Multi-objective neuroevolution to optimize deep learning on artificially inflated datasets

Sasha Abramowitz
Computer Science
University of Cape Town
Cape Town, South Africa
reallysasha@gmail.com

Abstract

The current state-of-the art performance of deep neural networks (DNNs) are unattainable in certain problem domains because of a lack of large labeled datasets. Therefore there is a high demand for techniques that improve a DNNs accuracy when trained on small datasets. This has been somewhat provided by advances in regularization techniques such as data augmentation (DA). However, these techniques still cannot produce acceptably accurate DNNs in all cases. With the rise in computing power neuro-evolution (NE) has become a viable option for finding more performant topologies for DNNs. NE allows naive users to create highly performant networks without too much effort. Thus combining these two techniques could allow for less reliance on big datasets and allow users from diverse backgrounds to use DNNs to solves complex problems in their respective fields. Analysis of the most popular NE algorithms found that there are possible improvements that can be made.

Keywords

Convolutional neural networks; Data augmentation; Neuro-evolution; Multi-objective optimization

1 Introduction

In 2012 Krizhevsky et al. used convolutional neural networks (CNNs) to create state-of-the-art image recognition software [25]. Henceforth CNNs became increasingly popular in the fields of computer vision. After Krizhevsky, Sutskever and Hinton's discovery CNNs have only increased in accuracy [26] with little change to exactly how they work. This increase is likely due to three factors the hike in available large scale labeled datasets, the rise of cheaper and more efficient hardware and research into more effective DNN topologies.

The rise in the availability of large scale labeled datasets, which was greatly aided by the advent of the internet, has unfortunately not found its way into a number of fields that would benefit from CNNs. However, given the current state-of-the-art efficiency of CNNs [26] many fields could benefit from them, but CNNs (and in general DNNs) do not perform adequately without large amounts of data [24, 30]. This presents an interesting problem: what is the best way to allow CNNs to learn from little data or what is the most efficient way to create labeled data. This literature review will mostly focus on the former considering active learning [20] seems to be the best solution to the latter.

CNNs, which are synonymous with DNNs, need large amounts of data to generalize. This generalization can be thought of as the avoidance of overfitting, which was one of the main contributions by Hinton et al. in [25]. However, a key factor of DNNs is that they have many layers, this creates many trainable weights, with each successive weight allowing the DNN to learn a more complex function. Yet similarly to humans DNNs require large amounts of learning material (data) to be able to understand complex topics, if this extra data is not present the DNN not be able to learn a general solution to the problem and in the extreme case will only be accurate on the data with which it was trained, this is overfitting.

Therefore the question is no longer how to allow CNNs to learn from little data, but rather how can overfitting be avoided when training CNNs with small datasets. The current best performing solutions are regularization methods. Most regularization methods modify the artificial neural network in some way, however, DA is a regularization that focuses on inflating the dataset. DA is the process of artificially modifying the data in the dataset such that after modification new data points are created with labels being preserved. Thus it is highly effective at inflating the size of a dataset. Other regularization techniques attempt to make the ANN generalize by forcing weights to learn independently of one another and thus more generally. Both methods have shown to be successful when reducing overfitting for CNNs trained on large datasets [25] however, there have been mixed levels of success when dealing with small datasets [1, 2, 10].

With the increasing accuracy of CNNs, optimal topologies became a commonly researched topic. This research produced networks such as VGG-16 which has a topology hand designed by experts and performs excellently across a range of computer vision tests [29]. However, with the increase in performance and decreasing cost of hardware there have been many attempts to create automated methods which search for optimal topologies [5, 6, 8, 22]. This field is known as autoML. Multiple different methods have been attempted to improve autoML such as bayesian optimization and reinforcement learning however, the current most promising method is neuroevolution (NE).

NE is the process whereby the topology and hyper parameters of an ANN are optimized by an evolutionary algorithm (EA).

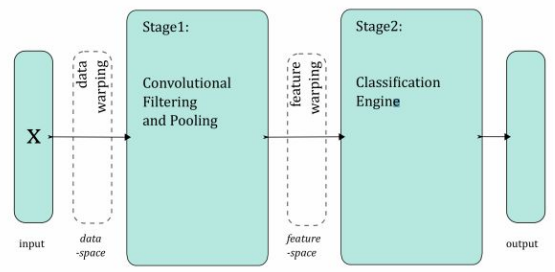

Figure 1. [16] The different points at which data augmentation can occur.

The basic premise of this is to add nodes and connections to the highest performing ANNs and see how it affects the accuracy continuously mutating and evaluating the most accurate ANNs. This is a resource intensive process since at each generation all ANNs must be trained and evaluated, which can take hours for a single ANN. This is why cheaper and more efficient hardware is so important, since it speeds up this training process.

Multi-objectivity has recently been shown to significantly improve the usefulness of NE [6, 8]. This is because NE can now be set to optimize more than the accuracy of an ANN. In cases where hardware is not as powerful, such as smartphones, ANNs do not perform well, since they require a large amount of computing power. Therefore using multi-objective NE one can set an objective of minimizing the ANNs size thus improving its performance and useability on low power devices.

