to as 2TGP-mix in [25]), and an improved "filtering" tier is proposed. The architecture of the two methods is shown in Figure 2.5 for comparison.



(a) 2TGP [25].



(b) 3TGP [24].

Figure 2.5: A compairson of existing GP architectures.

2.3.2 Convolutional Neural Networks for Image Classification

Convolutional Neural networks are perhaps the most successful image classification method. Each year ImageNet runs a competition Large Scale Visual Recognition Challenge (ILSVRC) and for the previous 6 years, the winning solution for classification has involved the use of ConvNets of some sort. 2012 had AlexNet [13], 2013 was Clarifai [26], 2014 was GoogLeNet [27] and runner-up VGGnet [28], 2015 was ResNet [11], 2016 was CUImage [29] and 2017 was BDAT [30]. The architecture of each solution is briefly explained below.

AlexNet AlexNet was a pioneering method which brought popularity back to convolutional neural networks. The network features 5 convolutional layers, 7 hidden weight layers, and 3 fully connected layers. The network featured max-pooling layers after some convolution layers, ReLU as the activation function ($max(0, x)$), and filters of various sizes (11×11

for layer one, 5×5 for layer 2, and 3×3 for subsequent layers).

Clarifai Clarifai was a refinement of AlexNet, which used visualization techniques to identify problems and improve upon AlexNet. No large architectural changes were made.

VGGNet The architecture of VGGNet was relatively simple, which featured 11-19 convolutional layers (there were multiple architectures of different depths), 5 max-pooling layers, ReLU as the activation function after each convolutional layer, and filters only of the size 3×3. The filter size was kept small, as these could be used back to back to generate similar results to larger filters (e.g. 2 filters would work as 5×5, and 3 as 7×7).

GoogLeNet (Inception) The GoogLeNet architecture is 22 layers deep, features filters of multiple sizes ($1 \times 1, 3 \times 3, 5 \times 5$), and uses both max and average pooling. One of the key developments with GoogLeNet (in contrast to previous architectures) is the convolution and pooling layers do not have to be fully stacked sequentially, some parts of the network can occur in parallel.

ResNet ResNet is an extremely deep architecture (152 layers) developed by Microsoft researchers. This paper proposed the "residual learning framework" showed which eased the training of very deep ConvNets. This architecture was significantly deeper than all previously seen convolutional neural networks, this was 8x deeper than VGGNet but had lower complexity. ResNet showed that very deep architectures can be powerful. Convolutional layers are mostly comprised of 3×3 filters.

CUImage In 2016 there was no major architectural advancements in the ImageNet contest. CUImage was largely an ensemble of existing networks.

BDAT Again, in 2017 there were no major architectural advancements. BDAT used the ResNet and Inception architecture as units in their "Residual Attention Network", and through the use of their Attention Model were able to outperform both the original networks.

In 2015 (with ResNet) the accuracy surpassed that of human professionals (5.1% error rate), so the current state-of-the-art ConvNets are clearly very powerful for image classification.

2.3.3 Combining Evolution and Deep Learning for Image Classification

Evolutionary computation and deep learning are both very hot topics in artificial intelligence, as such hybrid methods which combine the two are a promising area of research. Using GP to automatically evolve convolutional neural network archictectures was examined in [12], and using GP to represent and evolve standard neural networks was examined in [31]. This family of algorithms are commonly referred to as NeuroEvolution, with the most famous examples being NEAT (NeuroEvolution of Augmenting Topologies) [32] and HyperNEAT [33]. The main limitations of such approaches are the computational time/computational resources required for such algorithms. [34] extends the NEAT algorithm to be optimised for convolutional neural networks, and uses volunteer computing to offset the problem of limited resources.

Another recent example is shown in [35], where evolutionary strategies are used as a replacement for sampling and backwards propagation in neural networks for reinforcement learning.

There does not appear to be much work which utilise the convolution and pooling functions as nodes in genetic programming, or methods which utilise evolutionary principles to evolve filters for image classification.

2.3.4 GP for Feature Construction

Feature construction is the process of creating new features from existing features [14], often with the goal of dimensionality reduction. Feature construction can also help to uncover interaction between features [36]. Feature construction is particularly useful as a preprocessing step for classification problems with a large number of features [37], for example, when using the raw pixels of an image as features. Machine learning techniques, specifically evolutionary computation, have been increasingly used for feature construction ([38], [39]).

Recently genetic programming has been used for feature construction with success, the reason being the flexible tree structure is able to automatically combine the original features to create new higher level features. Furthermore, GP performs implicit feature selection as well, so the constructed features are often combinations of features which themselves are high-performing. The features constructed from genetic programming in [38] were shown to "significantly reduce the dimensionality and maintain or even increase the accuracy" for classification tasks, and genetic programming techniques for feature construction on images was shown to outperform manually constructed features for determining aesthetic value in [40] .

2.4 Chapter Summary

An overview of the current state of image classification and feature construction was given. The focus was placed on genetic programming from evolutionary computation, and convolutional neural networks from deep learning. Hybrid methods which combine elements from both were also presented, as this is a promising area of current research.

Chapter 3

New Method

A new GP method was developed that utilises key ideas from convolutional neural networks to improve the performance of GP methods for image classification. This chapter discusses the main components of the newly proposed method, and results on four commonly used image classification datasets are presented.

3.1 Introduction

As discussed in the background section, a common approach to image classification is to split the process into four seperate stages: Preprocessing, Feature Extraction, Transformation and Classification.

Various levels of human intervention are often required in one (or more) of these stages, however, 2TGP and 3TGP removed this dependence on human intervention and allowed a fully automatic solution. Both these methods suffer from the same key problem, the feature extraction phase works on the raw pixels, which can limit the extraction ability.

Convolutional neural networks help overcome this limitation, and are an approach that has had much success in recent years. This is due to the networks ability to build up higher level representations (i.e. object outlines) from lower level representations of the image (i.e. edges) with stacked layers of convolutions. However, the architecture of such networks (the number of convolutions to apply in each layer, and the number of hidden layers) is often complex and must be manually defined.

In this section, a new method is proposed, referred to as convolutional genetic programming (ConvGP), which aims to overcome the limitations mentioned. The various stages of the pipeline are combined into a single component (with no human intervention required), while also offering the ability to extract features from higher level representations of the original image (rather than the raw pixels), without the need for a manually crafted architecture.

3.2 Chapter Goals

The aim is to develop and evaluate a method which overcomes some limitations of existing approaches, namely one which automates and combines the various stages of the image classification pipeline into a single system, and develops an architecture which is evolved automatically rather than specified manually as with convolutional neural networks.

3.3 Program Structure/Representation

To be able to combine the various stages of image classification (preprocessing, feature extraction, transformation and classification) into a single tree, strongly-typed GP [22] was used to enforce a tiered structure. The stages are roughly broken into tiers, these tiers are then stacked in such an order that the original image classification pipeline is simulated. An example of the structure is shown in Figure 3.1

The first (bottom) tier is flexible and can feature multiple convolution layers, this tier is where higher level representations of an image can be developed using lower level representations. Pooling functions also appear optionally throughout the convolution tier (pooling does not have to appear after a convolution has been applied) as a way of reducing the image's dimensions and thus improving the processing speed.

Next is the aggregation tier, this tier identifies key regions of the image, and also appropriate statistical functions to apply over the regions to produce a feature representing the region. This tier is compulsory, and is fixed to be a single layer high.

The final (top) tier is the classification tier, the output of this tier is a single numerical value, and based on a threshold this value is used to classify an instance as either positive or negative. This tier is compulsory, however, this is not fixed to a single layer and can be any arbitrary height (constrained only by the maximum tree depth).

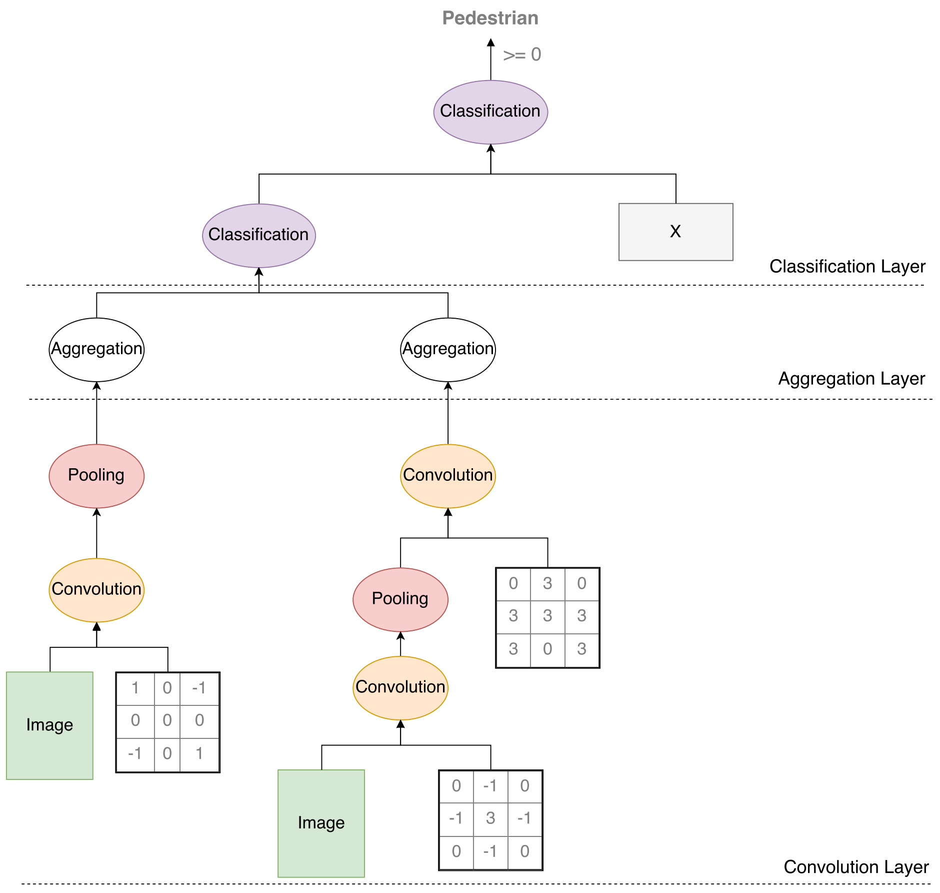


Figure 3.1: Example tree displaying the tiered structure.

3.3.1 Function set

There are two functions which occur in the convolution tier. The first is convolution, which applies a filter over an image. The convolution function is the core function used in convolutional neural networks, so this has been shown repeatedly to be useful for the task of image classification. However, convolution is a costly function, as the convolved value must be computed for every pixel in an image. To offset some of this cost, pooling was also introduced. Pooling reduces the dimensionality of an image, which speeds up the convolution process. Pooling is used in all top performing convolutional neural network architectures.

An optional parameter can be set inside the convolution function, which is whether or not to apply an activation function. If the parameter is set to `true`, the activation function applied is $max(0, x)$ (ReLU), if it is set to `false` no activation function is applied. All results in this report are with the ReLU function applied. This was also trialled as a node in the tree however better results were achieved applying the activation function after every convolution.

The aggregation tier applies a statistical function (Mean, Standard Deviation, Minimum or Maximum) over a window of the image, and returns the output of this function. The window shape can be one of four possible shapes: Ellipse, Row, Column or Rectangle. The performance of the function depends on the size of the window, where larger window sizes will be more costly as all the pixels contained within the window must be iterated. The chosen functions are all useful for feature construction, the mean can help to summarise a region, the standard deviation can show the variation within a region, and the minimum and maximum can be particularly useful for highlighting differences between two classes.

The classification tier uses the basic arithmetic operators (addition, subtraction, multiplication and division). Protected division was used to prevent any errors when dividing by zero. Classification functions can be stacked to produce additional operators, for example, the square function could be represented by (Mul X X), and the cube function as (Mul X (Mul X X)). All classification functions are very fast to compute as they are just performing basic arithmetic operations on the input values.

Table 3.1 gives an overview of each function used.

Table 3.1: Function set.

Function	Input	Output	Function
Convolve	image, filter	image	This applies the filter over the image
Pool	image	image	Applies 2x2 max pooling, so a 16x16 grid becomes an 8x8 grid with the largest pixel values.
AggMean	image, x_cord, y_cord, size, shape	double	Returns the mean pixel value of the aggregation window
AggStdev			Returns the standard deviation of the pixel values in the aggregation window
AggMin			Returns the minimum pixel value of the aggregation window
AggMax			Returns the maximum pixel value of the aggregation window
Add	double, double	double	Adds the two children together
Sub			Subtracts the second child from the first
Mul			Multiplies the two children together
Div			Divides the first child by the second child, if the second child is zero, zero is returned (protected division).

