Comparing Population-Based Algorithms to Fine-Tune a CNN for Image Classification

Anna Carter, 6702876
Department of Computer Science
University of Surrey
Guildford, UK
ac02351@surrey.ac.uk

Omojevwe Zion Ejechi, 6687869
Department of Computer Science
University of Surrey
Guildford, UK
oe00205@surrey.ac.uk

Melric Lazaro, 6695534
Department of Computer Science
University of Surrey
Guildford, UK
ml01663@surrey.ac.uk

***Abstract*—In this report, we compare the performance of a convolutional neural network for the classification of images in the CIFAR-10 dataset using three different optimisation algorithms for the minimisation of the error metric in the final layer, where previous layers have been pre-trained using stochastic gradient descent (SGD). Using SGD and social learning particle swarm optimisation (SL-PSO) as benchmark optimisation algorithms, we find our chosen algorithm (a modification of SL-PSO) performs better than SL-PSO, but fails to reach the accuracy level of SGD. We also compare the performance of the derived model when viewed as a single- and bi-objective optimisation problem (adding a Gaussian regulariser as the second objective), finding the single-objective solution typically performs better than the bi-objective solution.**

***Index Terms*—optimisation, classification, neural networks, particle swarm, gradient descent**

I. Introduction

Neural networks are a subset of machine learning models that aim to approximate a function that best fits the dataset done by optimising the weights of its networks. Their design is based on the complex neurological pathways found in the human brain of that enables them to learn by making millions of connections between other neurons. These neural networks can learn in various ways, reinforcement learning, semi-supervised learning, and supervised learning. In this paper we will discuss only supervised learning. In supervised learning, the ground truth labels of the dataset are available, and we can directly train the network based on the difference between its predictions and the actual value, calculating a resulting loss value. This allows for methods such as backpropagation to teach a network to perform well on image classification. This project aims to optimise neural network training by comparing the performance of single-objective (SGD, SL-PSO, SS-PSO (our own algorithm)) and multi-objective (NSGA-II) optimisation methods. The primary objective is to maximize prediction accuracy for unseen images, and we explore simultaneously minimising regularisation to prevent overfitting. SGD, a common gradient-based algorithm, will be evaluated alongside two evolutionary algorithms inspired by bird flocking behaviour: SL-PSO, SS-PSO. NSGA-II, a multi-objective evolutionary algorithm, is evaluated to balance prediction accuracy and regularisation. The effectiveness of each method will be assessed based on prediction accuracy and regularisation measures. The findings will provide valuable insights into the trade-offs between prediction accuracy and regularization, as well as the suitability of single-objective and bi-objective optimization approaches for neural network training.

Our own optimiser, SS-PSO, attempts to build on existing optimisers and boosts its performance by controlling the movement of particles when we think premature convergence is happening. Our method uses theory that is inspired by similar papers that deal with a similar problem, in more complicated ways.

A. Literature review

1) Architectures: Convolutional neural networks (CNNs) are the most common architecture for image classification. These networks are specifically designed to work with spatial data such as images by creating complex feature maps from them through deep learning. This allows the network to be unaffected by translations of input images, making it more robust in its responses. VGG [1], AlexNet, GoogLeNet, and ResNet are common architectures that work well on unseen image classification tasks [2].

a) AlexNet: AlexNet is one of the earliest CNNs that revolutionised deep learning. It featured ReLU activation functions instead of Tanh functions which was the standard at the time [3]. It also allowed for scalability in terms of being able to run on multiple GPUs and addressed overfitting issues by using methods such as dropout. AlexNet uses 60 million parameters which contributed to its accuracy , however, due to its age, newer architectures achieve greater parameter efficiency. AlexNet does not have great flexibility, as removing layers greatly affects its performance [4]. This is important as we want to improve running times of our population-based algorithms as much as we can, and this may involve scaling the model down [5]. While AlexNet was important for the development of CNNs, its aging architecture allowed for newer models to surpass it in terms of accuracy and efficiency, which is what we aim to utilise in our work.

b) ResNet and GoogLeNet: Newer architectures like ResNet and GoogLeNet improve on AlexNet by being much more efficient for training deep networks [6]. This is done through skipping connections in regard to ResNet to solve the vanishing gradient issue along with dimensionality reduction through 1×1 kernels with GoogLeNet. While ResNet and GoogLeNet both offer these powerful features, our original problem does not have a large dimensionality, as we work with very small images in comparison to what the network was originally designed with [7]. ResNet's relative performance improvment stems from having more parameters which allows for more effective training of deep networks than other architectures. This increase in the number of parameters for training puts more of a strain on our resource bound environment as a result. We ideally want to find a balance between accuracy and performance, which we find with VGG, which offers powerful training mechanisms with a simpler, more flexible architecture [8] as we explain later in section II.

c) Transformers: Transformers are commonly used for NLP tasks but can be modified for image classification as seen by the Vision Transformer [9] with strong performance on CIFAR-10 [10]. Transformers often have many more parameters than CNNs, requiring significantly more training data to surpass the performance of a CNN, and they are slower to run in resource-bound environments [11]. In section II we critically compare the architectures to decide on one that bests suit our problem.