This literature review will critique and analyse papers in the field of DA and NE, while trying to find out if there is a novel way in which to combine these approaches for the purpose of improving the performance of CNNs on small datasets. Specifically papers on the topics of DA, NE, multi-objective optimization (MOO) and EA will be considered to find the top performing solutions in those areas. Discussion will proceed to try and find any shortcomings of the current solutions. Combinations of these solutions will be analysed to see if they could produce CNNs that have been trained using small datasets, but still have performance comparable to state-of-the-art. This is done in an attempt to make CNNs more ubiquitous across a wide range of different fields so that more people can benefit from them.

2 Data augmentation

In previous research DA has been shown to both improve the accuracy of large scale CNNs [23] and reduce overfitting by inflating small datasets [1, 2]. It is well known that ANNs and specifically deep ANNs learn better the more data they are fed. However not all fields contain large quantities of labeled data and as such ANNs can be difficult to implement. However as seen in [1, 10] it is possible to get good performance from small datasets if DAs are used.

There are two main spaces where DA can occur namely data space and feature space. As seen in figure 1 if an augmentation occurs in data space it means that it has been applied before the convolution operations. On the other hand if an augmentation occurs in feature space it means that augmentation has occured after the convolution operations. As seen in [16] data space augmentations always outperforms feature space augmentations. Therefore this paper will only focus on data space augmentations and not feature space.

There are two main types of DA namely geometric and photometric augmentation. Geometric augmentation focuses on applying geometric transformations to the image. A good example of this is any affine transformation such as rotating, flipping and shearing the image. On the other hand photometric augmentations transform the colour by modifying the individual pixels of the image. An example of this is colour jittering. As seen in [10] geometric augmentations performed noticeably better than photometric augmentations. If an EA is given an opportunity to choose optimal DAs, then from the results in [10] one would expect it to choose mostly geometric augmentations.

All augmentations performed when inflating datasets must be label preserving. Therefore an augmented image of a cat must still resemble a cat, since if it does not it would be useless to any ANN trying to identify cats. As such once the augmentation is done it can be easily added to the dataset which contained the original image. An interesting side effect of label preserving DAs is that multiple DAs can be consecutively performed on a single image and the label will still be preserved. Therefore it can be investigated whether compound or single augmentations produce more accurate networks. In other words is a compound transformation more powerful than the sum of its parts?

As seen in [1] a choice must be made whether to augment the data prior to training the ANN or during the training process. Both approaches have their advantages and disadvantages. Prior augmentation saves time during training as no new images need to be generated during this period, but this takes up more space when training the ANN. On the other hand augmentation during training takes very little space since an image can be discarded once it has been used, but it does add more overhead to what can be a lengthy training time.

From the results seen in [10] DAs alone do not necessarily provide a large increase in accuracy of small datasets. Excluding cropping all the other DAs only provided up to a 2% increase [10]. While this is significant it is not enough to make the network comparable to current benchmarks, which in 2014 were 91.4% accuracy on the Caltech-101 dataset [26]. Thus for DAs to make a meaningful impact on the accuracy it would likely be useful to combine them with other machine learning methods.

2.1 Smart augmentations

A novel approach to DA comes in the form of style transfer, which was shown to improve the performance of an image recognition CNN [17]. Style transfer can be defined as extracting the *style* of the content of image A and transfer that *style* onto the content of image B, in such a way that it acts as a label preserving augmentation [18]. The reason style transfer was used is because Geirhos, R et al. realised that CNNs were learning the texture of items in the images instead of the shape of said items. The style transfer was used to increase the accuracy, while inflation of the dataset was a secondary goal since the dataset used was already large enough. However it would be interesting to see how these specific style transfers (greyscale, silhouette, edges and texture [17]) perform when specifically used to inflate data.

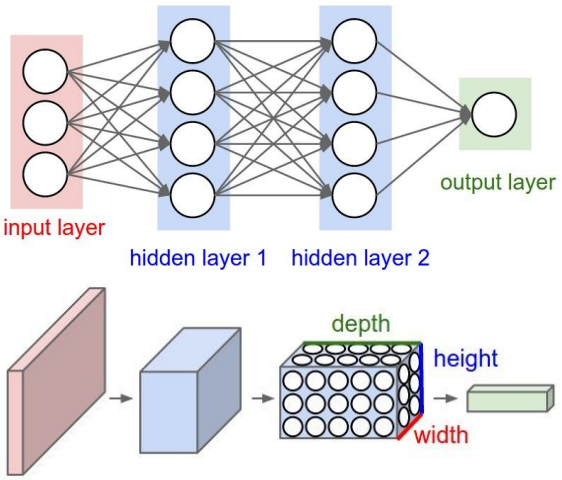

Figure 2. A simple representation of an ANN (top) and a CNN (bottom) [28]. The ANN shows how nodes are connected in a fully connected layer. The CNN shows how the image is downsized and reshaped from the convolution operations.