3.3.2 Terminal set

Table 3.2 shows the terminal set. The main base terminal is the raw image being used, the image is represented as a 2D array of grey values between 0 and 255.

There is only one convolution specific terminal, this is the filter to apply to the image. Each value in the filter is randomly selected integer between -3 and $+3$, as used in [41].

There are several terminals for the aggregation tier, which are used to determine the position, size, and shape of the window. The position (x_Cord and y_Cord) is specified as a percentage of the images size, raw pixels were not used for positioning as pooling can change the size of an image. The shapes used were shown in [25] to offer the highest classification accuracy (with the 2TGP-mix method).

The classification tier can optionally use random values (constants) between -5 and $+5$, this can give some additional power instead of just using the result of aggregation window on the image as shown in [25] . Larger constants could be created by combining such terminals, i.e., (Add 5 5) to generate 10, and fractions could also be represented, i.e., (Div 1 2) to generate 0.5.

Table 3.2: Terminal Set

Terminal	Value Type	Value Range	Purpose
Image	Integer[][]	$\{0, 1, \dots 255\}$	Corresponds to the raw input image
Filter	Integer[3][3]	$\{-3, -2, \dots, +3\}$	Filter to apply over the image
X_Cord	Double	$[0.05, 0.9]$	The starting (left-hand side) x coordinate of the window for the aggregation function. Specified as a percentage of the width, so 0.05 = 5%.
Y_Cord			The starting y (top) coordinate of the window for the aggregation function. Specified as a percentage of the height, so 0.05 = 5%.
Size	Double	$[0.15, 0.75]$	Used as the width and the height for the aggregation function windows
Shape	String	"Row" "Column" "Rectangle" "Ellipse"	Input to the aggregation functions, determines the shape window to use.
Constant	Integer	$\{-5, -4, \dots, +5\}$	Used as input to the classification layer

3.3.3 Fitness Function

The fitness function used was simply the proportion of correctly classified instances (i.e. the classification accuracy): $(TP + TN)/Total$. TP is the number of correctly classified positive instances (e.g. correctly classified "face" instances), and TN is the number of correctly classified negative instances (e.g. correctly classified "nonface" instances). $Total$ is the total number of instances. This assumes an even number of positive and negative examples (balanced), which is the case for all datasets used in this study. For unbalanced datasets, a different fitness function would need to be used (such as the F-measure) as the function above would be biased towards individuals which did well on the class with more examples. The classification accuracy is used for assigning a "goodness" to individuals in the evolutionary stage, and also as a measure of classification accuracy in the testing stage.

3.3.4 Overall Algorithm

The overall evolutionary process followed is given in Figure 3.2. In the initial stage, trees are randomly constructed. Throughout the evolutionary process, trees are evolved using the crossover and mutation operators. Reproduction (elitism) is used to ensure the best individuals of a generation never get worse. After 50 generations have been reached, the evolution is considered complete and the highest performing individual (tree) is returned.

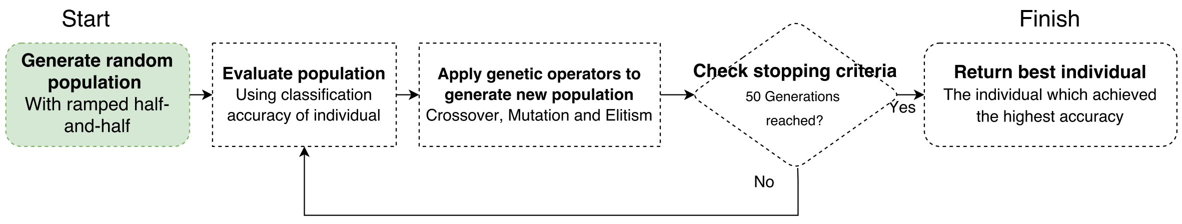

Start
Generate random population
With ramped half-and-half
Evaluate population
Using classification accuracy of individual
Apply genetic operators to generate new population
Crossover, Mutation and Elitism
Check stopping criteria
50 Generations reached?
Yes
No
Finish
Return best individual
The individual which achieved the highest accuracy

Figure 3.2: Evolutionary process followed.

3.4 Experiment Design

3.4.1 Parameter Settings

The parameter values are important for genetic programming, a brief overview of the parameters is given in Table 3.3.

Table 3.3: Parameter settings used for the evolutionary process

Parameter	Value	Justification
Population	1024	Increase chance of generating a good individual in early generations
Generations	50	Gives adequate time to evolve good solutions
Max Tree Depth	10	Give individuals room to grow, while not being excessively large
Tournament Size	7	Too small of a tournament can result in selecting poor individuals
Crossover Rate	0.8	Majority of individuals should be crafted from crossover
Mutation	0.2	Maintain diversity in the population
Elitism	0.01	Ensure best individuals never get worse

3.4.2 Datasets

All datasets were grayscale images. To ensure a fair trial, the data was split 50:50 into training and test sets, and no validation set was used for any method.

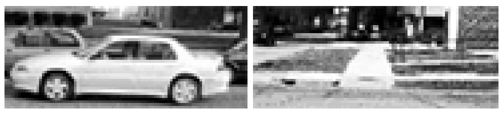


Figure 3.3: Sample Car Images.

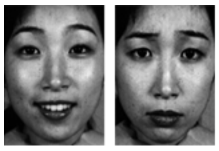


Figure 3.4: Sample JAFFE Images.

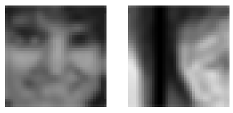


Figure 3.5: Sample Face Images.

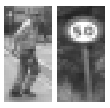


Figure 3.6: Sample Pedestrian Images.

Cars

The car dataset [42] has 1,000 images, with an equal distribution of car and non-car images. All images are of the size 100x40 pixels. All car images were side on, non-car images were a variety of background scenes. This dataset is simpler than the others (as side on cars are relatively distinctive), however also has fewer instances to learn from. Figure 3.3 shows an example of one car and non-car image.

JAFFE

The JAFFE dataset [43] is a collection of Japanese female facial expression images. There are seven sets of facial expressions in the original dataset, however for this project only the "Happy" and "Sad" facial expressions were used as the focus is only on binary classification. "Happy" and "Sad" were chosen as the resulting models will likely give interesting insight to regions automatically deemed useful for distinguishing between the two emotions. Each set of facial expressions contains a total of 30 images, from 10 different models (3 images per model), so in this case there are 60 images total. The images are all of the size 130x180 pixels. Figure 3.4 shows an example of one happy and sad image.

Face

The face dataset was a subset of the CBCL Face Database images [44]. This dataset contains 6,000 images total, with an equal distribution of face and non-face images. Face images were all full frontal views containing the entire face, and non-face images are a mixture of background photos. All images in this dataset are of the size 19x19 pixels. Figure 3.5 shows an example of one face and non-face image.

Pedestrian

The pedestrian dataset [45] is the most complex of the trialled datasets, the pedestrian images were extracted from videos. Pedestrians are all standing upright and fully visible, but there were no other constraints on the pedestrians stance or clothing. There are 9,600 images, again with an even distribution of pedestrian and non-pedestrian images. All images were of the size 18x36 pixels. Figure 3.6 shows an example of one pedestrian and non-pedestrian image.

3.4.3 Baseline Methods

All methods used the exact same training:testing splits for a fair comparison. The results were averaged over three shuffles of training:testing splits, to ensure the results were not dependant only on the split used. It was computationally prohibitive to run over a larger number of splits (as some processes took days to run), however, doing so could yield more accurate insights.

General Classifiers

Image classification can be treated as a typical classification problem by representing each pixel as a feature. For example, consider the sample image below containing 361 pixels (red lines dividing the pixels), this could be represented as 361 features where each feature corresponds to the intensity of the pixel.

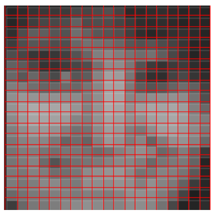


Figure 3.7: Sample Features

We could then use this input to train and test a classifier. That is exactly what has been done in the Results and Discussion section, using the powerful open source Weka classifiers [46] as a point of comparison.

Classifiers were chosen from a range of paradigms, to ideally give a broad range of results. The general classifiers used were: Adaboost, Decision Trees, Nearest Neighbour, Naive Bayes, and Support Vector Machines. All of these classifiers are commonly used, and often provide high accuracy for classification tasks.

GP Classifiers

A full evolutionary process (of 50 generations) was repeated 30 times for each split of the data, and the training/testing results computed for each run. This gave a large enough

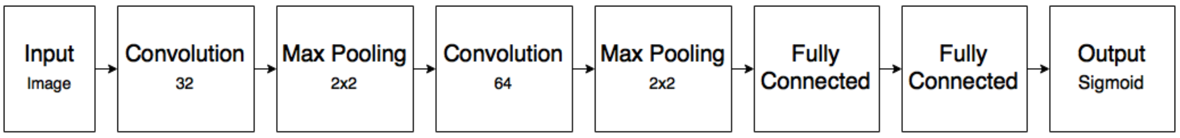


Figure 3.8: ConvNet architecture used for comparison.

sample to ensure the results were not purely dependent on the seeds used.

Convolutional neural networks

As previously discussed, convolutional neural networks are the current state-of-the-art approach to Image classification. The various architectures outlined in the background section offer high accuracy for larger images (256x256 is common for ImageNet), but require extensive training with a large number of examples to achieve such results. Such architectures are therefor not suitable for the datasets being used here (due to the comparatively small number of of images and sizes of such images), so a manually crafted architecture was used for comparison. The architecture is roughly inspired by the original LeNet architecture, featuring two convolutional layers and two 2x2 max pooling layers. ReLU was used as the activation function for all convolutions. Sigmoid was used as the output, as we are dealing with binary classification.The architecture is given in Figure 3.8.

Stochastic Gradient Descent was used with a learning rate of 0.01. A stopping criteria of 100 epochs was used, and weights were initialized randomly. To account for the random initialization of weights, each network was trained 30 times and the results averaged (as done with the GP methods) to ensure the results were not dependant on the random seed used. Keras [47] was used for implementation.

3.5 Results and Discussions

The proposed solution (ConvGP) was tested on the four datasets with a range of difficulties. Datasets from different domains were used to ensure the developed solution is domain-independent. Statistics are reported on the training and testing accuracies, and the training and testing times. An unpaired Welch *t*-test was used to compare the average testing accuracy between the proposed solution, and each of the other classifiers on each dataset to verify if the differences in means were statistically significant or down to chance. Welch *t*-tests were used, as equal variance can not be assumed between classifiers. A significance level of 0.05 was used. If a p-value was below 0.0001, 0.0001 was used instead for readability purposes.

Each of the following sections has two tables and a bar chart. The first table gives summary statistics for all the classification methods, including the mean, standard deviation and maximum for accuracy and computational time on both the training and testing data. The second table gives the results of the Welch *t*-test outlined above, testing if the average testing accuracy for a given classifier was significantly different to the proposed solution. The bar chart visualises the testing accuracy, showing the average testing accuracy for a classifier in blue, and the maximum testing accuracy achieved by that classifier in red. The chart is sorted in descending order of maximum testing accuracy.

3.5.1 JAFFE

Table 3.4: Summary statistics for the JAFFE dataset

JAFFE Dataset	Training Accuracy		Testing Accuracy		Training Time		Testing Time	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
Adaboost	100 ± 0	100.00	84.44 ± 2.96	93.33	0.36 ± 0.20	1.06	0 ± 0	0.02
Decision Tree	100 ± 0	100.00	82.22 ± 2.43	90.00	0.07 ± 0.04	0.29	0 ± 0	0.00
1NN	100 ± 0	100.00	68.89 ± 5.51	86.67	0.07 ± 0.02	0.17	0.08 ± 0.01	0.20
Naive Bayes	87.78 ± 1.05	90.00	61.11 ± 2.75	70.00	0.32 ± 0.06	0.48	0.22 ± 0.04	0.34
SVM	100.00 ± 0.00	100.00	91.11 ± 2.75	100.00	0.06 ± 0.04	0.26	0.01 ± 0.00	0.02
ConvNet	89.93 ± 13.72	100.00	80.74 ± 13.37	100.00	322.84 ± 1079.72	7915.27	0.39 ± 0.04	0.63
2TGP	96.93 ± 3.17	100.00	75.22 ± 11.71	96.67	135.62 ± 63.66	294.90	0.01 ± 0.02	0.14
ConvGP	96.19 ± 3.55	100.00	73.52 ± 10.54	96.67	354.09 ± 238.60	1090.92	0.03 ± 0.02	0.12
ConvGP (No pooling)	96.67 ± 3.31	100.00	75.19 ± 9.62	90.00	501.27 ± 442.28	2616.66	0.04 ± 0.04	0.17

When comparing the maximum testing accuracy across the runs, the proposed solution (with pooling) performed on par with the existing GP approach, and outperformed all general classifiers except for SVMs. Interestingly, the SVM was able to correctly classify every instance in one run, and offered a very high average accuracy of 91.11%. The ConvNet was able to achieve 100% accuracy in one case, however on average still underperformed SVMs.

