

Optimising Convolutional Networks with Metaheuristic Strategies on CIFAR-10

Danish Whelan Bin Zamri
URN: 6944467

Eleana Vrachimi
URN: 6941002

Elpida Dimitriou
URN: 6910145

Mohammad Shayann Aslam Khan
URN: 6941885

Abstract— This work examines how neural networks for image classification can be optimized when certain population-based techniques are applied, using the CIFAR-10 dataset. We adopt a Resnet-9 architecture and initially train the convolutional neural network (CNN) using the Adam optimizer, a standard gradient-based method to establish a baseline. All feature-extraction layers except for the final fully connected layer are frozen, and the last layer is then re-initialized, so that its parameters can be optimized through different population-driven techniques. Specifically, we investigate Differential Evolution (DE), Comprehensive Learning Particle Swarm Optimization (CLPSO) and self-adaptive differential evolution (SADE), comparing their convergence behaviour, computational overhead, and classification accuracy relative to the gradient-based baseline. For completeness, standard Particle Swarm Optimisation (PSO) is also included to enable direct comparison with its advanced variant (CLPSO) and the other algorithms as well. Additionally, a bi-objective formulation of the optimisation problem is explored using NSGA-II, enabling a detailed analysis of the trade-off between accuracy and Gaussian weight regularisation. Through a set of computational experiments, quantitative comparisons, learning curves and Pareto-front visualization, this work investigates the strengths, limitations and trade-offs between gradient-based and population-based strategies to neural network optimization.

I. INTRODUCTION

In Machine Learning, image classification is a fundamental problem that motivates the implementation and optimisation of a model that, based on a given dataset of images and predefined classes, assigns an input image to one of these classes. We particularly focus on the CIFAR-10 dataset, a widely used benchmark introduced by Krizhevsky [1]. It consists of 60,000 coloured images of size 32×32 across 10 categories, split into 40,000 training, 10,000 validation, and 10,000 test examples.

The development of convolutional networks dates back to early work by Fukushima [2] and then by LeCun et al. [3] on digit recognition. The release of AlexNet [4] sparked a renewed interest in deep CNNs. This showed that deeper architectures trained on GPUs might significantly

outperform earlier methods. This achievement made it possible for recent CNN families like VGGNet [5], to focus on hierarchical feature extraction and very deep layer structures.

Residual networks (ResNets), a central example of modern CNNs, have become particularly influential. He et al. [6] introduced deep residual learning, where layers learn residual functions added to their input via identity shortcuts, making very deep networks easier to optimise and improving accuracy on benchmarks such as ImageNet and CIFAR-10. Compact residual variants can still perform well while keeping model size and training cost low. For instance, CoRN [7] achieves competitive test errors on CIFAR-10 and CIFAR-100, while Awad and colleagues [8] showed that a compact ResNet-9 trained on only 10% of CIFAR-10 can still reach about 88% accuracy. These results further support the effectiveness of small residual architectures, such as ResNet-9, to give a good balance between accuracy and computational cost for CIFAR-10.

Training CNNs is often accomplished using gradient-based optimisation. More specifically, adaptive methods like Adam introduced by Kingma and Ba [9] adjust each parameter's learning rate based on estimates of first and second moments of the gradients, while simpler approaches such as stochastic gradient descent (SGD) update parameters using mini-batch gradient estimates [10]. However, such techniques can be sensitive to hyper-parameter choices and may get trapped in poor local minima or plateaus, especially in highly non-convex loss landscapes. Therefore, achieving consistently high performance across architectures and training conditions can still be challenging.

In parallel, population-based optimisation methods have been explored to assist or replace gradient-based training. Particle Swarm Optimisation (PSO) [11] and Differential Evolution (DE) [12] are widely used approaches for continuous global optimisation, with several variants proposed to improve convergence and robustness, including self-adaptive Differential Evolution (SaDE) [13] and Comprehensive Learning Particle Swarm Optimisation (CLPSO) [14]. Surveys on evolutionary deep learning and neuroevolution [15], [16] demonstrate how PSO and DE have been used to optimise network weights, architectures and hyperparameters but also note that fully evolving all parameters of a modern CNN is extremely expensive. This makes pure evolutionary training difficult to scale and

motivates hybrid approaches, where evolutionary search is applied only to selected parts.