As can be seen in [1, 13, 21] attempts at trainable DA have been made. The method used in [13] augments new images by using a statistical model created by sampling the given dataset. This method showed improvement over standard DAs. At a high level the method in [1, 21] used a CNN to combine 2 images from the testing set. This CNN would reduce both images to the size of a single image and perform image recognition on this new image. The results from the classification of the newly created image could be backpropagated to the CNN, thus creating a learnable augmentation. Perez, L. et al. found that this type of style transfer did not significantly improve the performance of the network over regular DA methods [1] and found that it produced some non-label preserving transformation in very few instances. However, they did note that if given more time and a more complex network this may have yielded better results.

A seemingly more robust approach was attempted in [21]. However it used almost exactly the same method as [1], insomuch as it still combined images from the training set and downsized them using CNNs. The main difference between the two approaches being the loss functions. In [21] two parameters were used to create a linear combination of the errors of the style transfer networks and it is noted that future work should try to optimize these parameters. This seems like the exact kind of optimization that an EA would be well suited to, as such this is a possible domain that could be explored.

In [21] the researchers claimed better accuracy can be achieved when using style transfer as a DA along with regular augmentations rather than only using regular augmentations. This of course contradicts the conclusion of [1], which is significant since it shows that style transfer and as a result learnable augmentations can be used to effectively increase the performance of ANNs on small datasets.

3 Artificial neural networks

In recent years ANNs and more specifically CNNs have become best in class at image recognition [25]. CNNs are a type of deep learning neural network architecture, which learns hierarchically by capturing increasingly abstract concepts in each successive layer of the network. This is done by downsizing the image at each convolutional layer as seen in figure 2. This makes CNNs well suited to image recognition, which is likely why they have far surpassed any other type of algorithm in this domain [25].

3.1 Small dataset solutions

The best performing CNNs are examples of DNNs, this means that they contain multiple hidden layers. Having many hidden layers is what causes this need for large datasets since the more layers a neural network has, the more data it needs to generalise [24] and thus avoid overfitting. This is why DA has already been found to be useful [1, 2, 10], as it mostly solves this issue of not having enough data.

Overfitting occurs when the results of an ANN match too closely with the training data and as such the ANN struggles to recognise any new data not in its training set. This is most commonly caused by not having enough training data, which results in the ANN learning the noise of the training dataset. The main way to avoid this is to employ the use of regularization techniques such as DA. However, there are a number of regularization methods other than DA that have proved very successful in reducing overfitting.

First batch normalization, which in a high level sense this stabilizes the network. Stabilization is done by normalizing the values in hidden nodes, this reduces the amount by which hidden values shift around [14]. Thus reduces overfitting since it can be thought of as adding noise to each hidden layers output. The exact method subtracts the mean of the entire batch and then divides by the batches standard deviation for each node in a hidden layer.

Second transfer learning, this technique is used to transfer the knowledge of a pre-trained model to better suit the current task [31]. This is done by taking an existing ANN, one that currently performs a similar task to the task that will be learned, and retraining it on the data that fits the desired task.

Third dropout is another popular method used to reduce overfitting [25]. It stochastically decides not to update certain neurons during backpropagation leading to reduced interdependence among neurons. A good analogy to this is thinking of all nodes as a workforce and when dropping out a node it is analogous to a worker being absent and as such other workers (nodes) must learn to do the job of the absent worker.

Another popular solution is to create more data using a process called active learning [20]. This process initially requires a small amount of labeled data and uses the ANN to decide which other data points need to be labeled. The process usually followed is training the model on the small amount of labeled data and then using that model to find the data points that it is most unsure about. However *unsureness* is hard to quantify the easiest way to quantify it is by finding a data points distance from the decision boundary. Then the ANN will flag all data points too close to the decision boundary, allowing a user to label them and retrain using the new data. This is useful as it reduces the amount of data needed to be labeled, but can cause issues if the initial dataset

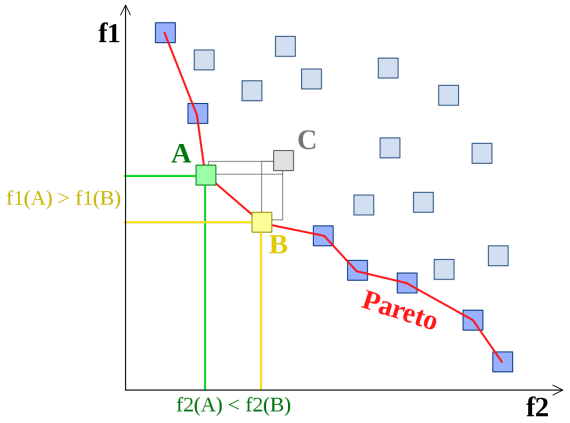

Figure 3. [32] The pareto front and pareto optimal solutions. Solution A is better than B at objective one and solution B is better than A at objective two. Both A and B dominate C.

does not represent the entire dataset very well. Added to this is the fact that it still requires humans to label the data, which is time consuming.