2) Training algorithms:

a) Gradient-based algorithms: An optimiser is required by the architecture to train the model on the input dataset by optimising the weights of each layer in the network. Gradient based optimisers such as stochastic gradient descent (SGD) are well studied and benefit from large flexibility to be used in many problems [4]. Additionally, convergence is fast when the correct hyperparameters are chosen. Despite this, gradient based methods often suffer from vanishing/exploding gradients [12] because of backpropagation and as a result, hinders convergence. Importantly, they are sensitive to the landscape of the problem and can fall into local minima or plateaus along with not working with discontinuous problems. As a result, optimisers such as ADAM have been created to try and circumvent problems such as rugged landscapes.

b) Population-based algorithms: Population based optimisers such as Genetic Algorithms (GA) and Particle Swarms are an alternative method for optimisation. GA's use a set of potential solutions acting as an initial population which then goes under various techniques such as crossover, mutation, and selection to provide a set of offspring that may achieve better results. The diversity of the population allows for greater exploration of the search space, increasing the chances of reaching the global minima. Non-differentiable objective functions and noisy fitness landscapes are much better handled by these than other optimisers [13]. Population based optimisers are also usually more computationally expensive than other optimisers, due to the quantity of evaluations required per generation, which further increases according to the population size. Additionally, they require much more careful tuning of the hyperparameters for the optimiser to converge and therefore is much less flexible when presented new problems [14].

In our work, we look at Particle Swarm Optimisation (PSO), an optimisation technique, first introduced in [15]. Canonical PSOs use a set of particles as potential solutions. They do not use selection, crossover or mutation operators and instead works on a personal best and global best system, so less hyperparameters need tuning to work. The movement of these particles mimics the behaviour of bird swarms in nature, where individuals learn from the best member of the group and move towards them. This learning process updates the particles velocity and direction leading to improved fitness [16].

However traditional PSOs are not well designed out of the box to handle problem of large dimensionality and as a result, often does not converge. SL-PSO takes a step to improve this.

SL-PSO uses a 3-stage process for optimisation involving evaluation, swarm sorting and behaviour learning. In particular, the behaviour learning aspect consists of worse particles learning from better ones and are more influenced the lower they are in the rankings. It is much more efficient with problems of high dimensionality and as a result, can converge faster at the cost of its complexity.

Even with SL-PSO's improvements, particle swarm optimisers still often fall into local minima, preventing them from further exploring the search space and reaching the global optimum [17]. To address this, we explore strategies that can be used to potentially escape local minima such as incorporating GA techniques like particle mutation. This technique forms the foundation of our third proposed algorithm SSPSO.

B. Our contribution

In our project, the following contributions make significant contributions to the field of neural network optimisation:

- To address the problem of premature convergence with regular PSO we come up with a new way to calculate the probability of a particle mutating. The probability calculation assigns a higher probability to particles with a smaller Euclidian distance of a particle from the best particle, to trigger a particle to jump out of local minima. We call our new genetic algorithm SSPSO that combines the strengths of PSO and Genetic Algorithms (GA).
- We compare SGD and SL-PSO to our own proposed algorithm by evaluating the accuracy of the neural network on the testing and training images. We find that SGD outperforms evolutionary based algorithms and remains the leader in neural network optimisation methods.

II. Architecture

A. Description

In our work, we use a Convolutional Neural Network based on a simplified version of VGG with several 3×3 convolutional layers stacked as our throughput.

The simpler VGG architecture we use is based off the original VGG-16 network and consists of multiple 3×3 convolutional kernels with pooling at the end of each layer. The architecture also utilises dropouts in the fully connected layers in the second half of the network to improve generalisation

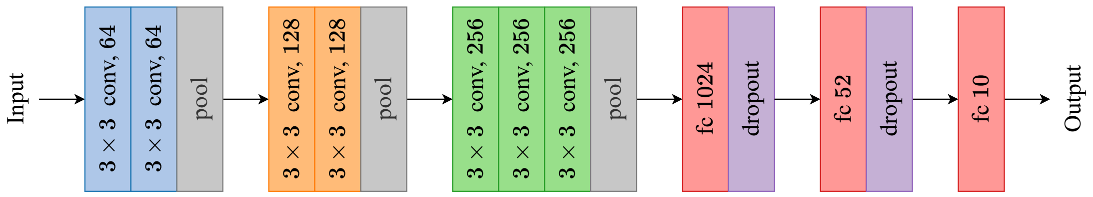


Fig. 1. Model architecture, which is a modified version of VGG.

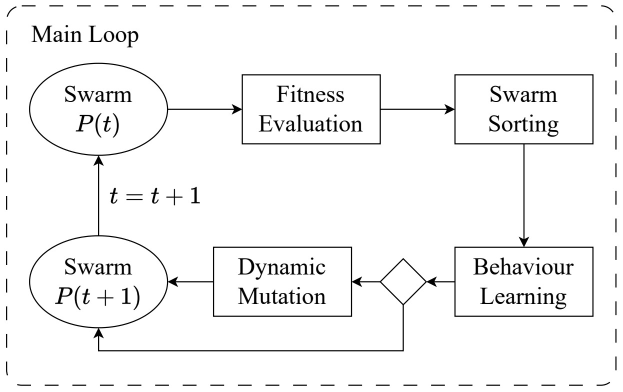


Fig. 2. Main loop for our chosen algorithm

while training. The network features dimensionality reduction throughout the convolutional layers, improving performance.