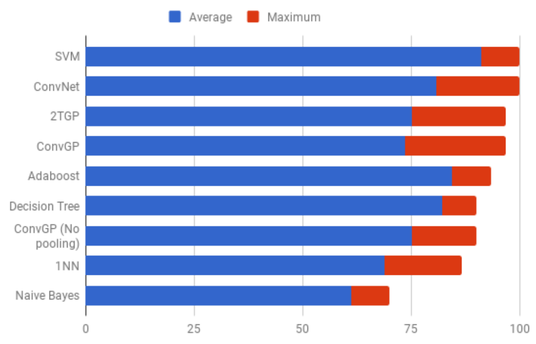


Figure 3.9: A comparison of the testing accuracy for various classifiers on the JAFFE dataset.

With this task, the proposed solution saw improvement over the "elementary" classifiers (Naive Bayes, Nearest Neighbour etc.), however, was not able to outperform SVMs. This shows for the JAFFE classification task, there are likely key discriminatory pixels in the original image which were detected by SVMs, and convolutions were not a requirement for achieving high testing accuracy (as SVMs also outperformed the ConvNet on average).

Table 3.5: Unpaired Welch *t*-test against mean testing accuracy for ConvGP.

	T-Value	P-Value
Adaboost	5.3573	0.0059
Decision Tree	4.8614	0.0046
1NN	1.374	0.3032
Naive Bayes	6.4041	0.0031
SVM	9.0772	0.0008
ConvNet	4.0232	0.0001
2TGP	1.0237	0.3074
ConvGP (No pooling)	1.1102	0.2684

The proposed method with pooling included outperformed the method without pooling, achieving a higher mean and maximum testing accuracy. However, the results were not statistically significant as shown in Table 3.5. This shows pooling is useful for the JAFFE dataset, drastically decreasing the training time while maintaining comparable accuracy.

In terms of the training time, the methods which used convolutions were by far the slowest, as the convolutional operator is costly to apply. Pooling helped to reduce this cost, as shown by the difference in average training time between the two ConvGP methods with and without pooling.

3.5.2 Cars

Table 3.6: Summary statistics for the Cars dataset

Cars Dataset	Training Accuracy		Testing Accuracy		Training Time		Testing Time	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
Adaboost	95.27 ± 1.95	97.2	88.27 ± 1.09	89.4	2.285 ± 0.117	2.659	0.003 ± 0.001	0.007
Decision Tree	99.53 ± 0.09	99.6	85.27 ± 3.08	89.6	0.883 ± 0.085	1.171	0.007 ± 0.028	0.188
1NN	100.00 ± 0.00	100	94.00 ± 1.40	95.6	3.284 ± 0.117	3.65	5.646 ± 0.481	6.46
Naive Bayes	93.00 ± 0.43	93.6	92.20 ± 0.71	92.8	0.791 ± 0.052	0.941	0.569 ± 0.016	0.662
SVM	100.00 ± 0.00	100	95.33 ± 0.90	96.6	0.255 ± 0.031	0.39	0.016 ± 0.002	0.023
ConvNet	100 ± 0	100	97.88 ± 0.56	99.4	302.3 ± 12.4	332.12	1.07 ± 0.11	1.77
2TGP	93.71 ± 3.04	97.6	89.45 ± 3.98	97.4	557.720 ± 137.451	1038.68	0.030 ± 0.014	0.079
ConvGP	94.26 ± 2.11	97.6	90.29 ± 2.84	95.8	1957.337 ± 1369.228	7096.995	0.187 ± 0.186	0.765
ConvGP (No pooling)	94.01 ± 2.30	97.6	90.19 ± 2.83	95.6	2521.239 ± 1469.789	7244.718	0.240 ± 0.205	0.999

For the Cars dataset, the proposed solution achieved similar maximum testing accuracy as SVMs (within 1%), however the mean testing accuracy of the proposed solution was significantly lower (shown in Table 3.7). The nearest neighbour classifier performed well on this dataset, likely because cars tend to have similar shapes and sizes, and with this dataset the cars are in similar positions, so a method based on distance between images such as nearest neighbour would likely work well. The proposed solution achieved comparable results (no statistical significant difference) to Nearest Neighbour in terms of average testing accuracy.

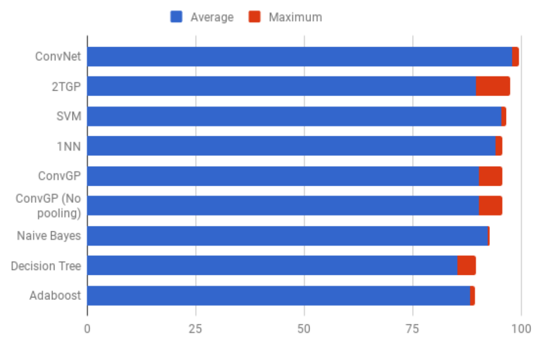


Figure 3.10: A comparison of the testing accuracy for various classifiers on the Faces dataset.

Table 3.7: Unpaired Welch *t*-test against mean testing accuracy for ConvGP.

	T-Value	P-Value
Adaboost	2.8986	0.0626
Decision Tree	2.7838	0.1084
1NN	4.3042	0.05
Naive Bayes	3.7629	0.0197
SVM	8.4045	0.0035
ConvNet		
2TGP	1.6298	0.1051
ConvGP (No pooling)	0.2366	0.8132

There was no statistical significance between 2TGP and the proposed solution, or the version without pooling and the version with pooling. Again, since the version which utilises pooling achieves faster training time, this should be the preferred method.

The convolutional neural network achieved the highest average and maximum testing accuracy for the cars dataset, however, also had a comparatively high testing time ($6\times$ greater than the proposed solution).

3.5.3 Faces

Table 3.8: Summary statistics for the Face dataset

Faces Dataset	Training Accuracy		Testing Accuracy		Training Time		Testing Time	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
Adaboost	91.51 ± 0.48	92.17	90.80 ± 1.37	91.87	1.02 ± 0.12	1.571	0.00 ± 0.01	0.058
Decision Tree	99.72 ± 0.11	99.8	96.57 ± 0.38	96.97	0.89 ± 0.12	1.337	0.01 ± 0.02	0.118
1NN	100.00 ± 0.00	100	96.52 ± 0.31	96.93	6.70 ± 0.38	7.545	8.56 ± 0.68	9.881
Naive Bayes	91.62 ± 0.70	92.6	92.19 ± 0.45	92.67	0.45 ± 0.06	0.655	0.33 ± 0.02	0.427
SVM	99.38 ± 0.18	99.63	97.48 ± 0.31	97.83	0.96 ± 0.15	1.312	0.01 ± 0.00	0.015
ConvNet	99.81 ± 0.2	99.97	99.29 ± 0.24	99.63	113.22 ± 11.64	148.91	0.43 ± 0.07	0.69
2TGP	96.26 ± 2.35	98.8	96.04 ± 2.38	98.6	424.644 ± 108.2	822.192	0.03 ± 0.02	0.186
ConvGP	95.29 ± 2.81	99.23	94.93 ± 2.68	98.47	963.11 ± 598.58	2646.282	0.10 ± 0.09	0.389
ConvGP (No pooling)	96.53 ± 1.73	98.73	96.09 ± 1.74	98.33	7644.95 ± 10997.02	37671.18	0.08 ± 0.11	0.503

All methods performed relatively well on the Faces dataset, achieving over 90% accuracy on average. The genetic classifiers (2TGP, and both the proposed methods) outperformed all the general classifiers in terms of maximum testing accuracy. The ConvNet achieved higher average testing accuracy than the proposed solution, however, the testing time was also much slower (4× slower on average).

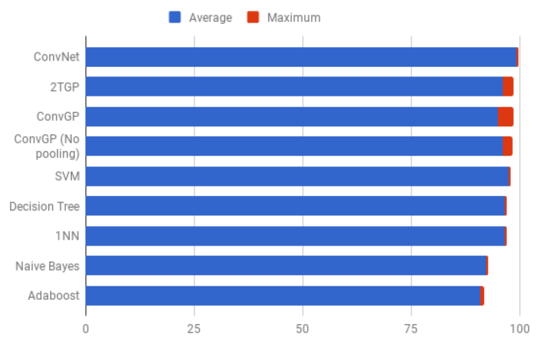


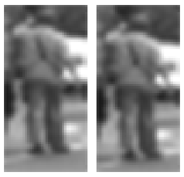
Figure 3.11: A comparison of the testing accuracy for various classifiers on the Faces dataset.

The training time of the genetic classifiers was drastically greater than the general classifiers. There are many images in the Faces dataset (3,000 total for training), which makes it costly to evaluate all the individuals in the population for the GP methods. In terms of testing time, however, the GP methods are comparable to most general classifiers, and were significantly faster than Nearest Neighbour, Naive Bayes and the ConvNet.

Table 3.9: Unpaired Welch *t*-test against mean testing accuracy for ConvGP.

	T-Value	P-Value
Adaboost	4.9172	0.039
Decision Tree	4.5851	0.0005
1NN	4.7545	0.0001
Naive Bayes	7.1391	0.0001
SVM	7.6251	0.0001
ConvNet	15.3723	0.0001
2TGP	2.938	0.0037
ConvGP (No pooling)	3.444	0.0007

With this task, the proposed method without pooling achieved a higher average testing accuracy than the method with pooling (statistically significant as shown in Table 3.9), however, the method with pooling achieved a higher maximum accuracy and offered drastically faster training times. Images from this dataset were 19 × 19 pixels, so pooling perhaps often loses too much of the original details with such small images i.e. if two pooling operators were stacked, the resulting image would only be 4 × 4 pixels which is unlikely to capture the details of a face adequately.


(a) Pedestrian Class

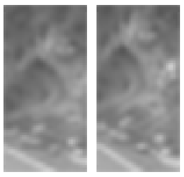

(b) Background Class

Figure 3.13: The nearest neighbour for an example from each class.

3.5.4 Pedestrian

Table 3.10: Summary statistics for the Pedestrian dataset

Pedestrian Dataset	Training Accuracy		Testing Accuracy		Training Time		Testing Time	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
Adaboost	79.26 ± 1.81	80.96	77.72 ± 1.57	79.21	3.108 ± 0.496	4.355	0.006 ± 0.001	0.009
Decision Tree	99.28 ± 0.08	99.38	89.24 ± 0.31	89.67	4.586 ± 0.636	5.962	0.007 ± 0.001	0.011
1NN	100.00 ± 0.00	100	98.08 ± 0.24	98.4	27.583 ± 3.088	37.516	41.437 ± 4.483	55.906
Naive Bayes	77.21 ± 0.24	77.38	76.53 ± 0.35	77.02	1.144 ± 0.056	1.438	0.872 ± 0.012	0.926
SVM	90.65 ± 0.12	90.79	84.48 ± 0.41	85	84.683 ± 10.504	99.744	0.029 ± 0.006	0.049
ConvNet	99.96 ± 0.04	100	97.6 ± 0.19	98.17	315.77 ± 10.75	349.88	1.27 ± 0.11	1.51
2TGP	82.53 ± 2.35	86.33	81.95 ± 2.44	85.42	1984.628 ± 519.344	3545.926	0.160 ± 0.117	0.593
ConvGP	81.27 ± 2.95	86.96	80.59 ± 2.96	86	3285.113 ± 2179.032	11554.993	0.313 ± 0.291	1.493
ConvGP (No pooling)	81.59 ± 2.76	86.54	81.04 ± 2.71	85.9	4636.360 ± 2802.698	14242.428	0.395 ± 0.336	1.707

What is very surprising for this dataset is the testing accuracy of the nearest neighbour method. Nearest neighbour outperformed all other methods for this task, which likely means there are photos which are very similar. To verify this, the nearest neighbour was run for a particular image from each class, and the results given in Figure 3.13. It is clear there are in fact very similar images in this dataset, which is why nearest neighbour performs so well. However, this does not mean the method will generalise well to other pedestrian photographs, i.e. if new pedestrian images were sourced that did not have a near match in the training set, this method would not likely do so well.