4 Multi-objective optimization

MOO presents a way to deal with problems that have multiple objectives. Each objective that MOO optimizes must have an associated objective function, such that if there are n objectives there will be a set of objective functions $\{f_1, ..., f_n\}$. In general MOO tries to find a point in the solution space (a solution) that minimizes (or maximizes depending on the objective) all of these objective functions. However there is usually not a single solution that is clearly better than all other solutions, in other words no single solution minimizes all objective functions better than every other solution. These are known as Pareto-optimal solutions [3]. Pareto-optimal solutions are the set of solutions that are not dominated by any other solutions. A solution is dominated if its performance for every objective is worse than a single other solution. Given a set of objective functions $\{f_1, ..., f_n\}$ that need to be maximized, pareto dominance is defined more formally as:

A solution vector **a** dominations a solution vector **b** iff:
$\forall\ i \in \{1, ..., n\}\ f_i(a) \geq f_i(b)\ \wedge\ \exists\ j \in \{1, ..., n\}\ f_j(a) > f_j(b)$
[9]

Thus the set of non-dominated solutions is called the Pareto front an example of which can be seen in figure 3.

4.1 Multi-objective evolutionary algorithm

EAs are a type of optimization and search method using the principles of darwinian evolution to evolve a population. In darwinian evolution only the fittests survive, therefore in EAs there needs to be a way to find the fittest members of the population (individuals). This is done by a predefined fitness function which can be thought of as how good an individual is at a specific task. EAs are the core of the NE algorithms that will be discussed in section 5.

However a traditional EA does have its drawbacks, most notably the fact that it can only optimize one objective and that most real world problems have multiple objectives [4] presents a clear problem. Thankfully there have been many attempts to solve this. The most naive and possibly most obvious solution is to make the fitness function a linear combination of all the objectives. Another approach and one that the currently best performing solutions is combining MOO and EA such that the EA uses MOO at its selection phase [19]. This can be thought of as the EA searching for all solutions while using the pareto front as a guideline.

Combining objectives using linear combinations inside the fitness function can cause unwanted behaviours. The linear combination method, which is usually associated with a penalty function to encourage certain behaviours, can prove to be problematic insomuch as it leads to the EA being hyper responsive to small changes in the coefficients of the penalty function [11]. This is of course an undesirable effect along with the fact that the less human input (setting the weights of the linear combination) required and the more the EA is *left alone* to learn the better and easier it is for the user.

As discussed in section 4.1 the pareto front is the set of all solutions that are non-dominated. However it is useful to understand this at a more abstract level. In a simple case given a EA with two objectives A and B, visualize plotting every solution on a graph with an x and y value corresponding to that solutions fitness for objective A and B respectively. Think of the pareto front as a (ideally hyperbolic) line along which all the best possible solutions congregate. If the requirement was to minimize A and B, then in the ideal case each successive generation would push this line closer and closer towards the origin, as better and better solutions were discovered. Figure 3 is an example of this.

Pareto approaches have been applied in many different ways to EAs such as NSGA and PEAS [9, 19]. As a broad overview the way that MOO is generally applied to EA is to focus on crossing-over and mutating parents on or near the pareto front. In most implementations there is also a priority to crossover and mutate parents that are in sparsely populated regions of the pareto front. This is so that the pareto front can be maximally explored, this means that there will be many diverse solutions which balance objectives in different ways. This mechanism of diverse mutation and crossover also helps preserve potentially innovative solutions, as if small innovative sub-populations were not given priority there is a higher chance that they would die out. Finally it has been noted that elitism highly increases the performance of these MOO EAs [19]. Elitism is the process of only allowing a small amount of the most fit individuals into the next generation, without them being mutated, this in turn would allow them to be reselected as parents.

4.2 NSGA-II

As the name suggests NSGA-II is an improvement of NSGA-I. NSGA-I was one of the first MOO EAs [19], but was heavily criticized. NSGA-IIs improvements are significant since they provide a good insight into what methods work for MOO EAs. The main issues of NSGA-I that were addressed are a high computational complexity, the lack of elitism and the need for a sharing parameter [19]. A direct improvement was making the fitness of an individual dependant on the number of individuals it dominates and not

allowing any dominated individual to be more fit than any non-dominated individual. The researchers note that this along with a new way of sorting non-dominated individuals decreases the computational complexity of the algorithm [19]. In this paper it is also noted that elitism is included in NSGA-II because it has performed so well in previous similar algorithms [19]. Elitism is the process of keeping the best performing individuals from the last generation in the current generation. This can be thought of as a minimum benchmark so that if the current generation only decreases in performance the best performers of the last generation will still be the best performers of the current generation.