We utilise the VGG architecture as it a more parameter efficient model in comparison to architectures like ResNet which allows the model to run faster in resource-bound environments [18]. Additionally, while other architectures may be more powerful in terms of deep learning, they require more training data to be fully taken advantage of. The VGG architecture provides a strong balance between flexibility and accuracy whilst also being more efficient to train.

B. Justification

As previously stated, population-based optimisers require a significant computational overhead to run. We utilise a less complex VGG architecture with less parameters to run and compare all three optimisers in a realistic amount of time. By using a simpler architecture and only optimising the final layer, we reduce the number of resources needed to process each forward pass in our population-based optimisers as this happens several times per generation, which adds up throughout the optimisation process to potentially be millions of calculations. Additionally, the VGG architecture provides a strong foundation for the model and the optimisation process in terms of accuracy to performance overhead [19] which allows the optimiser to work at its best. To avoid overcomplicating the problem, we avoid using transformers as they would not provide enough of an accuracy benefit compared to its performance trade-off.

III. Training algorithm

A. Description

Our chosen algorithm, Social Snake Particle Swarm Optimisation (SS-PSO), extends SL-PSO and adds GA's particle mutation to promote exploration. Additionally, we also promote exploitation in the sense that if the particles have reached a potential solution, we try to mutate them in the hopes of finding a potentially better solution later. SS-PSO adds mutation based on the standard deviation of the population σ_{pop} at the end of the generation cycle, with a dynamic probability associated with it as shown in Fig. 2. Similarly, to behaviour learning, the particles that are in the worst positions have a greater likelihood of mutating than others once converged into a local minimum. We define 'worst' in this sense by calculating the Euclidean distance of each particle compared to the current best particle by measuring their losses, where the best particle has the lowest loss in the population.

By calculating σ_{pop}, we identify when the particles have converged into a single area, indicating the presence of a possible local minimum.

B. Justification of algorithm

We chose to base our algorithm off a PSO due to the number of advantages it has over traditional GAs. These include:

- Being simpler to implement than a canonical GA due to a smaller number of operators needing to be implemented. The more operators you have, the more tuning of hyper-parameters for your problem are needed and consequently makes the optimiser less flexible for different architectures and problems as a result. This makes the process more tedious in the case that a switching of network architecture is needed.
- Additionally, PSOs do not require creating or deleting existing individuals in the population and instead just requires a simple update to the particles velocity and direction, using less resources as a result.
- Importantly, PSOs have been seen to be faster at converging towards the global optimum than GAs as seen in [20] where they compared the results of the two optimisers. We also perform a quick experiment to confirm this. As you can see in out SL-PSO vs GA graph in the appendix, SL-PSO reaches a maximum accuracy faster and so it converges faster than GA.
- Extending a solution from SL-PSO allows us to enjoy the increased performance with large dimensionality prob-

lems, allowing us to focus more on smaller but substantial tweaks to hyperparameters to boost the accuracy of the optimiser in a shorter amount of time.

C. Euclidean distance

Measuring accuracy, our fitness function, tends to be more volatile, therefore we choose to calculate the Euclidean distance

$$dist(p, q) = |q - p| \quad (1)$$

of each particle's loss instead, $loss(pop_i)$ compared to the loss $loss(pop_0)$ of best loss particle pop_0 at generation t and give particles that are closer to pop_0 a greater chance of mutating than others, encouraging the particles to escape the local minimum, hopefully, leaving the previous best particle behind and finding a new loss leader.

Euclidean distance was used similarly in [21] to measure the distribution of particle positions around the global best particle. Inertia and acceleration parameters were adaptively updated depending on a particle's estimated evolutionary state.

Based on the dynamic influence of inertial weights on particles, we can pull particles out of local minima or conversely focus the search around one area to help the optimiser converge to an optimal solution quicker, enhancing the performance of classical PSO [22].

D. Mutation operator

We explore GA population mutation operators because in regular PSOs, the main operators, velocity, and position in some situations, often do not do enough to update the state of the search space, leading to premature convergence, especially in high dimensional spaces [23]. The authors added a GA mutation operator to regular PSO and proposed Chaotic PSO. They maximised the accuracy of correctly classified objects and obtained higher accuracy in comparison in a smaller number of fitness evaluations. Additionally, [24] and [25] proposed adaptive and non-adaptive mutation size methods with PSO encouraging leaders to explore more and jump out of a local minimum to lead the other particles to better positions however their methods still led to premature convergence. Finally [26] use particle distance calculations such as Euclidian distance and trackers to influence particles that haven't changed in a while to other areas by mutating by randomising several dimensions of the particle according to a given probability, essentially repositioning the particle to be outside of the local minimum.

We took this initiative and applied a different mutation technique, we gave a probability of mutation to all particles to put more encouragement on the 'jumping out' evolutionary state, where (in basic terms) a particle with smaller Euclidian distances to the best particle are given a higher mutation probability as they need the most help to escape as if they were in piled up in a ditch for example. For mutation, we chose the mutGaussian operator from the DEAP library [27] that uses a fixed hyperparameter Sigma value to control the standard deviation σ_m from the mean of a Gaussian distribution. Note that this deviation is different to the standard deviation σ_{pop}

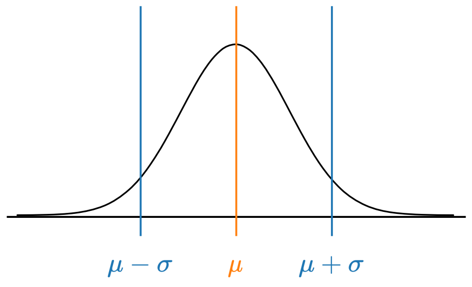


Fig. 3.