The proposed solution (both with and without pooling), slightly outperformed 2TGP, SVMs, Adaboost and Naive Bayes in terms of maximum testing accuracy. When comparing average testing accuracy, the proposed solution outperformed Adaboost and Naive Bayes, however, only Naive Bayes was statistically significant (as shown in Table 3.11).

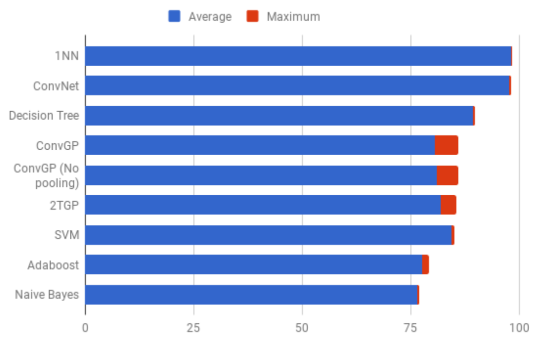


Figure 3.12: A comparison of the testing accuracy for various classifiers on the Pedestrian dataset.

Table 3.11: Unpaired Welch t-test against mean testing accuracy for ConvGP.

	T-Value	P-Value
Adaboost	2.9938	0.0958
Decision Tree	24.0478	0.0001
1NN	51.2309	0.0001
Naive Bayes	10.9219	0.0001
SVM	9.9325	0.0001
ConvNet	54.4053	0.0001
2TGP	3.3634	0.001
ConvGP (No pooling)	1.0638	0.2889

3.6 Chapter Summary

In terms of maximum testing accuracy achieved, the proposed method often outperformed the basic classification methods, however, was unable to beat the convolutional neural networks. However, the proposed solution achieved faster testing time than the convolutional neural network. In terms of average testing accuracy, the proposed method was comparable to the general classifiers, however, struggled to provide state-of-the-art performance on any of the four datasets trialled. The proposed method with pooling included was shown to outperform the method without pooling in general, so this should be the preferred architecture, and was the architecture used for the remaining sections.

A clear advantage of the proposed method is the interpretability of the model, which will be investigated in Chapter 4.

While the proposed architecture was comparable when used as a classification algorithm, this architecture can also serve as a feature construction method, which is examined in Chapter 5.

Chapter 4

Visualisation

4.1 Introduction

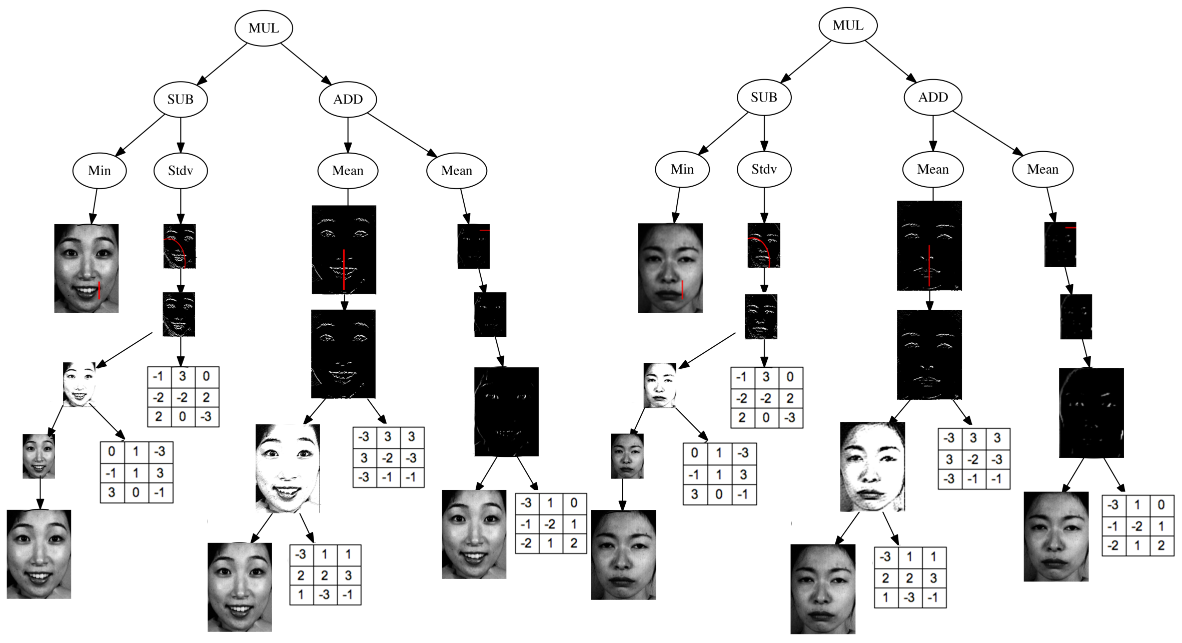
One important property of the proposed approach over other state-of-the-art (convolutional neural networks) image classification methods, is the interpretability of the model.

Visualising an evolved program gives an idea of what regions have been deemed useful, and what filters have been learnt for a particular task. For each dataset, one tree has been visualised, and an example is given for both the positive and negative classes.

Each of the programs (trees) in this chapter represents a well performing program (a program which achieved high testing accuracy). The programs were chosen to show how a good solution performs, to find similarities amongst these programs, and to analyse characteristics of well performing individuals. In some cases there were trees which achieved higher testing accuracies than those shown in this chapter, however, they were also very large so have been excluded for readability purposes.

4.2 Chapter Goals

The aim of this chapter is to first develop a way of visualising the evolved models, which can show a model works (i.e. interpret the learnt filters and regions). Next is to investigate (using these visualisations) whether the proposed method is able to effectively detect key regions of the images, and learn useful filters for the image.



(a) Happy Class

(b) Sad Class

Figure 4.1: A good performing individual on the JAFFE dataset. This individual achieved 96.67% testing accuracy. An example from each of the two classes is given for comparison.

4.3 JAFFE

The JAFFE task is to distinguish between images of happy and sad faces. There are many tell tale signs which we look out for when interpreting emotions as humans such as the eyes, the positioning of the mouth, and the overall demeanor of a person. Visualising the constructed trees for this dataset can give an interesting insight to features which have been automatically deemed relevant by the computer for this problem.

The program shown in Figure 4.1 achieved 96.67% testing accuracy. An example for each class is given. For all the visualisations, aggregation windows are represented by a red shapes over the image being aggregated. Pooling is represented by reducing the images size. In some cases the image size is only reduced by 25% for visualisation purposes, however in the real program as 2×2 pooling is applied the images will halve in size after each pooling operator. Filters are represented by a 3×3 matrix. If an image was fed directly into an aggregation node (i.e. no convolution was applied), the raw image will not be shown as a terminal for clarity.

The tree has a maximum depth of seven. For the convolutional tier, the maximum number of layers was 3. Stacked convolutions appear very useful for the JAFFE dataset, and pooling was also used throughout the tree. The images from this dataset are 130x180 pixels, so due to the large size pooling is particularly effective as minimal details from the original image are lost.

From Figure 4.1 it is clear many useful filters were learned, where the majority of these filters served as a form of edge detection as shown in Figure 4.2. Edge detection looks for large changes in pixel values in either the vertical or horizontal direction. Looking at the outcome of the convolutions, it is very clear whether or not a smile is present (due to the large presence of white).

The four identified regions are shown in Figure 4.3, this shows that the proposed GP system can automatically detect key regions without human intervention. 3 of these 4 re-

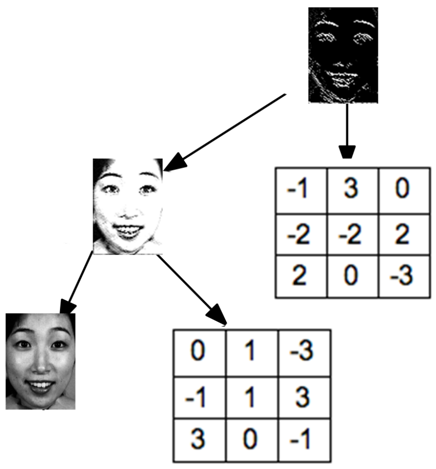


Figure 4.2: An example of two stacked convolutions from Figure 4.1. The bottom convolution applies a high contrast filter, then the top convolution applies edge detection on top of this. The example shows an instance from the happy class.

gions have had convolutions applied. The majority of these regions focus on the mouth area. This matches intuition as the mouth tends to be a tell-tale difference between determining if somebody is happy or sad. For example, a large amount of white pixels around the mouth region likely indicates the subject is smiling. Region d. focuses on edge detection around the eyebrow region, taking the mean value slightly above the resting position of the eyebrow. When people are sad, there eyebrows tend to be lowered and thus the mean of this region will be low. When people are happy the eyebrows are often raised, and thus, the mean will be higher due to the presence of an edge in this region of the image.

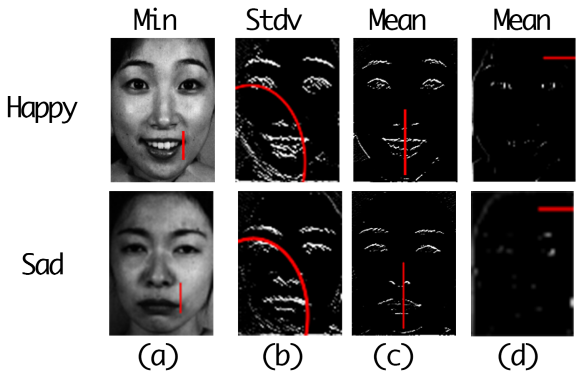


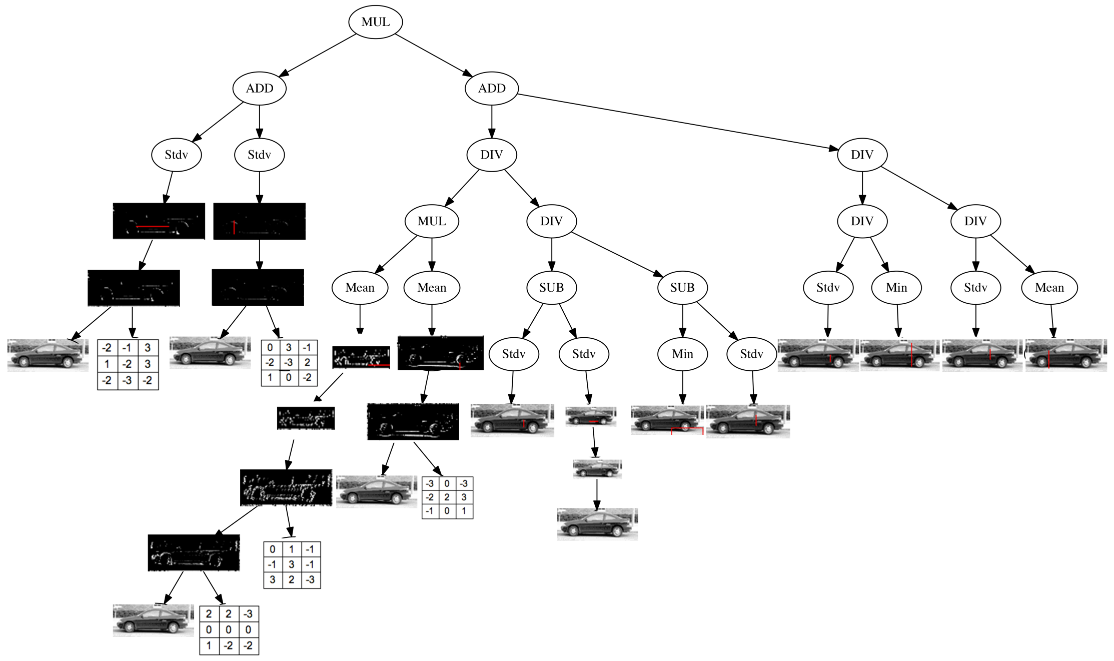
Figure 4.3: A comparison of the four aggregation regions used from the tree in Figure 4.1.

The visualisation indicates that the evolved individual effectively learned useful filters for the JAFFE dataset, and extracted discriminative regions from the image. A high testing accuracy of 96.67% was able to be achieved with these learnt features and regions.