Here a basic overview of how NSGA-II will be given as again it is useful to understand how this works in order to understand the best methods for MOO GAs. First parents and children are combined into a single population, this is the elitism step. Then this new population is sorted, using a fast non-dominated sorting method, this method was one of the main contributions of [19]. More than the top N individuals - where N is the population size - are selected for the next stage. This population is then sorted in order of increasing density and only the top N individuals are kept. Then, similarly to other EAs, the next population is created using mutation and crossover.

5 Neuroevolution

Neuroevolution (NE) is "the artificial evolution of neural networks using genetic algorithms" [5]. In early versions of NE only the networks hyper-parameter space was searched by the EA. More recent versions also search for improved topology of the network. This topology search was originally done while allowing the EA to not only modify the topology, but also find the optimal weights for the network, these NE algorithms are referred to as topology and weight evolving neural networks (TWEANNs). More recent iterations of NE algorithms only search for improved topologies and hyper-parameters, while weight updates are done by backpropagation. This method takes many more hours to train since each generation needs to train multiple networks via backpropagation, however some of the extra training time can be mitigated by training each NN in parallel on different machines.

5.1 NEAT

As noted by [5], the main difficulty when using EAs to search for better ANN topologies is that during crossover of two good solution a damaged offspring could be produced. This is because the two ANNs could have reached equivalently accurate solutions using completely different approaches and as such crossing these over would not produce a good result. This is because during the crossover of two networks with vastly different topologies, the topologies would be *mixed* together and thus would be unlikely to produce an ANN that takes the best aspects of the two networks. NEAT solves this problem by using speciation. Speciation, also known as niching, is the process of clustering the population by some similarity metric. In the case of NEAT is done by historical marking of genes.

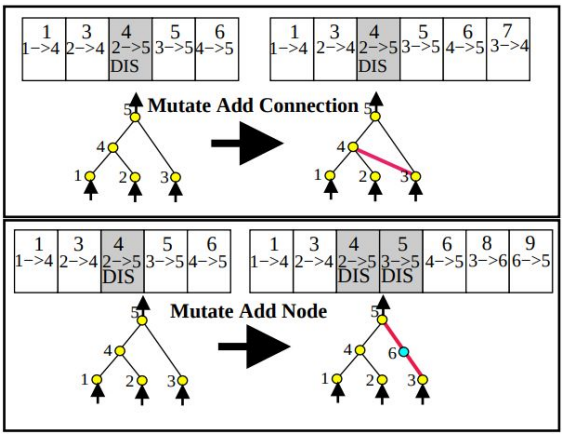

Figure 4. [5] Possible mutations NEAT can make: adding a connection and adding a node.

Historical marking is a way to tell origin of the individuals and thus is a good way to group genes into species. Therefore this will mitigate the negative effects of crossover between two equivalently accurate, but architecturally distinct ANNs, since only genes that have had a common parent can be used together to perform crossover. This is known as the competing conventions problem, which more formally is being able to express a solution to a weight optimization problem using distinct ANNs [15].

A useful side effect of speciation is that it encourages *innovation* within a species. This would not occur without speciation as when mutating and, for example, adding a node the new individual would not be optimized and likely would not last longer than a generation. However since every new individual belongs to a species, it can compete solely within that species without *fear* of being overpowered by other more optimised, but possibly worse architectures.

NEAT initially starts off with a population of minimal ANNs. This is done to allow NEAT not to waste time exploring inefficient topologies, since if topologies were initialized completely randomly a subset of inefficient species may occur, it also does not require users to have any expert knowledge to initialize intelligent weights. NEAT will then proceed to evaluate each individual and provide them with a fitness score. The top performers may be crossed over with other top performers whose historical marking line up and others may be randomly mutated. These mutations can take the form of weight changes as well as topology changes. There are two topology types of topology changes, namely adding a connection and adding a node, an example of this can be seen in figure 4. These knew children will be added into their correct species as the next generation and the process will start again.

As shown by Turabieh, H. in [12] NEAT has the potential to perform better than standard backpropagation for specific tasks. Turabieh, H. used a slightly modified version of NEAT, where mutations did not only complexify the ANN, but also simplified the network. Naturally this produces a more balanced search space. The ANN was trained to predict breast cancer and achieved an accuracy of 98.5% compared to a standard ANN achieving 95.3% accuracy [12].

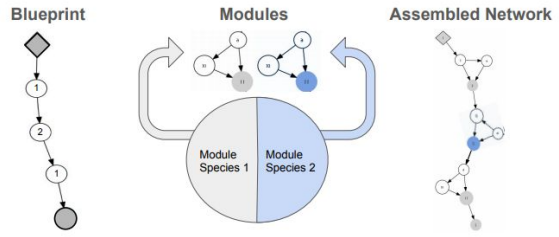

Figure 5. [6, 8] A representation of the blueprint and module system used by CoDeepNeat.