Algorithm 1: SS-PSO's addition to SL-PSO main loop

```
input : $\sigma_{pop}$, standard deviation of population's fitness
        values
        MutationStart $= 1.0$, threshold for $\sigma_{pop}$
        $\boldsymbol{pop}_0$, fittest particle in population
if $\sigma_{pop} <$ MutationStart then
    MutationTick $\leftarrow$ MutationTick $+1$
    else
        MutationTick $\leftarrow 0$
    end
end
if MutationTick $\geq 3$ then
    for $i = 1 : m$ do
        Calculate $d_i = dist(loss(pop_i), loss(pop_0))$
        Add to a data structure $\boldsymbol{d}$ containing distances
          for the whole population.
    end
    Rescale the distances in $\boldsymbol{d} = \{d_i, \ldots, d_m\}$ such
      that $\forall d \in \boldsymbol{d} : 0 \leq d \leq 1$ and the smaller the value,
      the closer it is to 1.
    for $i = 1 : m$ do
        Mutate the particle $pop_i$ with DEAP's
          mutGauss($\mu = 0$, $\sigma = 0.5$, $d_i$).
    end
end
```

of the particle's losses. If an individual pop_i mutates, we can imagine as if a random number is picked from the distribution which controls how much to mutate the individual's weights by which will in turn affect the loss value and adjusts its position in the loss space for the next evaluation, causing the particle to move around the search space. Fig. 3 demonstrates that a smaller σ_m encourages more exploitation whereas a larger σ_m encourages more exploration by diversifying the population more through a greater range of mutation [27]. The probability determines if the particle moves and if so, sigma denotes the range of how much it moves by. We believe giving a mutation probability to more than one particle increases the likelihood the swarm will explore elsewhere when necessary.

The pseudo-code for the Social Snake part of the PSO is shown in Algorithm 1.

Escaping local minima with SS-PSO is never guaranteed but the chance of converging towards the global minimum instead of local minima, as seen with SL-PSO, greatly increases with the right hyper-parameters set. However, SS-PSO requires more computational resources to run compared to SL-PSO due to needing to apply mutation to potentially many particles. For SS-PSO work at its best, an appropriate value of standard deviation to start mutating at must be chosen along with the σ_m and μ_m values of the mutation operator itself. Choosing too high of a σ_m will cause the particles to bounce around and not converge, while too small will cause the particles to not escape a local minimum.

To improve the performance of the algorithm, we use a simpler determination by just using the standard deviation to make an education guess if the particles have reached a minimum. Additionally, mutation allows for greater flexibility when implementing different architectures in comparison to other approaches [17] to perform a similar thing as there are less hyperparameters to be tuned when doing so.

We use a probability to mutate instead of mutating all the particles by a certain amount for two reasons. The most important is that we do not want the position of a particle to drastically change, instead we only want to assist it in getting out of a minimum by shifting some of the decision variables slightly. As we do not know which decision variables in a particle are already good during the algorithm, we want to give a chance that the particle will not mutate which will not affect the good decision variables, also improving performance as we do not need to mutate as many particles.

We do not mutate the current best particle to act as a checkpoint for the best fitness so far in the case that the other particles have much worse fitness after mutation or if the area was previously not a local minimum and can then move back towards that particle later if needed. Minimum loss will still change slightly due to methods used by the inherited SL-PSO methods, but it is still important to not destroy the learning of the best particle by mutating it in case mutation fails on the other particles.

IV. Results

A. Setup

The first step involves evaluating the fitness of each particle in our search space. To do this, we generate our population with the dimensionality of $n = op+b$, where b is the number of bias weights, o is the number of classes and p is the number of parameters for each output. Each particle is a different version of the final layer. To evaluate a single particle, we split it into weights and biases and place it into the final layer of the model. We then perform a forward pass of the network with a selected batch size of training images and calculate the accuracy for that particle to act as our fitness function. We also calculate the cross-entropy loss of the particle at the same time. After doing this for the population, we then sort the swarm by fitness in ascending order and then perform the same particle update as SL-PSO using a velocity equation.

A baseline model was created using Stochastic Gradient Descent. To compare the efficacy of population-based methods

TABLE I
Summary of parameters for single objective algorithms

Optimiser	SGD	SL-PSO [28]	SS-PSO
Swarm size	-	153	153
Batch size	32	512	512
Social influence	-	0.034	0.034
Dynamic Mutation	No	No	Yes
Epochs	40	-	-
Generations	-	100	100

TABLE II
Summary of results

Optimiser	SGD	SL-PSO [28]	SS-PSO
Training Accuracy	78%	50%	62%
Testing Accuracy	74%	42%	56%

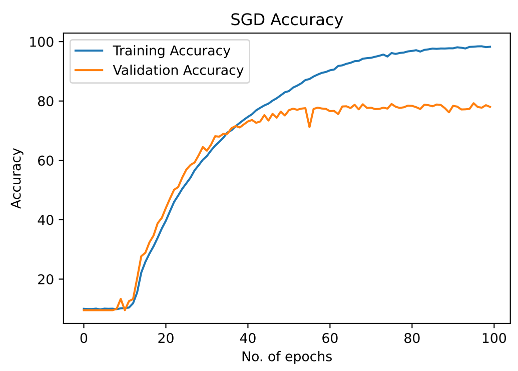
with the typical gradient based method in a reasonable amount of time for the classification of images in the CIFAR-10 dataset, fine tuning was performed on the final fully connected layer. The initial population size for the population-based optimisation algorithms is set to n. For our network, $b = 10$, $o = 10$, $p = 52$, therefore $n = 530$.