Neural network learning can also be treated as a multi-objective problem, balancing accuracy against model complexity or weight magnitude. Jin and co-authors [17] used multi-objective algorithms to regularise neural networks, trading off approximation error against measures of network complexity, and creating a set of Pareto-optimal models. NSGA-II, introduced by Deb et al. [18], is a fast and elitist example of such algorithms, but adopting this approach at deep-learning scale is computationally demanding and still leaves open the question of how to choose and interpret appropriate trade-offs between accuracy and regularization.

Other approaches that have been suggested to improve deep models include neural architecture search [19], meta-learning [20], and second-order optimisation methods [21]. Nevertheless, it is still difficult to apply them in practice in large, complex networks. Second-order methods include expensive curvature calculations, and both NAS and meta-learning typically require very high computational resources, which limits their real-world use. Consequently, finding techniques that reduce training cost without sacrificing performance remains an unresolved challenge in deep learning.

Hence, in this work, we initially train the full ResNet-9 model on CIFAR-10 using Adam as a baseline gradient-based optimiser and then freeze all feature-extraction layers. At that point, we re-initialise the last layer and considered its weights and biases as the decision variables of an optimisation problem. Differential Evolution (DE), Comprehensive Learning Particle Swarm Optimisation (CLPSO) and self-adaptive DE (SaDE) are used to re-optimize this last layer, and their performance is compared against the Adam baseline. Standard Particle Swarm Optimisation (PSO) is also evaluated as a baseline population-based approach for contextual comparison. Finally, we consider a bi-objective version of the last-layer training problem, aiming to maximise accuracy and simultaneously minimise the sum of squared weights (Gaussian regularisation), using NSGA-II to approximate the corresponding Pareto front. Overall, our contribution is to provide a controlled comparison between Adam and these population-based algorithms on the same ResNet-9 architecture for CIFAR-10, along with an exploration of a bi-objective optimization via NSGA-II.

II. ARCHITECTURE

Overview

A customised ResNet-9 architecture inspired by the standard ResNet-9 was implemented and tailored for the CIFAR-10 dataset [6]. The network processes 32×32 RGB images and uses residual connections to improve gradient flow and enable more efficient training of deeper feature representations. The design increases model depth gradually, while reducing spatial resolution. This helps the network to learn high-level characteristics without being extremely

computationally expensive.

Initial feature extraction

Initially, the network starts with a 3×3 convolution with stride 1 and padding 1, which expands the channels from 3 to 64 and then Batch Normalisation and a ReLU activation are applied. Using small 3×3 kernels maintains local spatial structure, enabling the network's learning of basic edge and texture patterns suitable for small images, such as CIFAR-10 [5]. Batch Normalisation is applied to keep training stable through normalisation within each mini batch [22], while ReLU is the non-linear activation required to model complex decision boundaries [23].

Downsampling blocks

Following the first stage, the model is composed of three residual blocks that gradually reduce spatial resolution while increasing the number of channels. In each residual block, the first convolution is followed by batch normalisation and a ReLU activation, while the second convolution is followed by batch normalisation. The output of the main and shortcut paths is then combined through residual addition and passed through a final ReLU. Batch normalization is used to improve training stability and decrease sensitivity to parameter initialisation, while ReLU activations introduce nonlinearity and boost representational capacity [6].

The first residual block increases the channels from 64 to 128, while a 3×3 convolution with stride 2 downsizes the images from 32×32 to 16×16 . A second 3×3 convolution with stride 1 enhances feature representations, and a parallel shortcut path uses a 1×1 convolution with stride 2 to match the change in channel dimensions before residual addition [6]. This residual connection allows feature reuse across layers while mitigating the vanishing gradient issue. The following blocks use the same structure.

The second block expands the number of channels from 128 to 256 and reduces the spatial resolution from 16×16 to 8×8 . The final block further increases channel dimension from 256 to 512, while spatial resolution declines from 8×8 to 4×4 .