5.2 DeepNeat

DeepNeat, as the name implies, is an extension of NEAT, however it does differ in significant ways. The first way it differs is that it is not a TWEANN, instead DeepNeat updates its weights via backpropagation. Second it still represents individuals as graphs, but each node in the graph now represents a layer in the ANN [22]. However DeepNeat still uses NEATs main contribution, historical markings. Similarly to NEAT this is done so that meaningful crossover can be performed, which increases performance.

Since each node in a NEAT graph no longer represents a node in an ANN, but rather a layer each NEAT node now holds relevant hyper parameters [22]. These hyper parameters represent required information to construct the ANN from the NEAT graph, such as number of nodes in the layer, activation function and fully connected or convolutional layer, among other relevant parameters. This is done to aid construction when being evaluated. Another effect of each node representing a layer is that the connections between nodes no longer have a weighting as they did in NEAT, since weights are now learned through backpropagation.

During evaluation ANNs are constructed from their DeepNeat representation (i.e nodes in the DeepNeat graph are replaced with layers in the ANN) and trained using backpropagation on the provided training data. The accuracy of the network is then returned as the fitness and mutation and crossover are then performed similarly to how they would have been in NEAT.

5.3 CoDeepNeat

CoDeepNeat is an extension on DeepNeat, which only adds a single change, which has a large impact on the working of the algorithm. CoDeepNeat uses a common evolutionary computing technique called cooperative coevolution. This is a process, which combines simpler components (in this case small ANNs) to discover more complex ones [8]. This combination of smaller, simple ANNs into larger and more complex ones is essential to how CoDeepNeat operates.

CoDeepNeat works using a system of blueprints and modules [8] as seen in figure 5. Blueprints are graphs where each node points to a specific species. Modules are representations of a small ANN and each one belongs to a species. Modules are represented the same as graphs in DeepNeat, such that each node represents a layer in the ANN. CoDeepNeat assembles networks from blueprints by transforming each node in the blueprint into a randomly selected module from the species it was pointing to. Therefore a network is created by joining multiple smaller networks together, this is where cooperative coevolution comes into play with CoDeepNeat. Each network is then trained for a limited number of epochs on the given dataset and the resulting accuracy of the network is returned as the average fitness of both the blueprint and all the modules associated with this network. Using this fitness mutation and crossover are performed on both the blueprints and the modules. Once CoDeepNeat has run to completion the best performing network is trained until convergence and validated on an unseen validation set [8].

The difference between DeepNeat and CoDeepNeat is essentially only blueprints. CoDeepNeat can be thought of as DeepNeat with added blueprints, because modules in CoDeepNeat are simply the graph in DeepNeat. Therefore CoDeepNeat makes complex ANNs by repeating simple modules instead of only making complex modules. This is useful since many powerful networks use a topology with a repeating structure and the *module blueprint* system provides an easy way to replicated this behaviour. This repeating structure is further emphasized since whenever a blueprint points to a species more than once it chooses the same module from that species again, such that only the first choice of a module is random.

A drawback of both CoDeepNeat and DeepNeat is its training time. This is because an entire population of ANNs have to be trained every generation, to a reasonable level of convergence. It took only 35 CPU hours to obtain similar performance to a naive approach, but took over 9000 CPU hours to get a best in class approach [8]. This immense training time puts this system out of reach of many users without sufficient resources to train the ANNs in parallel.

It was found that CoDeepNeat could be easily adapted to use MOO. This proved very useful in [8] as it was used to maximize the accuracy, while minimizing the number of parameters. Contrary to expectations the multi-objective version of CoDeepNeat converged faster than the single objective version for the same task [8]. This is very interesting and may have a lot to do with the specific secondary objective as it was: generate a minimally complex network. However CoDeepNeat has not been tested with more than two objectives and it would be interesting to see how this would impact performance and if results would remain meaningful.

The multi-objective version works similarly to many MOO EAs, by crossing over and mutating the top performers from successive pareto fronts. However CoDeepNeat has the notion of a primary and secondary objective and as such does weigh the primary objective more heavily than the secondary. This results in the objectives on the pareto front being ranked by the primary objective, in other words the best objective is the one that has the best solution to the primary objective. This would likely lead to a poor approximation of the pareto front because solutions that perform well in the secondary solution, but poorly in the primary objective would be unlikely to last many generations. This can be preferable in certain situations although an algorithm which more uniformly explores the pareto front should be investigated.

5.4 LEMONADE

LEMONADE is a competing algorithm to CoDeepNeat, it achieves the same goal, but uses intelligent methods to

greatly increase the efficiency. To begin with LEMONADE stands for **L**amarckian **E**volutionary algorithm for **M**ulti-**O**bjective **N**eural **A**rchitecture **DE**sign [6]. This encapsulates the idea of the algorithm quite nicely, as what it does is use Lamarckian inheritance (which is a means of passing an individual's skills to its children) to greatly decrease the training time when compared to other NE methods.