All experiments were conducted on a computer running Ubuntu 22.04 LTS with a 12-core Intel Core i7-8700 CPU 3.20GHz with 32GB of RAM and an Nvidia Quadro P4000 with 8GB of VRAM.

B. Interpretation

Firstly, we observe the training performance of SGD, acting as our benchmark. We use 100 epochs and a validation set during training to identify where the model starts overfitting. Up to 40 epochs, the optimiser reaches an accuracy score of 74% on the training set and reaches a peak accuracy of 98% by 100 epochs. We can tell from Fig. 4 that the optimiser is causing large quantities of overfitting to the training set, damaging its generalising capabilities. Following this, we retrained the model with 40 epochs and froze these weights when training the last layer to test our population-based algorithm. As we stated earlier, the VGG architecture is powerful enough to achieve good performance with less epochs [29] and thus for better generalisation to unseen data so it is enough to stop training the network after this. SGD shows very fast convergence speeds in terms of both loss (see appendix) and accuracy due to the efficiency of gradient based calculations and backpropagation.

In comparison, both the population-based optimisers behave very differently. When searching in the space of between ±1, we can see a much slower rate of convergence in loss along with slower increases in accuracy over time. SL-PSO, by generation 100, reaches a peak accuracy of 50% on the training set with an average of 46%, significantly lower than SGD. SL-PSO reaches an accuracy of 42% on our unseen test set. Additionally, we also see that the accuracy and loss both plateau early on and do not improve further identified by the small standard deviation values of the loss.



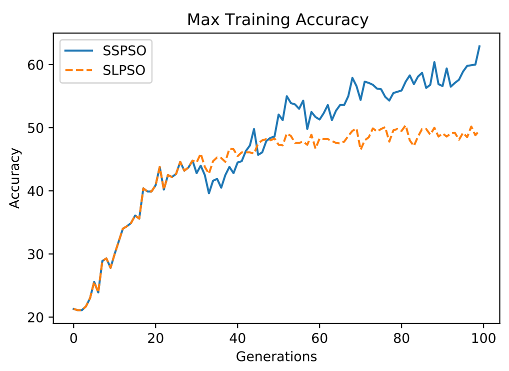


Fig. 4.

Our custom algorithm, SS-PSO, shows an expected improvement where it reaches a peak accuracy of 62% with an average of 56% on the training set, a peak increase of 12%, with a 55.5% accuracy on the test set. This is a good increase of $+13.5\%$ on the unseen data. Observing the average accuracy (see appendix) of the two algorithms we can see SS-PSO appears unstable in comparison spiking throughout this is due to the mutation of the particles, briefly causing this jump but ends up finding a new loss leader later. From 0 to 30 generations, as we set the same seed value for both optimisers, the random decisions are the same and therefore, so are the values up to this point. Beyond 30 generations, mutation kicks in and gives an increase in performance as it has escaped the local minimum as we have set the appropriate hyperparameters. SSPSO also does not appear to converge early on and continues to increase in accuracy throughout the 100 generations, potentially showing that it may increase in accuracy if we allow more than 100 generations.

While Social Learning by itself is sufficient to solve the problem and achieve a high accuracy, after many generations, the particles often fall into a local minimum which then struggles to get out of it with the standard velocity and directional updates.

It's important to understand that the σ_m and μ_m values are vital to SSPSO's ability along with choosing an appropriate `mutationStart` and `mutationTick` value. `mutationStart` is the value that standard deviation needs to go under three times in a row to trigger mutation of the population and `mutationTick` is the counter. Both can be chosen manually. Choosing too high of a `mutationStart` or too low of a `mutationTick` value will cause the particles to be mutated and shuffled prematurely, hindering its convergence, and causing its overall accuracy to suffer as a result. If σ_m is too high then the particles will be scattered more than necessary and it could take more generations to gather its bearings, harming convergence to the global optimum. An example of how performance degrades from mutation by selecting too high of a value of σ_m can be seen in Fig. 10 in Appendix B.

Additionally, due to using a simple method to determine convergence, performance may be affected as it is very difficult to tell if we have reached a local minimum or not and is also harder to tell if mutation is affecting the ability of SSPSO to converge. Runs can be more random than others without a fixed seed meaning mutation might activate more frequently than in other runs which also hinders performance. Other papers such as Chaotic PSO from before use more complicated methods to determine when to mutate and as a result, is more robust to different problems. Due to the lack of robustness, there are times where it is possible to see SL-PSO perform the same or slightly better than SS-PSO and it may be harder to set the correct hyperparameters to make mutation successful.