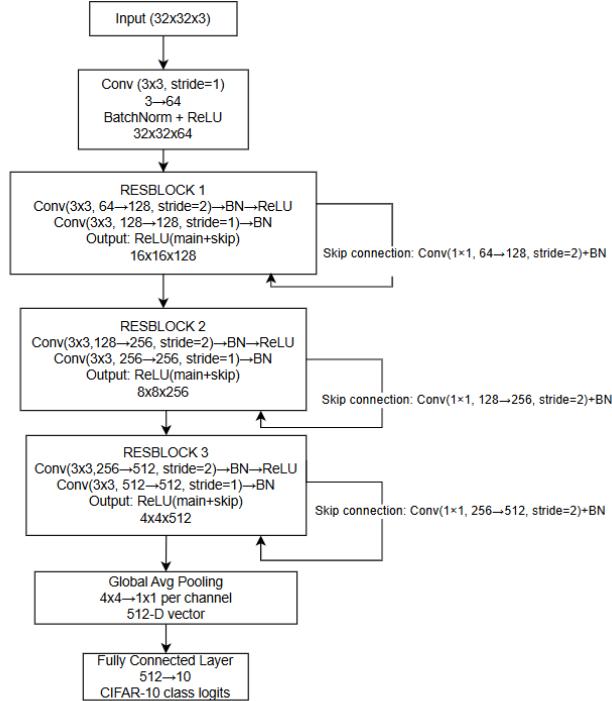
Classification Layer

Following the last convolutional block, a global average pooling operation with a kernel size of 4 averages each 4×4 feature map into a single value, resulting in a 512-dimensional feature vector per image. This vector is passed into a fully connected layer with 10 output units corresponding to the CIFAR-10 classes. The use of global average pooling acts as a form of regularisation by reducing the number of trainable parameters [24]. The network contains approximately 4.8 million trainable parameters, with 5,130 belonging to the final classification layer.

Design Justification

This ResNet-9 configuration balances model capacity and computational cost for CIFAR-10. Using residual BasicBlocks demonstrates strong performance on feature extraction with relatively modest depth [6], [5], [22]. Meanwhile, keeping the classifier shallow isolates a small

set of weights (5130 parameters) that can be re-initialised and optimised with population-based algorithms while the convolutional backbone remains fixed. This makes the optimisation problem computationally manageable, as evolving a full deep network is known to be expensive [25], and is consistent with using deep features as fixed representations with simple classification heads [26].



Flowchart: Resnet Architecture

III. TRAINING ALGORITHMS

In this section, we discuss the two main optimization algorithms chosen for training the final layer of our ResNet-9 architecture: Self-Adaptive Differential Evolution (SADE) and Comprehensive Learning Particle Swarm Optimization (CLPSO). These algorithms were chosen to enhance the baseline methods (gradient descent, standard PSO, and standard DE), as they introduce adaptive features and different search strategies that might help in exploring the complex weight space of neural networks.

A. Comprehensive Learning Particle Swarm Optimisation (CLPSO)

CLPSO, introduced by Liang et al. [14], is an advanced variant of standard Particle Swarm Optimisation specifically designed for multimodal and high-dimensional optimization problems. Unlike standard PSO, where all dimensions learn from the same global best particle, CLPSO employs a comprehensive learning strategy, where each dimension of a particle's velocity can potentially learn from different particles' historical best positions.

Given the high dimensionality of the optimisation problem (5,130 parameters), CLPSO is an appropriate choice for this

work. Its dimension-wise learning mechanism helps to reduce stagnation behavior, which is frequent in standard PSO when dealing with high-dimensional problems. The particle-dependent learning probability provides an effective balance between exploration and exploitation, while the refreshing strategy maintains population diversity and prevents premature convergence. The algorithm has demonstrated superior performance on multimodal functions compared to standard PSO and other variants [14]. In addition, CLPSO has been effectively used for deep learning and image-based tasks. Dang et al. [27] utilised CLPSO to optimise ensemble weights of deep neural networks for medical image segmentation, proving its effectiveness when paired with deep models on real-world images.