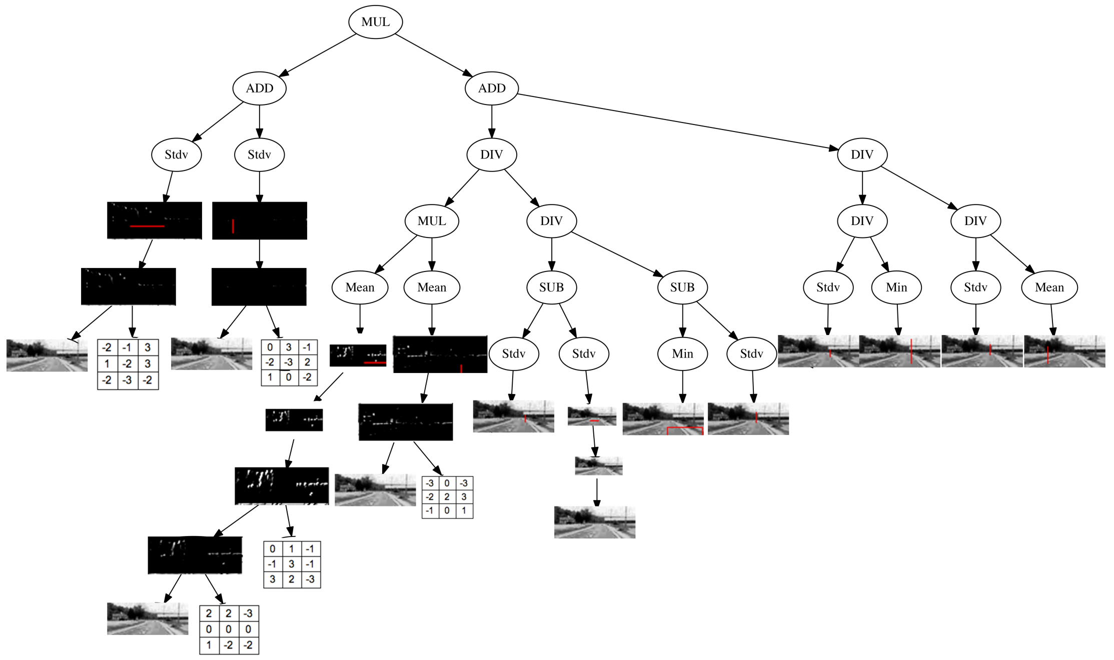
4.4 Cars

The Cars task is to distinguish between images of cars and backgrounds. The tree in Figure 4.4 achieved 92.53% testing accuracy.

The tree in Figure 4.4 is relatively wide and has a maximum depth of 9. There are many convolution and pooling nodes within the tree. Each of the filters in the image work very



(a) Car



(b) Background

Figure 4.4: A good performing individual on the Car dataset. This individual achieved 92.53% testing accuracy. An example from each of the two classes is given for comparison.

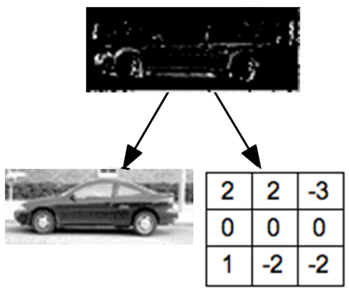


Figure 4.5: An evolved filter for the Cars dataset which performs vertical edge detection.

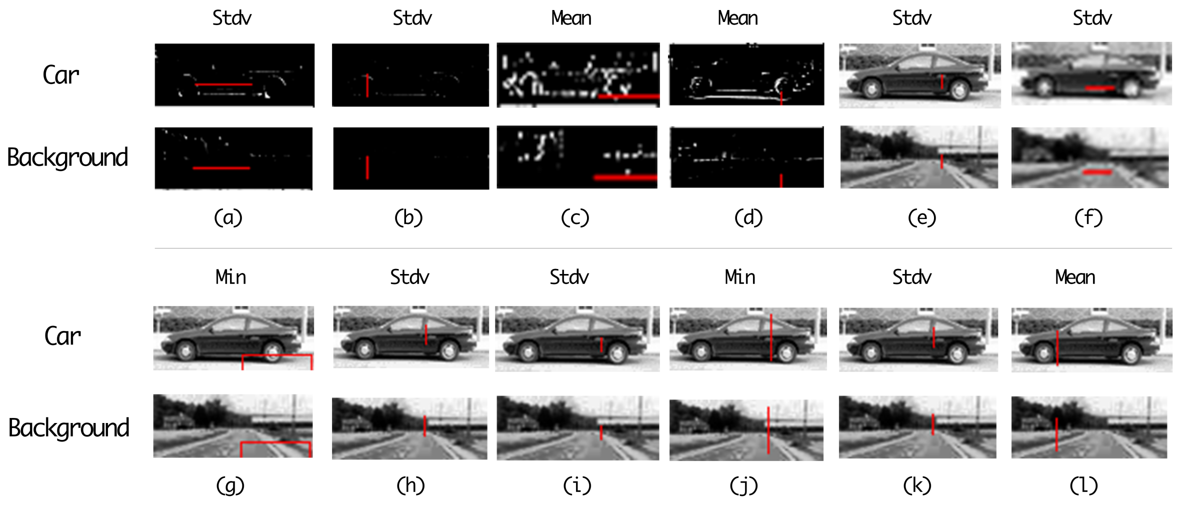


Figure 4.6: A comparison of the 12 extracted regions for the Car dataset with an example from each class.

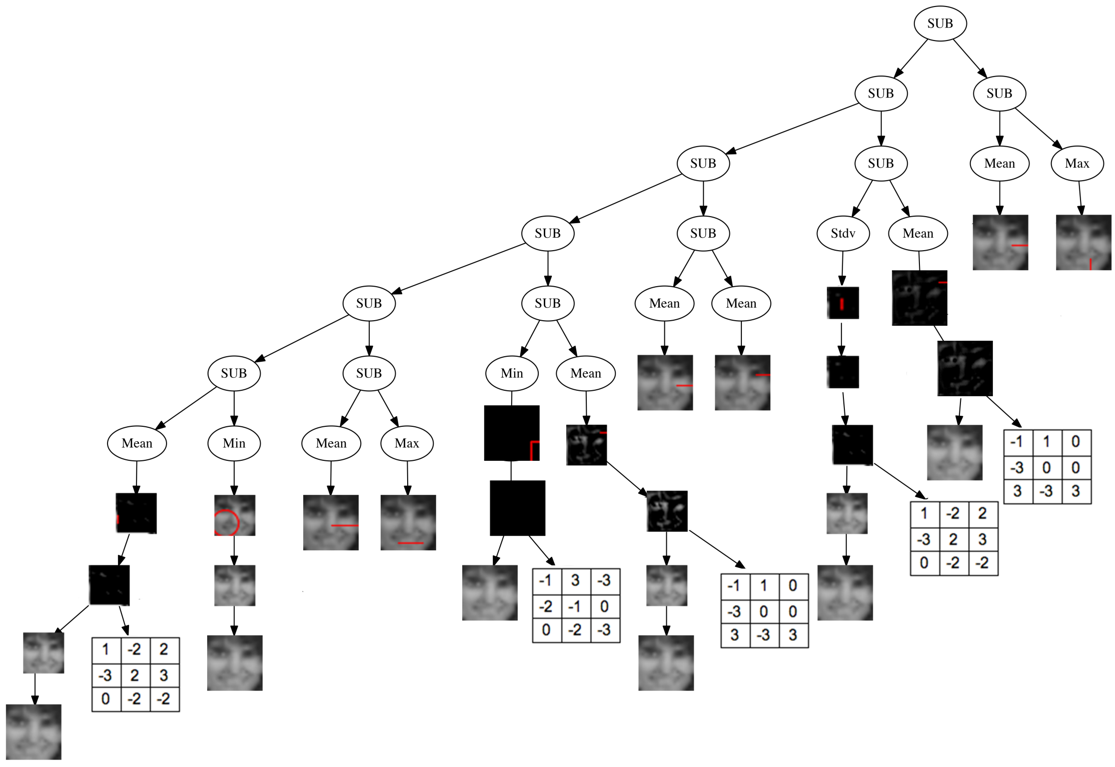
similar to an edge detector. For example, as shown in Figure 4.5, the filter will find vertical edges (large pixel changes in the y direction), and highlight these in white while leaving the rest of the image (without large changes) dark.

Comparing the aggregation windows gives a good idea of key regions identified by the evolutionary process, which are shown in Figure 4.6. Cars can be facing either left or right in the image, so regions learnt need to be independent of such directions. The regions found tend to correspond to boundaries of cars, such as the base of the chassis or the frame around a window. A large number of such regions was learnt, as it is unlikely a background image will feature similar values to a car in all of these regions. For example looking at the background image in Figure 4.6, the image has much greater variation in such regions whereas the cars pixel value will be relatively constant as the vast majority of cars are painted a single colour.

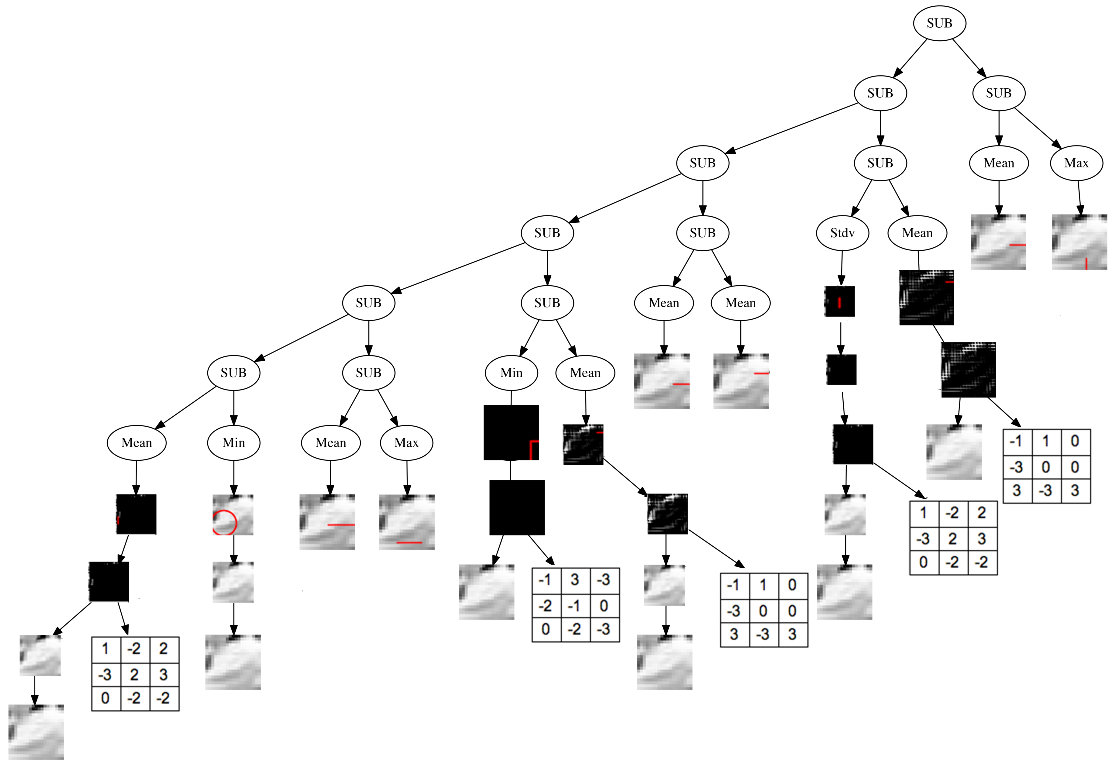
4.5 Faces

The Face task is to distinguish between photos of a face, or photos of a background image. The tree in Figure 4.7 achieved 97.70% testing accuracy. The tree had a maximum depth of 10, and the convolutional tier was a maximum of four layers high.

Pooling was used throughout the tree, and several filters were also used. However, many of the evolved filters appear overly dark. A potential fix for this is not limiting the output pixels to be in the range $0 - 255$, instead allowing free range of the pixels, and then only for



(a) Face



(b) Background

Figure 4.7: A good performing individual for the Faces dataset. This individual achieved 97.70% testing accuracy.

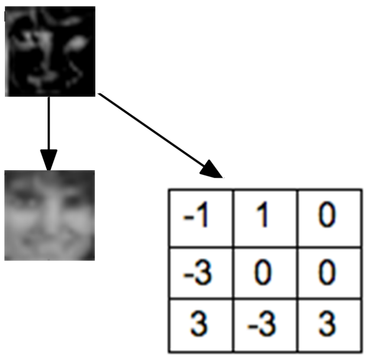


Figure 4.8: An evolved filter from the Faces dataset capable of edge detection.