V. Bi-objective problem

It is common to introduce regularisation into a model to address the problem of overfitting, meaning it has learned the training data too well and doesn't generalise well on unseen data. Dropout or normalisation layers can be added to the model directly to make the model less complex. Instead of doing this we can turn regularisation into a bi-objective function and represent the optimisation problem as a bi-objective problem containing two objectives; to maximise accuracy and minimise model complexity. We use a known optimisation algorithm, Non-Dominated Sorting Genetic Algorithm (NSGA-II), to find a set of solutions, in figure 2, for the two objective functions accuracy and a chosen regulariser, sum of weights squared,

$$L2 = \lambda \sum_{i=1}^{k} w_i^2. \tag{2}$$

The set, called the Pareto front, is not dominated by any other solutions in the objective space and provides us with solutions that represents a trade-off between the conflicting objectives, accuracy and complexity.

The crowding distance of the solutions within each front are calculated so that the algorithm can sort leading with the best solution. Dominance rank as well as crowding distance is considered to sort the solutions, the best solutions have larger crowding distances to maximise the diversity, and lower dominance ranks. Then the algorithm can select using tournament selection the best solutions from the pareto front for the next population.

TABLE III
SUMMARY OF PARAMETERS FOR NSGA-II

Hyperparameters	Values
Iterations	100
Dimensions	530
Population size	100
Encoding	Real coded
Upper bound	1
Lower bound	-1
Crossover Probability	0.9

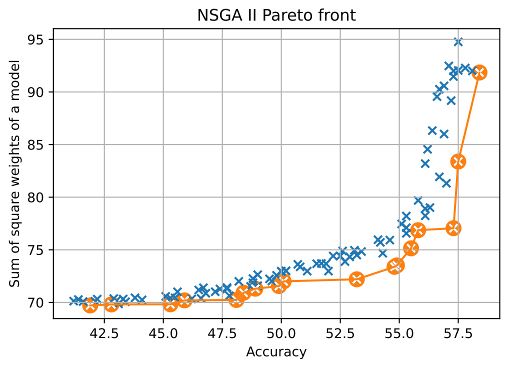


Fig. 5. Pareto plot of NSGA-II

The set-up of our bi-objective optimisation experiment is detailed in Table III. We use a population size and generation size of 100. The dimensions of the individuals are 530 and we define a crossover probability of 0.9. We use ± 1 as upper and lower bounds and Tournament selection based on dominance followed by crowding distance to find the optimal solutions in the pareto front.

Our primary objective is to maximise accuracy in the network. It is clear to see from the results of NSGA-II in Fig 6 that by adding a second objective function it does not give as high accuracy as the single objective optimiser SS-PSO. The bi-objective task is trying to balance a set of conflicting solutions, so resultingly does not return a single accuracy based optimum solution. Furthermore, weight optimisation in the case for neural networks is highly dimensional so gradient based optimisation methods generally work better and are more effective for training neural network as opposed to multi-objective optimisation which struggles more on this domain of problems. Our accuracy results from SGD outperformed NSGA-II, therefore we should use a single objective optimiser such as SGD to find this and monitor regularisation in the network layers rather than in the optimiser itself.

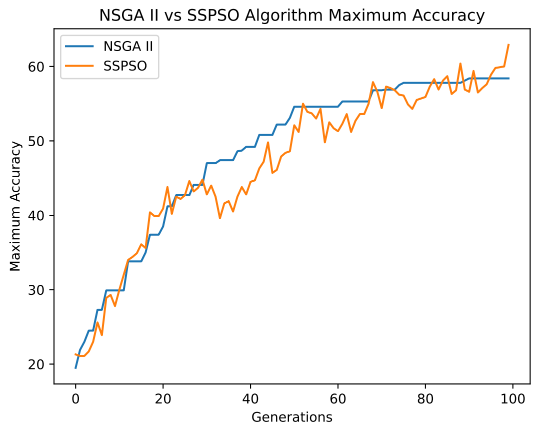


Fig. 6.

Research papers suggest that other single optimisers such as population based are also better suited for this problem so should also perform better than NSGA-II. Therefore, we expected NSGA-II to perform worse than our population-based optimiser. Research in [30] suggest regularisation can be effectively applied to the network during training whilst still achieving high generalisation performance. Overfitting is important but it can be incorporated into the network more efficiently during training using methods such as dropout layers or image augmentation. In [31] the authors also find single-optimisation methods outperform multi-objective methods in terms of accuracy and convergence speed.

Interestingly, throughout the generations, our NSGA-II algorithm appears to match the performance of the single objective population optimisers. While this is not what we expected there is reason to believe that there are scenarios where this may happen. The exploration of different solutions due to the balancing of two objectives may act similar to how SSPSO attempts to escape a local minima and as a result, the NSGA algorithm is more likely to find the optimal set faster. Similarly, utilising the crowding distance allows for preservation of diversity which again reduces the chance of a converging prematurely. Multi-objective optimisers can be utilised for single-objective problems and can surpass their performance when given the right circumstances [32].