Algorithm Description

The key innovation of CLPSO lies in its dimension-wise learning mechanism. For each dimension d of particle i , an exemplar particle $f_i(d)$ is selected via tournament selection with size $k = 2$. That means two candidate particles are compared, and the best performing one is chosen as the exemplar. The probability that a particle learns from others rather than its own personal best is determined by the learning probability P_c , which varies across particles based on their rank in the population. Better-performing particles have lower P_c values, allowing them to exploit their own solutions, while worse-performing particles have higher P_c values, encouraging more exploration.

The velocity update equation for CLPSO is:

$$v_{i,d} = \omega \cdot v_{i,d} + c \cdot r \cdot (pbest_{f_i(d),d} - x_{i,d})$$

where ω is the inertia weight (decreasing linearly from 0.9 to 0.3), c is the learning coefficient equal to 1.5, r is a uniformly distributed random number in $[0,1]$, $pbest_{f_i(d),d}$ is the personal best position of the exemplar particle for dimension d and $x_{i,d}$ is the current position of particle i . A critical component is the refreshing mechanism: if a particle's personal best does not improve for m generations (refreshing gap = 5), new exemplars are selected for all dimensions. This prevents premature convergence while maintaining search diversity.

Pseudocode for CLPSO

Input: Population size $N = 50$, Generations $G = 500$, Dimensions $D = 5130$, Learning coefficient $c = 1.5$, Refreshing gap $m = 5$, Learning probability range $[P_{c_min}, P_{c_max}]$

Output: Best weight vector w^*

Initialisation:

For $i=1$ to N do:

$$\begin{aligned} x_{-i} &\sim U(-0.5, 0.5) \\ v_{-i} &\sim U(-0.1, 0.1) \\ pbest_{-i} &\leftarrow x_{-i} \\ stagnation_{-i} &\leftarrow 0 \end{aligned}$$

End for

Evaluate $f(x_{-i})$ for all i

$P_c(k) <- P_{c_min} + (P_{c_max} - P_{c_min}) * k / (N - 1)$, for $k = 0, \dots, N-1$
 Select $f_i(d)$ via tournament selection for all i, d

For $g = 1$ to G do:

$w <- 0.9 - 0.6 * (g/G)$

Rank particles by $f(pbest_i)$ (best → worst)

For $i = 1$ to N do:

For $d = 1$ to D do:

$exemplar_d <- pbest\{f_i(d), d\}$

End for

$v_i <- w * v_i + c * rand() * (exemplar - x_i)$

$v_i <- clip(v_i \text{ to } [-0.6, 0.6])$

$x_i <- x_i + v_i$

Evaluate $f(x_i)$

If $f(x_i) > f(pbest_i)$:

$pbest_i <- x_i$; $stagnation_i <- 0$

Else:

$stagnation_i <- stagnation_i + 1$

End if

If $stagnation_i \geq m$:

Obtain rank(i)

$P_c(i) <- P_c(rank(i))$

For $d=1$ to D do:

If $rand() < P_c(i)$:

$f_i(d) \leftarrow$ Tournament selection

Else:

$f_i(d) <- i$

End if

End for

$stagnation_i <- 0$

End if

End for

End for

Return best $pbest_i$ (Highest fitness)

B. Self-Adaptive Differential Evolution (SADE)

Self-Adaptive Differential Evolution (SADE) was proposed by Qin and Suganthan [28] as an extension of the standard DE [12] by automatically adapting both the mutation strategy and parameters during the evolution process, eliminating the need for extensive parameter tuning.

Self-adaptive DE has also been applied to neural networks. Cao et al. [29] suggested Self-adaptive Evolutionary Extreme Learning Machine (SaE-ELM), where a self-adaptive DE is used to optimise the input weights and hidden biases of a single-hidden-layer network. Better prediction accuracy was observed compared to both fixed-parameter DE and standard ELM on several regression and classification tasks. Ku et al. [30] introduced a SaDE-ELM variant that further improves generalisation by using SaDE to tune the network parameters before solving the output layer in closed form. Additionally, Baioletti et al. [31] designed a deep-network optimiser based

on a self-adaptive DE variant called MAB-ShaDE, applying DE operators in a per-layer way, showing that this approach can train large feed-forward networks. These studies support the use of SaDE-type algorithms for neural network weight optimisation, particularly when gradients are noisy or difficult to make use of.