There is one area where LEMONADE performs very well, namely efficiency. This is done by taking advantage of two key factors. Firstly lamarckian inheritance, in the case of NE lamarckian inheritance can be seen as giving a network a new head start. This is achieved by a process called network morphisms, which only perform small modifications to existing networks and use already generated, but similar topologies to set the weights of new architectures. This would of course greatly increase the initial accuracy of networks and achieve a similar final accuracy with less training. Secondly LEMONADE cleverly allows one to label an objective as cheap or expensive, using this the algorithm knows which objective it can easily evaluate. It then initially finds the most sparse individuals on the pareto front according to the cheap objectives. Once that is done it generates children mostly using individuals in the sparse regions of the pareto front. These children and their parents are again sorted according to their performance of the cheap objectives and only the top performers of this group are evaluated on the expensive objectives. This of course minimizes the number of times the expensive checks are run, which is very desirable to decrease running time. Obtaining the error of an ANN is an example of an expensive objective as it has to be trained and then evaluated on unseen data, while number of parameters (i.e ANN complexity) is an example of a cheap objective. Therefore in general evaluating expensive objectives take orders of magnitude longer than evaluating cheap objectives.

A major advantage of CoDeepNeat over LEMONADE is that CoDeepNeats entire architecture is geared towards allowing meaningful crossover. This is of course a very useful operation as it allows for more useful ways to evolve individuals when compared to random mutations. An interesting idea that arises from this would be to considering using lamarckian inheritance at the blueprint level of CoDeepNeat, this would of course require networks to store their weights which would massively increase the storage required to run the algorithm.

A possible shortcoming of LEMONADE is seen in its results [6]. It seems to trade off training time for small drops in accuracy, which makes sense since accuracy is an expensive objective and thus is not evaluated often. The results show that it has up to 0.5% more error on the CIFAR-10 dataset while using a larger network when compared to other network architecture search methods methods. However, when specifically compared to MOO NEs it did perform at least as well, if not better. This time versus accuracy trade-off can be desirable in many, if not most cases. However it is worth noting for those who have the computing resources and time to run other algorithms that will return a more accurate network.

5.5 AutoML

Both CoDeepNeat and lemonade are examples of automatic machine learning (autoML). These are systems which optimize ANNs without the need for expert knowledge. Both of these methods take a large amount of time to compute, but as computing resources get faster and cheaper autoML will become more accessible and will likely eventually become the norm. This is why novel autoML methods should be investigated since it is unlikely to be long before they become incredibly useful and take much less time to compute. This trend to try and provide autoML to everyone is already underway with companies like Google already having systems up and running [27].

6 Discussion

This literature review has indicated that there are a few ways to improve popular methods in DA and NE. First in [21] Lemley et al. proposed a novel augmentation strategy, which essentially transfers the style of one image onto another. They noted this method was highly successful however, it could be improved by optimizing two parameters relating to the style transfer. This would fit well into a NE strategy where an objective could be to find the most appropriate values for those parameters.

Second the main drawback of CoDeepNeat was its lengthy training time while LEMONADE specifically tries to optimize this using lamarckian evolution. There is high potential for investigation into using lamarckian evolution inside of CoDeepNeat to allow the ANNs which are created to have a *warm start*. This would most likely decrease training and slightly mitigate one of the main drawbacks of CoDeepNeat.

Third as was noted in section 5.2 the multi-objective version of CoDeepNeat does not accurately represent the entire pareto front. In fact it purposefully biases towards solutions that perform better in the primary objective. Since this objective is almost always accuracy this does make a lot of sense as one would want accuracy to be emphasized above all else however, a more uniform exploration of the pareto front would perhaps give rise to more novel and useful architectures. This exploration could be implemented similarly to how NSGA-II explores the pareto front, but the impact on accuracy would then need to be investigated.

7 Conclusion

There are currently many methods that improve the process of learning with minimal data, such as DA and regularization in general. Along with this the rise in popularity and availability of autoML has provided a large increase in the performance of many different ANNs. However it seems that these methods are yet to be combined to allow for more efficient learning on small datasets.

Given that MOO NE has been shown to produce state-of-the-art performance [6, 8], could objectives be developed that encourage creation of novel architectures which perform better on augmented data data or even small datasets? Or could selection of DA type be included in the evolutionary process of a NE algorithm such that novel augmentations do not need to be created, but rather the best performing augmentations are selected from a pool of known augmentations.