VI. CONCLUSION

The final result of each algorithm shows that SGD as a whole is better for most neural network training scenarios due to its flexibility along with great convergence performance relative to the population-based algorithms. As said earlier, while this is enough for most scenarios, gradient based optimisers do not work when the objective is not a continuous function and is not differentiable, stopping backpropagation from working. In more complicated networks, the issue of vanishing and exploding gradients appear. In these scenarios, it may be more viable to use a population-based algorithm such as SL-PSO which can tackle the problem at the cost of convergence speed and overall accuracy [26]. SS-PSO attempts to improve on this by attempting promote more exploration towards to the global minimum. When using mutation in this way, it is important that the hyperparameters are correctly set to avoid harming the convergence of the algorithm and requires careful consideration. SS-PSO shows a proof of concept into attempting to escape local minima.

Considering that the main objective of an optimiser is to maximise the accuracy. We can conclude that for the majority of scenarios, the single objective optimisers are better suited to the problem and a more optimal set of weights will be obtained. There is also a performance trade-off between single and bi-objective optimisers, since the bi-objective is balancing multiple objectives it is more computationally expensive and takes longer to run. If a multi-objective optimiser is given the right circumstances and is tuned correctly, there are situations where it may be more beneficial to use than a single objective population-based optimiser, beating its performance. This is a situational however and is often a difficult task to do purposefully and should be considered anomalous.

Due to the lack of time, in the future we hope to extend our work by conducting more experiments to improve the performance of SS-PSO. Currently, the sigma and mu values of the mutation operator are static and may be tedious to set correctly. To combat this, we could consider dynamically adjusting these values similarly to how we dynamically calculate probability. To address the robustness of the algorithm, we could use a more complicated and powerful method of determining convergence compared to the simple method we currently use, in an effort to consistently beat SL-PSO in more cases, and make the outcome less random.

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[2] N. Sharma, V. Jain, and A. Mishra, "An analysis of convolutional neural networks for image classification," *Procedia Computer Science*, vol. 132, pp. 377–384, 2018.

[3] J. Wei, "AlexNet: The Architecture that Challenged CNNs," https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951, Jul. 2019.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012.

[5] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," 2020.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[9] A. Khan, Z. Rauf, A. Sohail, A. R. Khan, H. Asif, A. Asif, and U. Farooq, "A survey of the vision transformers and their CNN-transformer based variants," *Artificial Intelligence Review*, vol. 56, no. 3, pp. 2917–2970, Dec. 2023.

[10] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," University of Toronto, Tech. Rep., 2009.

[11] L. Deininger, B. Stimpel, A. Yuce, S. Abbasi-Sureshjani, S. Schönenberger, P. Ocampo, K. Korski, and F. Gaire, "A comparative study between vision transformers and CNNs in digital pathology," 2022.

[12] Y. Bohra, "The Challenge of Vanishing/Exploding Gradients in Deep Neural Networks," https://www.analyticsvidhya.com/blog/2021/06/the-challenge-of-vanishing-exploding-gradients-in-deep-neural-networks/, Jun. 2021.

[13] D. Xu, Y. Li, X. Tang, Y. Pang, and Y. Liao, "Adaptive particle swarm optimization with mutation," in *Proceedings of the 30th Chinese Control Conference*, 2011, pp. 2044–2049.

[14] L. Bliek, A. Guijt, R. Karlsson, S. Verwer, and M. de Weerdt, "Benchmarking surrogate-based optimisation algorithms on expensive black-box functions," *Applied Soft Computing*, vol. 147, p. 110744, 2023.

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

[16] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 1995, pp. 39–43.

[17] V. Steffen, "Particle swarm optimization with a simplex strategy to avoid getting stuck on local optimum," *AI, Computer Science and Robotics Technology*, Oct. 2022.

[18] A. Anwar, "Difference between AlexNet, VGGNet, ResNet, and Inception," https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96, Jun. 2019.

[19] A.-C. Mihai and D.-T. Iancu, "Optimizing a convolutional neural network using particle swarm optimization," in *2022 14th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, 2022, pp. 1–7.

[20] Y. Ding, W. Zhang, L. Yu, and K. Lu, "The accuracy and efficiency of GA and PSO optimization schemes on estimating reaction kinetic parameters of biomass pyrolysis," *Energy*, vol. 176, pp. 582–588, 2019.

[21] Z.-H. Zhan, J. Zhang, Y. Li, and H. S.-H. Chung, "Adaptive particle swarm optimization," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 6, pp. 1362–1381, 2009.

[22] J. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[23] Z. Assarzadeh and A. R. Naghsh-Nilchi, "Chaotic particle swarm optimization with mutation for classification." *J Med Signals Sens*, vol. 5, no. 1, pp. 12–20, 2015 Jan-Mar.

[24] J. Tang and X. Zhao, "Particle swarm optimization with adaptive mutation," in *2009 WASE International Conference on Information Engineering*, vol. 2, 2009, pp. 234–237.

[25] J. Chen, Z. Ren, and X. Fan, "Particle swarm optimization with adaptive mutation and its application research in tuning of PID parameters," in *2006 1st International Symposium on Systems and Control in Aerospace and Astronautics*, 2006, pp. 5 pp.–994.

[26] N. Li, Y.-Q. Qin, D.-B. Sun, and T. Zou, "Particle swarm optimization with mutation operator," in *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826)*, vol. 4, 2004, pp. 2251–2256 vol.4.

[27] DEAP Project, "Evolutionary Tools - DEAP 1.4.1 documentation," https://deap.readthedocs.io/en/master/api/tools.html, Jul. 2023.