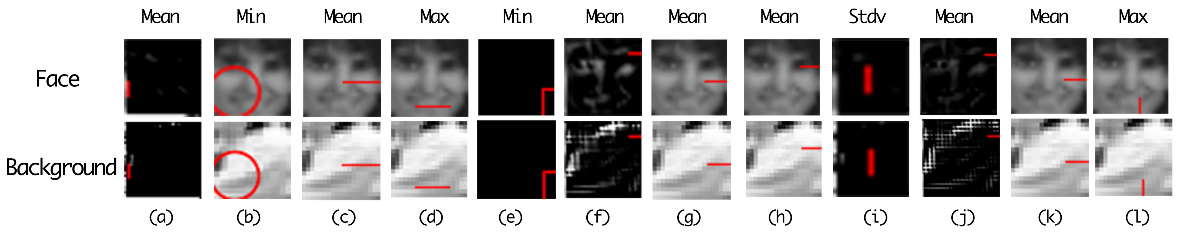


Figure 4.9: A comparison of the 12 extracted regions for the Faces dataset.

visualisation normalising back into the range $0 - 255$.

One interesting filter is shown in Figure 4.8, which was used twice in the tree. We can see this serves as a form of edge detection, however, the output is not completely crisp due to the filter being asymmetric.

The evolved tree (pictured in Figure 4.7) is relatively large, which reflects the difficult nature of the task. There are many filters which when applied to the image result in overly dark output images, which makes it difficult to distinguish between the two classes in this case. However, for other images in the training and testing set, these filters will likely "activate" to show a difference between the two classes. There are many branches covering different regions of the image, accounting for various regions across the large dataset (i.e. generalising rather than being restricted to the example image).

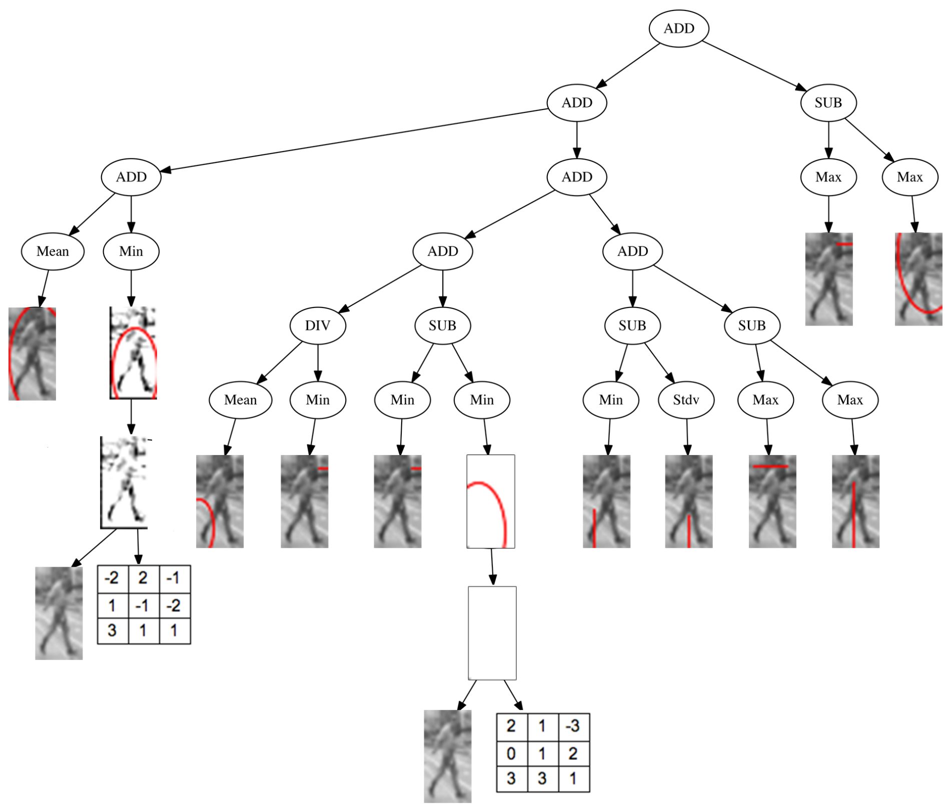
Figure 4.9 shows the evolved regions with an example from each of the two classes. The regions appear to focus on physical facial features, such as the eye region (h), cheek region (g, k) and the mouth region (d, l).

4.6 Pedestrians

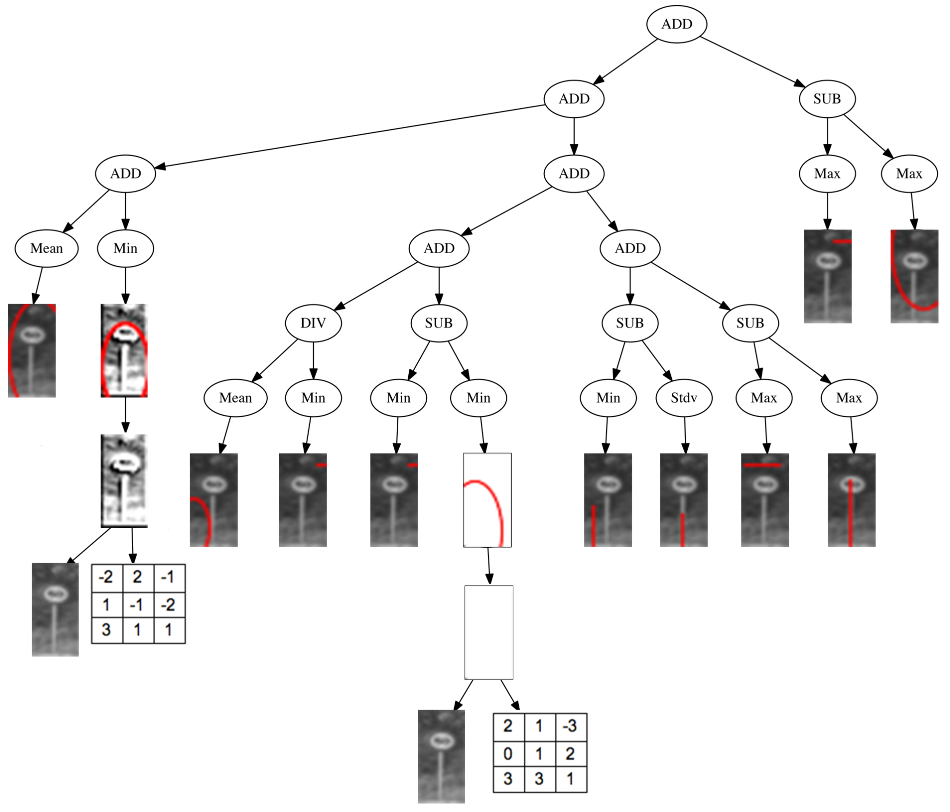
The pedestrian dataset aims to distinguish photos of pedestrians from photos of backgrounds. The tree shown in Figure 4.10 achieved 84.08% testing accuracy.

The tree is 8 layers deep at the highest point, which reflects the difficult nature of the dataset for classification. More complex datasets tend to result in deeper trees. Pedestrians can be in all different positions (and regions) in the image, which makes it very difficult to distinguish between pedestrians and backgrounds. This is likely why some of the evolved regions cover almost the entire image, to try and capture the pedestrians position within the image. No pooling was used in this tree, which likely means too much information was being lost when the size of the image was reduced. Only two filters were used for this tree, shown in Figure 4.11.

Filter 4.11(a) converts the image to a high contrast representation of the original image,



(a) Pedestrian



(b) Background

Figure 4.10: A good performing individual on the Pedestrian dataset. This individual achieved 84.08% testing accuracy. An example from each of the two classes is given for comparison.

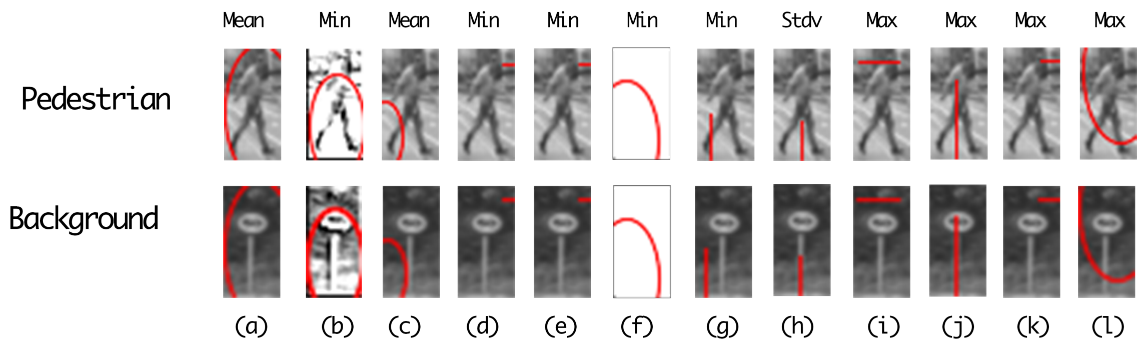
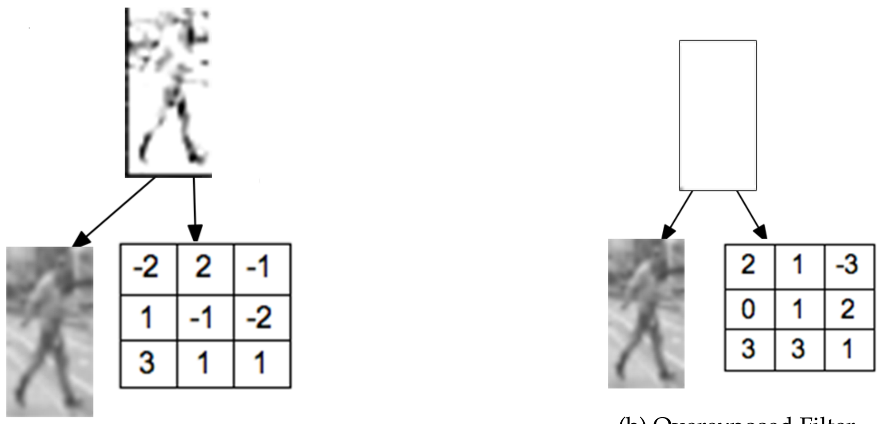


Figure 4.12: A comparison of the 12 extracted regions for the Pedestrian dataset.

and Filter 4.11(b) completely over exposes the image resulting in a full white image. This shows one of the benefits of visualisation, i.e. potential problems in the solution can be identified and fixed. To prevent this over exposure occurring (or the alternative case of a completely under exposed black image), a maximum value could be specified for the absolute sum of the filter or the pixel values could be allowed free range and then scaled back to the allowable range ($0 - 255$) for visualisation purposes.



(a) High Contrast Filter

(b) Overexposed Filter

Figure 4.11: Two of the evolved filters for the Pedestrian dataset.

4.7 Chapter Summary

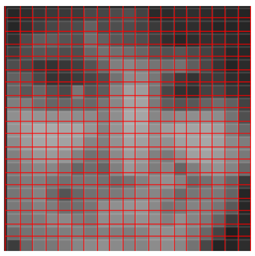
The GP architecture lends itself naturally to visualisation. The tree-based representation provides an intuitive view of each step in the classification process. As shown in the visualisations above, it is clear that the program learns to evolve many useful filters, such as filters able to perform edge detection and filters able to increase the contrast of an image. Visualisation can also help find issues in implementation, for example, above a filter which completely over-exposed the image was identified. This can provide insight for improving the algorithm in the future.

Chapter 5

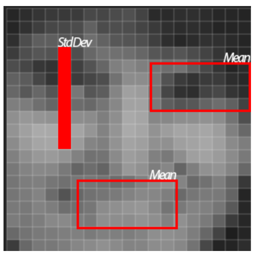
Feature Construction

5.1 Introduction

While the evolved tree from the proposed solution can be used as a classifier, due to the structure of the tree it can also be used as a form of feature construction. Feature construction involves creating a new set of useful features from the original feature space, often with the goal of dimensionality reduction. Specifically, in the case of images the feature space is the entire set of pixel values (so there are $image\_width * image\_height$ features). Feature construction aims at generating new features from this feature space. In this context this involves applying an aggregation function over a region of the (potentially pooled/convolved) image. An example is given below.



(a) Original Features



(b) Constructed Features

Figure 5.1: A comparison of two feature sets for the Face dataset.