8 References

[1] Perez, L. and Wang, J., 2017. The effectiveness of data augmentation in image classification using deep learning. arXiv preprint arXiv:1712.04621, 1-7

[2] Nelson Marcelo Romero Aquino et al, 2017. The Effect of Data Augmentation on the Performance of Convolutional Neural Networks. CBIC paper 51, 8-10

[3] Van Veldhuizen, D.A. and Lamont, G.B., 1998, July. Evolutionary computation and convergence to a pareto front. In Late breaking papers at the genetic programming 1998 conference (pp. 221-228), 1-3

[4] Fonseca, C.M. and Fleming, P.J., 1993, June. Genetic Algorithms for Multiobjective Optimization: Formulation Discussion and Generalization. In Icga (Vol. 93, No. July, pp. 416-423), 1-2

[5] Stanley, K.O. and Miikkulainen, R., 2002. Evolving neural networks through augmenting topologies. Evolutionary computation, 10(2), pp.99-127, 1-12 22-24

[6] Elsken, T., Metzen, J.H. and Hutter, F., 2018. Efficient Multi-objective Neural Architecture Search via Lamarckian Evolution. arXiv preprint arXiv:1804.09081, 1-9

[7] Zitzler, E. and Thiele, L., 1998, September. Multiobjective optimization using evolutionary algorithms—a comparative case study. In International conference on parallel problem solving from nature (pp. 292-301). Springer, Berlin, Heidelberg, 1-2

[8] Liang, J., Meyerson, E., Hodjat, B., Fink, D., Mutch, K. and Miikkulainen, R., 2019. Evolutionary Neural AutoML for Deep Learning. arXiv preprint arXiv:1902.06827.

[9] Knowles, J. and Corne, D., 1999, July. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In Congress on Evolutionary Computation (CEC99) (Vol. 1, pp. 98-105).

[10] Taylor, L. and Nitschke, G., 2018, November. Improving Deep Learning with Generic Data Augmentation. In 2018 IEEE Symposium Series on Computational Intelligence (SSCI) (pp. 1542-1547). IEEE.

[11] rey Horn, J., Nafpliotis, N. and Goldberg, D.E., 1994, June. A niched Pareto genetic algorithm for multiobjective optimization. In Proceedings of the first IEEE conference on evolutionary computation, IEEE world congress on computational intelligence (Vol. 1, pp. 82-87).

[12] Turabieh, H., 2016. Comparison of NEAT and Backpropagation Neural Network on Breast Cancer Diagnosis. International Journal of Computer Applications, 139(8), pp.40-44.

[13] Hauberg, S., Freifeld, O., Larsen, A.B.L., Fisher, J. and Hansen, L., 2016, May. Dreaming more data: Class-dependent distributions over diffeomorphisms for learned data augmentation. In Artificial Intelligence and Statistics (pp. 342-350).

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

[15] Radcliffe, N.J., 1993. Genetic set recombination and its application to neural network topology optimisation. Neural Computing & Applications, 1(1), pp.67-90.

[16] Wong, S.C., Gatt, A., Stamatescu, V. and McDonnell, M.D., 2016, November. Understanding data augmentation for classification: when to warp?. In 2016 international conference on digital image computing: techniques and applications (DICTA) (pp. 1-6). IEEE.

[17] Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F.A. and Brendel, W., 2018. ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. arXiv preprint arXiv:1811.12231.

[18] Gatys, L.A., Ecker, A.S. and Bethge, M., 2016. Image style transfer using convolutional neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2414-2423).

[19] Deb, K., Agrawal, S., Pratap, A. and Meyarivan, T., 2000, September. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In International conference on parallel problem solving from nature (pp. 849-858). Springer, Berlin, Heidelberg.

[20] Settles, B., 2009. Active learning literature survey. University of Wisconsin-Madison Department of Computer Sciences.

[21] Lemley, J., Bazrafkan, S. and Corcoran, P., 2017. Smart augmentation learning an optimal data augmentation strategy. IEEE Access, 5, pp.5858-5869.

[22] Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N. and Hodjat, B., 2019. Evolving deep neural networks. In Artificial Intelligence in the Age of Neural Networks and Brain Computing (pp. 293-312). Academic Press.

[23] Salamon, J. and Bello, J.P., 2017. Deep convolutional neural networks and data augmentation for environmental sound classification. IEEE Signal Processing Letters, 24(3), pp.279-283.

[24] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), pp.1929-1958.

[25] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105), 1-3

[26] Thoma, M. (2019). State of the Art in ML. [online] Martin Thoma. Available at: https://martin-thoma.com/sota/ [Accessed 28 Apr. 2019], 1

[27] detection, A. (2019). AutoML for large scale image classification and object detection. [online] Google AI Blog. Available at: https://ai.googleblog.com/2017/11/automl-for-large-scale-image.html [Accessed 28 Apr. 2019].

[28] Convolutional Neural Networks for Visual Recognition: 2017. http://cs231n.github.io/convolutional-networks/:01/05/2018

[29] Das S (2017). CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more [online] Medium Available at: https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5 [Accessed 1 May 2019]

[30] Brownlee J. (2016) What is deep learning [online] Machine learning mastery Available at: https://machinelearningmastery.com/what-is-deep-learning/ [Accessed 1 May 2019]

[31] Brownlee J. (2017) Transfer learning for deep learning [online] Machine learning mastery Available at: https://machinelearningmastery.com/transfer-learning-for-deep-learning/ [Accessed 1 May 2019]

[32] Pareto efficiency [online] Wikipedia Available at: https://en.wikipedia.org/wiki/Pareto_efficiency [Accessed 1 May 2019]