[28] R. Cheng and Y. Jin, "A social learning particle swarm optimization algorithm for scalable optimization," *Information Sciences*, vol. 291, pp. 43–60, 2015.

[29] X. Jin, X. Du, and H. Sun, "VGG-S: Improved small sample image recognition model based on VGG16," in *2021 3rd International Conference on Artificial Intelligence and Advanced Manufacture (AIAM)*, 2021, pp. 229–232.

[30] Y. Tian and Y. Zhang, "A comprehensive survey on regularization strategies in machine learning," *Information Fusion*, vol. 80, pp. 146–166, 2022.

[31] T. G. Tan, J. Teo, and K. O. Chin, "Single- versus Multiobjective Optimization for Evolution of Neural Controllers in Ms. Pac-Man," *International Journal of Computer Games Technology*, vol. 2013, p. 170914, Feb. 2013.

[32] M. Mahrach, G. Miranda, C. León, and E. Segredo, "Comparison between single and multi-objective evolutionary algorithms to solve the knapsack problem and the travelling salesman problem," *Mathematics*, vol. 8, no. 2018, 2020.

Appendix

A. Contributions

Each member contributed equally to all project components.

B. Plots

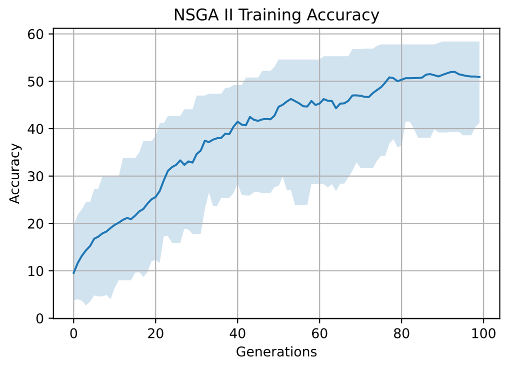


Fig. 7. Training average, minimum and maximum for NSGA-II.

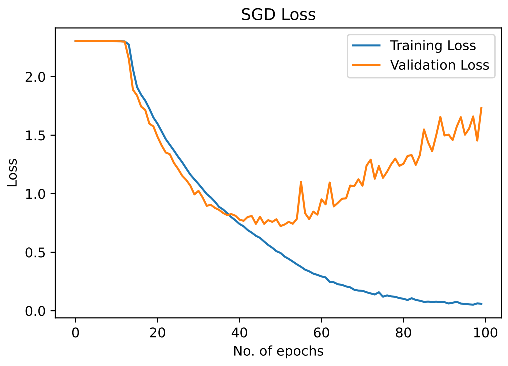


Fig. 8. Loss plot for training and validation of baseline SGD model.

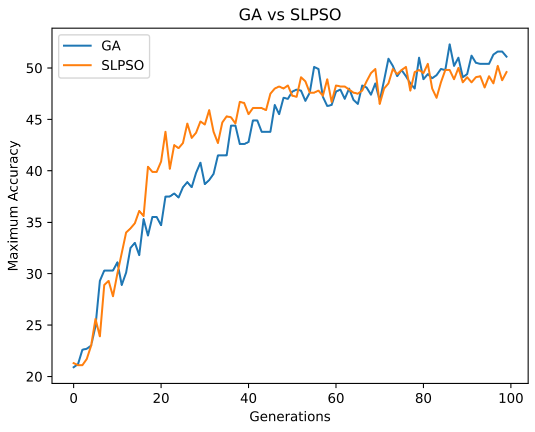


Fig. 9. Maximum training accuracy for GA and SL-PSO.

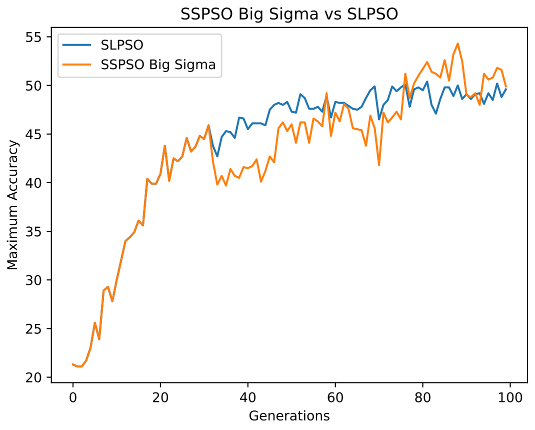


Fig. 10. Comparing SS-PSO and a large value of σ_m with SL-PSO.

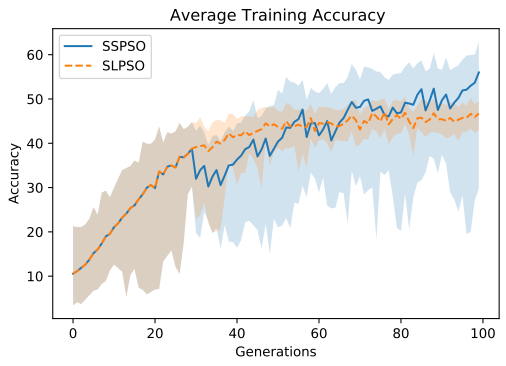


Fig. 11. Average training accuracy, including minimum and maximum values, for SS-PSO and SL-PSO.