To retrieve the automatically constructed features from the proposed solution, the output of the aggregation tier for a given tree can be used. Each node in the aggregation tier represents a constructed feature, and the output of this node for a particular image is the feature value in this new set of features. The number of constructed features is equal to the width of the aggregation tier (as this tier is limited to a single layer high). The number of constructed features will be significantly lower than the total number of original features (pixels) in the image, so should help with not just the accuracy of the classifier but also reducing the time taken to train the classifier.

Figure 5.2 shows the aggregation tier of one program for the Pedestrian dataset

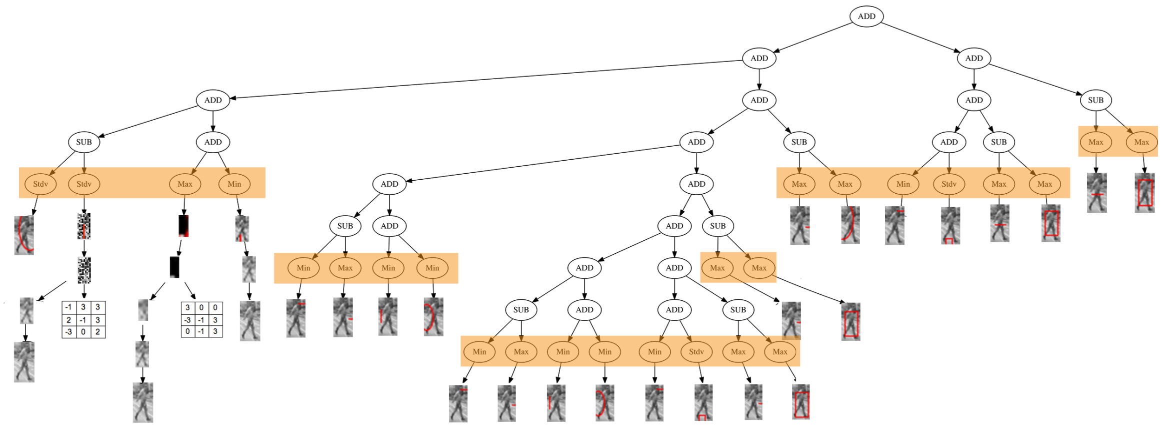


Figure 5.2: An example tree for the Pedestrian dataset. The layer highlighted in yellow is the aggregation tier, and corresponds to the constructed features.

We can see there are 26 constructed features. Compared to the original number of $image\_width \times image\_height = 18 \times 36 = 648$ features, this is a large reduction in the dimensionality of the problem.

To test the performance of the constructed features, various classifiers were trained and the performance on an unseen testing set analyzed. This is not only useful for evaluating the "goodness" of the learnt features, it can also give an insight to how well the classification tier of the program is working (i.e. if other classifiers achieve a high accuracy, while the tree results in a comparatively lower accuracy with the same features then the classification tier may need to be adjusted).

5.2 Chapter Goals

To evaluate if the proposed architecture from the New Methods section (ConvGP) can also be used as a valuable form of feature construction for general classification methods to reduce the computational cost, and dimensionality of the data.

5.3 Technical Details

The best performing feature set from each run was used for comparisons. The results are presented as "Mean $\pm$ Standard Deviation". The exact same training/testing split was used for comparing the constructed and the raw pixels, and no other parameters of the classifiers were adjusted. The time is measured in seconds, the training time includes the time taken to build and evaluate the model (the testing time is only the time taken to evaluate the model). The accuracy is the average percentage of instances correctly classified. When reporting the average number of constructed features, the number was rounding to the nearest integer. In some cases, the time taken was less than a hundredth of a second, in this case the time will simply be reported as 0.0 seconds. The speedup column calculates the number of times faster it was using the constructed features over the original featueres. For calculating the speedup, if either of the results are 0.0, then the + symbol will be used to show the minimum speedup gained from using the constructed features.

The two main measures of importance are the training time, and the testing accuracy. An unpaired Welch t-test was used to evaluate whether or not the difference in mean testing accuracy from the raw features and the constructed features was due to chance, or whether there was a true difference in the means. Again, Welch t-test was used as the variance of the two groups can not be assumed to be equal.

5.4 Results

5.4.1 JAFFE

The first point of comparison is the number of constructed features versus the original features, as shown in Table 5.1. The constructed features show a significant reduction in the original dimensionality, going from 23400 features to an average of only 8 features.

Table 5.1: Comparison between the number of original features and number of constructed features

	Raw Pixels	Min	Mean	Max
Number of features	23400	3	8	21

Table 5.2 reports the training and testing accuracy using the raw pixels versus the newly constructed features. In terms of average testing accuracy, the newly constructed features gave a statistically significant improvement over using the raw pixels for the Nearest Neighbour and Naive Bayes methods. Adaboost and Decision Trees achieved a lower average testing accuracy when using the constructed features, however, the differences were not statistically significant. The reason no improvement was seen with these methods is likely because all three are able to perform implicit feature selection, so when using the raw pixels they are not using all 23400 of them, but rather only a subset which has been deemed useful. Support vector machines were the only method which saw a statistically significant decrease in testing accuracy when using the newly constructed features, likely because SVMs are able to adequately deal with high dimensional problems.

Table 5.2: Comparison of accuracy with the two feature sets

	Training Accuracy		Testing Accuracy			
	Raw Pixels	Constructed	Raw Pixels	Constructed	T-Value	P-Value
Adaboost	100 ± 0	98.52 ± 3.88	84.44 ± 2.96	79.18 ± 13.45	2.3689	0.064
Decision Tree	100 ± 0	96.41 ± 7.63	82.22 ± 2.43	78.74 ± 13.66	1.731	0.1217
1NN	100 ± 0	99.93 ± 0.99	68.89 ± 5.51	81.52 ± 15.80	3.5173	0.039
Naive Bayes	87.78 ± 1.05	90.41 ± 9.84	61.11 ± 2.75	77.33 ± 16.58	6.8694	0.0001
SVM	100.00 ± 0.00	91.15 ± 9.53	91.11 ± 2.75	78.48 ± 14.59	5.7138	0.0007

Table 5.3: Comparison of computational time with the two feature sets.

	Training Time			Testing Time		
	Raw Pixels	Constructed Features	Speedup	Raw Pixels	Constructed Features	Speedup
Adaboost	0.36 ± 0.20	0.01 ± 0.01	**36x**	0.00 ± 0.00	0.00 ± 0.00	**1x**
Decision Tree	0.07 ± 0.04	0.00 ± 0.00	**7x**	0.00 ± 0.00	0.00 ± 0.00	**1x**
1NN	0.07 ± 0.02	0.01 ± 0.03	**7x**	0.08 ± 0.01	0.00 ± 0.00	**8x**
Naive Bayes	0.32 ± 0.06	0.00 ± 0.00	**32x**	0.22 ± 0.04	0.00 ± 0.00	**22x**
Random Forest	0.54 ± 0.02	0.01 ± 0.01	**54x**	0.00 ± 0.00	0.00 ± 0.00	**1x**
SVM	0.06 ± 0.04	0.01 ± 0.01	**6x**	0.01 ± 0.00	0.00 ± 0.00	**1x**

Table 5.3 shows the average training and testing time for the raw pixels vs the constructed features. It is clear that in all cases the constructed features resulted in drastically reduced training times, for testing times the times were already so small (i.e. less than a hundredth of a second) that no clear improvement was seen.

For the JAFFE dataset, the constructed features offered a statistically significant improvement over the raw pixels for Naive Bayes and nearest neighbour in terms of testing accuracy. For SVM, the constructed features performed significantly worse than the raw pixels. For Decision trees and Adaboost, there was no statistically significant difference, however, the constructed features offered drastically reduced training and testing time.

5.4.2 Cars

Table 5.4: Comparison between the number of original features and number of constructed features.

	Raw Pixels	Min	Mean	Max
Number of features	4000	3	13	27

With the Cars dataset, again, we see a drastic decrease in the dimensionality of the task when using the constructed features in comparison to the raw pixels as shown in Table 5.4. When comparing the average testing accuracy from Table 5.5, the constructed features offer comparable accuracy (no statistically significant difference) to using the raw pixels for both Adaboost and Decision Trees, however, for the other classifiers the average accuracy slightly deceases.

The constructed features show a significant speedup for both training and testing times for all classification methods, as shown in Table 5.6. For the Cars dataset, there is a trade-off between computational time and classification accuracy. The constructed features offer lower training and testing times, however, this is at the expense of slightly decreased average testing accuracy.

Table 5.5: Comparison of accuracy with the two feature sets on the Cars dataset.

	Training Accuracy		Testing Accuracy			
	Raw Pixels	Constructed	Raw Pixels	Constructed	T-Value	P-Value
Adaboost	95.27 ± 1.95	92.47 ± 2.76	88.27 ± 1.09	88.98 ± 3.07	1.0033	0.3896
Decision Tree	99.53 ± 0.09	96.91 ± 2.16	85.27 ± 3.08	89.07 ± 3.08	2.1022	0.1703
1NN	100.00 ± 0.00	99.93 ± 0.45	94.00 ± 1.40	89.81 ± 3.55	4.7041	0.0423
Naive Bayes	93.00 ± 0.43	89.50 ± 3.69	92.20 ± 0.71	88.31 ± 3.22	7.3093	0.0008
SVM	100.00 ± 0.00	92.71 ± 2.97	95.33 ± 0.90	90.76 ± 3.03	7.4927	0.0049

Table 5.6: Comparison of computational time with the two feature sets.

	Training Time			Testing Time		
	Raw Pixels	Constructed Features	Speedup	Raw Pixels	Constructed Features	Speedup
Adaboost	2.285 ± 0.117	0.012 ± 0.011	**190x**	0.003 ± 0.001	0.000 ± 0.001	**3x**
Decision Tree	0.883 ± 0.085	0.004 ± 0.005	**220x**	0.007 ± 0.028	0.000 ± 0.001	**7x**
1NN	3.284 ± 0.117	0.101 ± 0.543	**32x**	5.646 ± 0.481	0.017 ± 0.009	**332x**
Naive Bayes	0.791 ± 0.052	0.003 ± 0.003	**263x**	0.569 ± 0.016	0.002 ± 0.001	**284x**
SVM	0.255 ± 0.031	0.006 ± 0.005	**42x**	0.016 ± 0.002	0.000 ± 0.001	**16x**

5.4.3 Faces

Table 5.7: Comparison between the number of original features and number of constructed features on the Faces dataset.

	Raw Pixels	Min	Mean	Max
Number of features	361	3	14	35

With the Faces dataset, the number of features was decreased from 361 (raw pixels) to an average of only 14 features.

In terms of testing accuracy, the constructed features showed no statistically significant difference to using the raw pixels for all classification methods except for SVMs, as shown in Table 5.8. While achieving comparable testing accuracy, the constructed features also offered a drastic speed up in both training and testing time (Table 5.9). This shows for the Faces dataset, the constructed features are able to represent the image in a far lower dimensionality than using the raw pixels while achieving comparable accuracy with drastically reduced computational time. The one exception being with SVMs, which were able to achieve higher classification accuracy while using the raw pixels, however, a significant speedup in training and testing time was seen using the constructed features.

Table 5.8: Comparison of accuracy with the two feature sets on the Faces dataset.

	Training Accuracy		Testing Accuracy			
	Raw Pixels	Constructed	Raw Pixels	Constructed	T-Value	P-Value
Adaboost	91.51 ± 0.48	92.46 ± 2.55	90.80 ± 1.37	92.12 ± 2.52	1.582	0.2545
Decision Tree	99.72 ± 0.11	98.20 ± 2.12	96.57 ± 0.38	96.05 ± 1.80	1.7927	0.1232
1NN	100.00 ± 0.00	99.93 ± 0.29	96.52 ± 0.31	96.90 ± 2.63	1.1516	0.2631
Naive Bayes	91.62 ± 0.70	91.51 ± 3.33	92.19 ± 0.45	91.59 ± 3.32	1.3766	0.1903
SVM	99.38 ± 0.18	94.45 ± 3.42	97.48 ± 0.31	94.51 ± 3.27	7.647	0.0001

Table 5.9: Comparison of computational time with the two feature sets.