Consequently, we selected SaDE as an optimisation method, as it reduces the need for manual problem-specific tuning of the DE control parameters. This self-adaptive algorithm automatically adjusts F and CR control parameters for each individual during the run, based on whether its trial solutions improve fitness. This is particularly valuable for neural network optimization where the loss landscape can vary significantly across different regions of the weight space. Furthermore, this kind of parameter adaptation helps the algorithm balance exploration and exploitation over successive generations, encouraging larger moves when searching broadly and more local refinements as promising regions are found. Research has shown that self-adaptive DE variants such as SaDE can outperform fixed-parameter DE on various benchmark functions [28].

Algorithm Description

The main operations of DE remain in SADE: mutation, crossover, and selection. For the rand/1/bin strategy employed in our implementation, the mutation creates a donor vector by combining three distinct randomly selected population members:

$$v_i = x_a + F \cdot (x_b - x_c)$$

Where x_a, x_b, x_c are taken from the current population and F is a differential weight sampled from a discrete pool. After mutation, the donor vector is clipped to the range $[-0.5, 0.5]$ to keep the last-layer weights within reasonable bounds. Then, the binomial crossover creates a trial vector u_i by mixing the donor v_i with the target x_i dimension by dimension, with the crossover rate CR determining the probability that each dimension is taken from the donor rather than the parent.

The self-adaptive mechanism maintains pools of candidate F values $\{0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ and CR values $\{0.1, 0.2, \dots, 0.9\}$. Initially, parameters are sampled uniformly from these pools for each individual. As evolution progresses, the algorithm tracks which F and CR values lead to successful offspring (trial vectors that replace their parents), and these successful parameter values are stored in a memory over around the last 50 generations. Parameter selection probabilities are then updated to favour values that have produced successful trials before, so that values with a stronger success history are more likely to be sampled in next generations.

Pseudocode for SADE

Input: Population size $N = 30$, Generations $G = 500$, Dimensions $D = 5130$, $F_pool = \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$, $CR_pool = \{0.1, \dots, 0.9\}$, $LP = 50$

Output: Optimized weight vector w^*

```

Initialize:  $x_i \sim U(-0.5, 0.5)$  for  $i = 1, \dots, N$ 
Initialize:  $f(X_i) = \text{accuracy}(X_i)$  for all individuals
Initialize  $\text{success\_memory\_F} \leftarrow \emptyset$ ,  $\text{success\_memory\_CR} \leftarrow \emptyset$ 
Initialize  $P_F, P_{CR}$  as uniform distributions
For  $g = 1$  to  $G$  do
    Update  $P_F, P_{CR}$  based on  $\text{success\_memory}$ 
     $\text{Gen\_success\_F} \leftarrow \emptyset$ ,  $\text{gen\_success\_CR} \leftarrow \emptyset$ 
    For  $i = 1$  to  $N$  do
        Sample  $F \sim P_F$ ,  $CR \sim P_{CR}$ 
        Select distinct indices  $a, b, c \neq i$  randomly
         $V_i \leftarrow X_a + F \times (X_b - X_c)$ 
         $V_i \leftarrow \text{clip}(V_i, -0.5, 0.5)$ 
        For  $d = 1$  to  $D$  do
            If  $\text{rand}() < CR$  then  $U_{i,d} \leftarrow V_{i,d}$  else  $U_{i,d} \leftarrow X_{i,d}$ 
        End for
        If  $f(U_i) > f(X_i)$  then
             $X_i \leftarrow I_i$ 
            Append  $F$  to  $\text{gen\_success\_F}$ 
            Append  $CR$  to  $\text{gen\_success\_CR}$ 
        End if
    End for
    Update  $\text{success\_memory}$  (keep last  $LP \times N$  entries)
End for
Return best individual (highest fitness)

```

IV. RESULTS

A. Experimental Setup

All experiments were conducted on a system with CUDA-enabled GPU using PyTorch 2.5.1. The CIFAR-10 dataset [4] was split into training (40,000 images), validation (10,000 images), and test (10,000 images) sets. Data augmentation including random horizontal flips and random crops with padding was applied to the training set. Images were normalized using mean and standard deviation of 0.5 for each RGB channel.

The ResNet-9 architecture [6] was first trained end-to-end using the Adam optimizer (learning rate = 0.001, weight decay = 10^{-4}) with a StepLR scheduler (step size = 5, gamma = 0.5) for 50 epochs, achieving a baseline test accuracy of 89.76%. Following this, all layers except the final fully connected layer were frozen, and the last layer's weights (5,130 parameters: 512×10 weights + 10 biases) were randomized for comparison of optimization methods.

For fair comparison, all optimization methods were run for 500 generations (or equivalent computational budget) with fitness evaluated on 20 batches (1,280 images) from the validation set. The final test accuracy was computed on the full test set of 10,000 images. The weight initialization bounds were set to $[-0.5, 0.5]$ for all population-based methods. Table 1 summarizes the key parameters for each algorithm.

Table 1: Algorithm Parameters

Algorithm	Pop. Size	Generations	Key Params	Evaluations
Gradient Descent (Adam)	1	500	lr=0.001	25,000
Standard DE	40	500	$F \in [0.4, 0.9]$	20,000
Standard PSO	40	500	$c_1=c_2=2.0$	20,000
CLPSO	50	500	$c=1.5, m=5$	25,000
SADE	30	500	LP=50	15,000

B. Accuracy Evaluation

Table 2: Performance Comparison Of Optimization Methods

Method	Best Val. Acc.	Test Accuracy	Best Gen.
Gradient Descent (Adam)	90.86%	89.59%	387
Differential Evolution	78.67%	77.93%	457
Standard PSO	52.73%	49.60%	445
CLPSO (Chosen 1)	88.44%	85.69%	490
SADE (Chosen 2)	84.38%	82.89%	467

C. Discussion Of Results

The experimental results reveal several important insights about the relative strengths and limitations of different optimization approaches for neural network weight optimization.

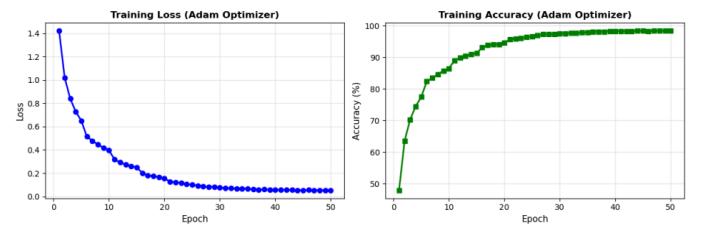


Figure 1: Baseline Training

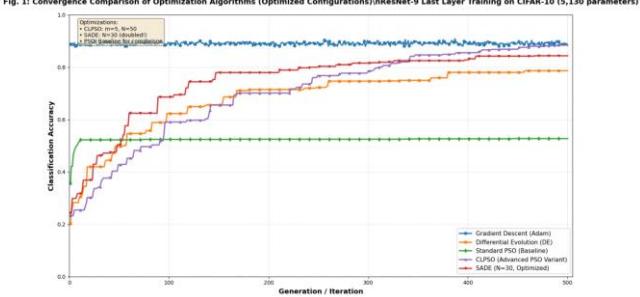


Figure 2: Performance of all Algorithms

Table 2 and Figure 2 summarise the performance of all optimisation methods on the last layer of ResNet-9. Training the re-initialised classification layer with Adam (gradient descent baseline) gives the highest test accuracy at 89.59%, and a best validation accuracy of roughly 91% confirming that exploiting gradient information is very effective once the convolutional backbone has been frozen. Among the population-based methods, CLPSO achieves the best result with a final test accuracy of 85.69%, followed by SADE at 82.89% and standard DE at 77.93%. A basic PSO run, included mainly as a reference for CLPSO, reaches only 49.60% test accuracy and saturates at a best validation accuracy of 52.73%, showing that the plain swarm dynamics struggle in the 5,130-D weight space. Thus, in this setting, the hierarchy of final performance is Adam > CLPSO > SADE > DE (>> PSO), with CLPSO and SADE (the advanced metaheuristics) narrowing a significant part of the gap to gradient descent baseline.