	Training Time			Testing Time		
	Raw Pixels	Constructed Features	Speedup	Raw Pixels	Constructed Features	Speedup
Adaboost	1.02 ± 0.12	0.060 ± 0.046	**17x**	0.00 ± 0.01	0.00 ± 0.001	**1x**
Decision Tree	0.89 ± 0.12	0.032 ± 0.025	**27x**	0.01 ± 0.02	0.00 ± 0.003	**1x**
1NN	6.70 ± 0.38	0.453 ± 0.430	**14x**	8.56 ± 0.68	0.46 ± 0.182	**18x**
Naive Bayes	0.45 ± 0.06	0.019 ± 0.018	**23x**	0.33 ± 0.02	0.01 ± 0.008	**33x**
SVM	0.96 ± 0.15	0.052 ± 0.035	**18x**	0.01 ± 0.00	0.00 ± 0.001	**1x**

5.4.4 Pedestrian

Table 5.10: Comparison between the number of original features and number of constructed features on the Pedestrian dataset.

	Raw Pixels	Min	Mean	Max
Number of features	648	3	14	37

The pedestrian dataset also saw a drastic decrease in the dimensionality of the images when using the constructed features over the raw pixels, as shown in Table 5.10.

Table 5.11: Comparison of accuracy with the two feature sets on the Pedestrian dataset.

	Training Accuracy		Testing Accuracy			
	Raw Pixels	Constructed	Raw Pixels	Constructed	T-Value	P-Value
Adaboost	79.26 ± 1.81	80.43 ± 2.44	77.72 ± 1.57	79.66 ± 2.31	2.067	0.1747
Decision Tree	99.28 ± 0.08	92.17 ± 5.84	89.24 ± 0.31	85.58 ± 3.46	9.0089	0.0001
1NN	100.00 ± 0.00	99.86 ± 0.67	98.08 ± 0.24	85.91 ± 6.48	17.4614	0.0001
Naive Bayes	77.21 ± 0.24	78.63 ± 2.64	76.53 ± 0.35	78.25 ± 2.50	5.1795	0.0002
SVM	90.65 ± 0.12	82.57 ± 3.33	84.48 ± 0.41	82.35 ± 3.27	5.094	0.0001

Table 5.12: Comparison of computational time with the two feature sets.

	Training Time			Testing Time		
	Raw Pixels	Constructed Features	Speedup	Raw Pixels	Constructed Features	Speedup
Adaboost	3.108 ± 0.496	0.102 ± 0.087	**30x**	0.006 ± 0.001	0.002 ± 0.001	**3x**
Decision Tree	4.586 ± 0.636	0.072 ± 0.049	**63x**	0.007 ± 0.001	0.002 ± 0.001	**3x**
1NN	27.583 ± 3.088	0.901 ± 0.315	**30x**	41.437 ± 4.483	1.048 ± 0.385	**39x**
Naive Bayes	1.144 ± 0.056	0.027 ± 0.019	**42x**	0.872 ± 0.012	0.020 ± 0.013	**43x**
Random Forest	3.093 ± 0.164	1.078 ± 0.196	**2x**	0.096 ± 0.008	0.110 ± 0.032	**0x**
SVM	84.683 ± 10.504	0.190 ± 0.149	**445x**	0.029 ± 0.006	0.002 ± 0.002	**14x**

Table 5.11 gives a comparison of the training and testing accuracies for the Pedestrian dataset. A statistically significant increase in testing accuracy was seen for Naive Bayes when using the constructed features over the raw pixels. For Adaboost, no statistically significant difference was seen between the two sets of features. For the decision tree, SVM, and Nearest Neighbour methods, a decrease was seen in testing accuracy when using the constructed features. In Chapter 3, the high performance of the nearest neighbour method for the Pedestrian dataset was explained, which is again relevant for these results.

For the training and testing time, a significant speedup was seen for all methods. For example, with the SVM, while the accuracy decreased 2% on the testing set, the training time reduced from 84 seconds to 0.2 seconds on average, and the testing time from 29 milliseconds to 2 milliseconds on average. Table 5.12 shows the average speedup for all classification methods.

5.5 Chapter Summary

The proposed architecture is able to serve as a valid method of feature construction by treating the output of the aggregation tier as the constructed features. These newly constructed features proved useful for classification, offering comparable or even increased testing accuracy over using the raw pixels with faster training and testing time for most classification methods trialled. The one notable exception was SVMs, SVMs on average saw a decrease in testing accuracy when using the constructed features, however, a drastic speedup was still seen for training and testing time.

Chapter 6

Conclusions and Future Work

6.1 Major Conclusions

The overall goal of the project was to develop a novel approach to binary image classification, which utilises state-of-the-art techniques from both deep learning and evolutionary computation paradigms. A GP based solution has been developed to achieve this goal, which takes key ideas from convolutional neural networks such as the convolution and pooling operators and combines these with a strongly-typed GP architecture to perform binary image classification. The proposed solution was tested on four widely used image classification datasets, from a variety of domains to ensure the developed solution was domain-independent. Several well-established classification methods from a range of machine learning paradigms were used as benchmarks for comparison.

1. For classification, the proposed method was able to outperform many general classification methods in terms of maximum testing accuracy in most cases. Although the method was unable to outperform state-of-the-art convolutional neural networks' classification accuracy, the proposed solution offered faster testing , and the architecture was able to be evolved automatically rather than manually crafted.

2. The proposed method offered high interpretability of the resulting models. The method offered greater visualisation ability than the state-of-the-art convolutional neural networks, which are almost impossible to visualise. A high performing model from the proposed method for each dataset was visualised as a tree structure, where each node shows the intermediary steps of the algorithms. Visualisation is an important tool for human comprehension of deep structures and can help to identify potential problems in learnt models.

3. For feature construction, the output of the aggregation tier can be used as a set of automatically constructed features. The constructed features for each dataset offered drastically reduced dimensionality over using the raw pixels, while on average matching (and in some cases improving) classification accuracy for the classification methods trialled. The constructed features also resulted in large speedup in both training and testing time for all classification methods trialled over. This shows the tree can serve as a useful method of feature construction, for reducing the images dimensionality while maintaining (or even improving) testing accuracy on various datasets.

6.2 Future Work

There are some limitations with the current approach, which could be mitigated in the future with further development on the following points

The four datasets used still have a moderate amount of data available for training. It has been shown that with an adequate fitness function genetic programming can learn a good solution from only a small number of instances [48]. Improved fitness functions could be trialled on newer datasets where there are a few training instances to learn from. It is likely that with such a fitness function, the benefits over other classifiers (such as the general classifiers examined and convolutional neural networks) would begin to become more apparent.

In some instances, the 2TGP solution outperformed the proposed solution. One architecture could potentially obtain the benefits of both if the architecture was changed such that one branch was restricted to the 2TGP architecture and the other branch is free to develop from the ConvGP architecture.

A visualisation method was proposed for the evolved trees, future work could extend the visualisation code used to create a package to serve as a general method of visualisation for evolved trees from various GP methods.

As with other current state-of-the-art image classification methods, a large limitation is the training time taken to learn the model. Future work could look to further improve the computational efficiency of the work.

Bibliography

[1] T. Hartley, "When parallelism gets tricky: Accelerating floyd-steinberg on the mali gpu," 2014.

[2] H. R. Roth, L. Lu, A. Farag, H.-C. Shin, J. Liu, E. B. Turkbey, and R. M. Summers, "Deeporgan: Multi-level deep convolutional networks for automated pancreas segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 556–564, Springer, 2015.

[3] W. B. Langdon, R. Poli, N. F. McPhee, and J. R. Koza, "Genetic programming: An introduction and tutorial, with a survey of techniques and applications," in *Computational intelligence: A compendium*, pp. 927–1028, Springer, 2008.

[4] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 815–823, 2015.

[5] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1285–1298, 2016.

[6] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[8] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[9] M. D. Zeiler, G. W. Taylor, and R. Fergus, "Adaptive deconvolutional networks for mid and high level feature learning," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2018–2025, IEEE, 2011.

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

[11] 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.

[12] M. Suganuma, S. Shirakawa, and T. Nagao, “A genetic programming approach to designing convolutional neural network architectures,” *arXiv preprint arXiv:1704.00764*, 2017.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[14] A. Lensen, H. Al-Sahaf, M. Zhang, and B. Xue, “Genetic programming for region detection, feature extraction, feature construction and classification in image data,” in *European Conference on Genetic Programming*, pp. 51–67, Springer, 2016.

[15] M. Iqbal, B. Xue, H. Al-Sahaf, and M. Zhang, “Cross-domain reuse of extracted knowledge in genetic programming for image classification,” *IEEE Transactions on Evolutionary Computation*, 2017.

[16] J. F. Peters, *Foundations of Computer Vision: Computational Geometry, Visual Image Structures and Object Shape Detection*, vol. 124. Springer, 2017.

[17] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.

[18] M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects,” *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[19] P. Domingos, *Master Algorithm*. Penguin Books, 2016.

[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[21] Z. Michalewicz, “Heuristic methods for evolutionary computation techniques,” *Journal of Heuristics*, vol. 1, no. 2, pp. 177–206, 1996.

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

[23] J. R. Koza, “Genetically breeding populations of computer programs to solve problems in artificial intelligence,” in *Tools for Artificial Intelligence, 1990., Proceedings of the 2nd International IEEE Conference on*, pp. 819–827, IEEE, 1990.

[24] D. Atkins, K. Neshatian, and M. Zhang, “A domain independent genetic programming approach to automatic feature extraction for image classification,” in *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pp. 238–245, IEEE, 2011.

[25] 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.

[26] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in *European conference on computer vision*, pp. 818–833, Springer, 2014.

[27] 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.

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[29] X. Zeng, W. Ouyang, J. Yan, H. Li, T. Xiao, K. Wang, Y. Liu, Y. Zhou, B. Yang, Z. Wang, *et al.*, "Crafting gbd-net for object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[30] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang, "Residual attention network for image classification," *arXiv preprint arXiv:1704.06904*, 2017.

[31] M. M. Khan, A. M. Ahmad, G. M. Khan, and J. F. Miller, "Fast learning neural networks using cartesian genetic programming," *Neurocomputing*, vol. 121, pp. 274–289, 2013.

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

[33] P. Verbancsics and J. Harguess, "Generative neuroevolution for deep learning," *arXiv preprint arXiv:1312.5355*, 2013.

[34] T. Desell, "Large scale evolution of convolutional neural networks using volunteer computing," *arXiv preprint arXiv:1703.05422*, 2017.

[35] T. Salimans, J. Ho, X. Chen, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017.

[36] L. S. Shafti and E. Pérez, "Feature construction and feature selection in presence of attribute interactions," in *Proceedings of the 4th International Conference on Hybrid Artificial Intelligence Systems*, HAIS '09, (Berlin, Heidelberg), pp. 589–596, Springer-Verlag, 2009.

[37] J. Brank, D. Mladenić, and M. Grobelnik, *Feature Construction in Text Mining*, pp. 1–6. Boston, MA: Springer US, 2016.

[38] B. Tran, B. Xue, and M. Zhang, "Genetic programming for feature construction and selection in classification on high-dimensional data," *Memetic Computing*, vol. 8, no. 1, pp. 3–15, 2016.

[39] E. Hart, K. Sim, B. Gardiner, and K. Kamimura, "A hybrid method for feature construction and selection to improve wind-damage prediction in the forestry sector," in *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, (New York, NY, USA), pp. 1121–1128, ACM, 2017.

[40] A. Bishop, V. Ciesielski, and K. Trist, "Feature construction using genetic programming for classification of images by aesthetic value," in *International Conference on Evolutionary and Biologically Inspired Music and Art*, pp. 62–73, Springer, 2014.

[41] H. Al-Sahaf, M. Zhang, and M. Johnston, "Genetic programming evolved filters from a small number of instances for multiclass texture classification," in *Proceedings of the 29th International Conference on Image and Vision Computing New Zealand*, pp. 84–89, ACM, 2014.

[42] S. Agarwal, A. Awan, and D. Roth, "The uiuc image database for car detection," *Retrieved March*, vol. 1, p. 2007, 2002.

[43] F. Cheng, J. Yu, and H. Xiong, "Facial expression recognition in jaffe dataset based on gaussian process classification," *IEEE Transactions on Neural Networks*, vol. 21, no. 10, pp. 1685–1690, 2010.

[44] K. K. Sung, *Learning and Example Selection for Object and Pattern Detection*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1996. AAI0800657.

[45] S. Munder and D. M. Gavrila, "An experimental study on pedestrian classification," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 11, pp. 1863–1868, 2006.

[46] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[47] F. Chollet *et al.*, "Keras." `https://github.com/fchollet/keras`, 2015. Accessed: 2017-10-1.

[48] H. Al-Sahaf, M. Zhang, and M. Johnston, "Genetic programming for multiclass texture classification using a small number of instances.," in *SEAL*, pp. 335–346, 2014.