The convergence curves explain how CLPSO and SaDE achieve these results. CLPSO, with a population of 50 particles, combines a linearly decaying inertia weight with dimension-wise exemplar learning and a refreshing gap of $m=5$. Different coordinates of each particle can learn from different exemplars, and when a particle's personal best does not improve for several generations, all exemplars are resampled. This prevents the swarm from collapsing too early around a single attractor. In the logs, CLPSO continues to make gradual progress across the whole run, achieving its best validation accuracy of 88.44% at generation 490 and eventually outperforming standard PSO by around 36 percentage points on the test set. SaDE, using a smaller population of 30 individuals, adapts its mutation factor and crossover rate by sampling F and CR from pools whose probabilities are updated from a 50-generation success history. Standard DE (population 40) flattens around a best validation accuracy of 78.67%, whereas SaDE keeps improving after DE has stabilised, reaching 84.38% on the validation subset and a higher test accuracy despite fewer individuals. This suggests that dimension-wise exemplar learning (CLPSO) and success-history parameter adaptation (SaDE) are both useful in this rugged, high-dimensional last-layer landscape. CLPSO finishes as the best-performing population-based method despite SADE climbing faster than it for roughly the first 330 generations, as it then begins to plateau while CLPSO keeps improving.

Standard DE had a superior result of 77.93% compared to 49.60% that standard PSO produced. The distinction between the two can be attributed to the different mutation mechanisms that standard DE uses to maintain the population's diversity via multiple vectors and continue to provide targeted exploration compared to PSO, which uses velocity to transport particles. When the offspring of differential mutation are created, they tend to have strong correlation-based relationships with existing good solutions, whereas particle movement of PSO is random and frequently erratic within high-dimensional spaces. The standard poor performance of standard PSO (49.60%) evidences the "curse of dimensionality" as cited in swarm intelligence algorithms where there are too many dimensions to be attracted to one global best. For example, when in a search space of 5,130 dimensions, particles seeking out one best solution will tend to converge into one point of the dimension space.

From a computational point of view, Adam remains the most effective method for training the last layer when gradients are available. It exploits local curvature information through its adaptive learning rates, enabling efficient navigation of the loss landscape. The early and steep convergence reflects Adam's ability to make large, informed steps in directions that reduce loss. The population-based methods require many more objective evaluations, but they have different advantages. For instance, they do not rely on back-propagation, they can be applied to non-smooth or highly multimodal objectives, and their evaluations can be parallelised across individuals. In our implementation, DE and SADE use CPU-level parallelism to evaluate different candidates at once, which helps to offset their higher evaluation budget. Overall, the experiments indicate that gradient descent is still the strongest baseline for this task, but well-designed metaheuristics such as CLPSO and SaDE can achieve competitive accuracy on the frozen-backbone CIFAR-10 problem and provide a flexible gradient-free alternative when derivative information is unreliable or unavailable.

V. BI-OBJECTIVE OPTIMISATION WITH NSGA-II

A. Formulating the problem

Optimizing the last layer of the model allows the exploration of the trade-off between accuracy and model complexity. Therefore, we treat this as a bi-objective optimisation problem and use NSGA-II to train the model, considering two objectives simultaneously:

1. Maximise classification accuracy
 $f_1(w) = \text{accuracy}(w)$
2. Minimise weight regularization
 $f_2(w) = \sum_i w_i^2$ (Gaussian regulariser/L2 penalty)

The Gaussian regularisation (L2 penalty) penalizes large weights, which leads to better generalization and reduces overfitting (smoother decision boundaries). In contrast to the scalarised single-objective loss functions, this formulation avoids manually choosing a regularisation coefficient λ .

B. Implementing NSGA-II

Each individual encoded a real-valued weight vector assigned to the model's last layer and fitness evolved via NSGA-II (Non-dominated Sorting Genetic Algorithm II) [18], which was implemented using the DEAP library with the following parameters:

- Population size: 50
- Number of generations: 150
- Crossover probability: 0.7
- Blend crossover α : 0.5
- Mutation probability: 0.3 using Gaussian mutation ($\sigma=0.1$, individual gene probability=0.2).

The fitness function returns a tuple (accuracy, regularisation) with weights (1.0, -1.0) indicating maximization of accuracy and minimization of regularisation.

C. Results

NSGA-II produced a Pareto front of 34 non-dominated solutions, and each shows a trade-off between accuracy and regularisation [32]. The extreme solutions illustrate the opposing nature of the objectives, as the solution with the highest accuracy reached 55.16% validation accuracy and a relatively large regularisation (L2) value of 572.61, whereas the lowest-regularisation solution achieved only 13.83% accuracy and regularisation of 419.08. After evaluating the best solution on the full test dataset, it achieved 50.61% test accuracy, indicating that during training mild overfitting occurred, confirming that NSGA-II identified meaningful structures within the weight space despite the limited population size. Meanwhile, the trade-off observed in the intermediate solutions was more balanced, with moderate L2 values (approximately 460-510) leading to reasonable predictive performance. Therefore, the Pareto front demonstrates that higher accuracy generally requires larger weight magnitudes. Nevertheless, the curve starts to flatten for very large weights, implying diminishing returns in classification performance. This observation suggests that additional increases in weight size do not necessarily mean vital improvements in accuracy beyond a threshold.

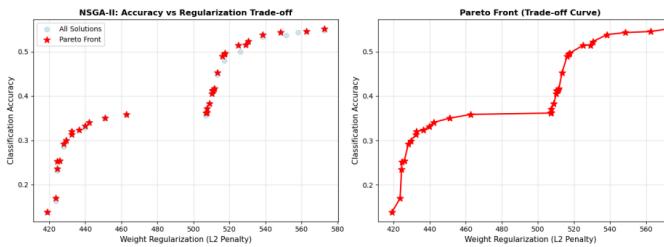


Figure 3: Pareto Front for accuracy vs. regularization trade-off

Single vs Multiple-Objective Analysis

Comparing bi-objective optimization with single-objective optimization, this problem appears to be better suited for single-objective optimization when the only component

trained is the last layer. Since the primary goal is to achieve high classification accuracy, regularisation is better considered as a secondary constraint. This is supported by the results, which show that single-objective approaches, specifically Adam, achieved the highest test accuracy (89.59%), outperforming NSGA-II. Although NSGA-II observed the trade-off between accuracy and regularisation using the Pareto front, most of the solutions it offered were not practically useful, as they concentrated too much on reducing weight magnitude at the cost of predictive performance. On the other hand, regularisation can be handled more effectively within a single objective framework through weight decay, which enhances generalization without losing accuracy. Overall, for this setting, multi-objective optimisation adds additional complexity without providing performance benefits, making single-objective optimisation the more appropriate choice.

VI. CONCLUSION

To conclude, this work illustrates a comprehensive comparison of population-based algorithms for optimizing the last layer weights of a ResNet-9 neural network for CIFAR-10 image classification. The gradient-based optimiser Adam achieved the strongest performance, outperforming DE, CLPSO, and SADE, while standard PSO produces the weakest results among all approaches. Furthermore, our experiments indicate that this problem is better suited to single objective optimisation, as the bi-objective NSGA-II method did not offer extra performance benefits despite providing an informative demonstration of the accuracy-regularisation trade off.

Future research could investigate hybrid optimisation algorithms that combine gradient-based learning with evolutionary search (memetic algorithms), extending optimisation to deeper layers of the network, or investigate co-evolutionary approaches where different subsets of weights evolve in separate populations. In addition, surrogate-assisted evolutionary algorithms could be explored to reduce computational cost. These strategies may better exploit the strengths of evolutionary algorithms beyond the simplified final-layer training scenario examined here.

VII. REFERENCES

- [1] A. Krizhevsky, “Learning Multiple Layers of Features from Tiny Images,” Apr. 2009. Available: <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>
- [2] K. Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, Apr. 1980, doi: <https://doi.org/10.1007/bf00344251>.
- [3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-Based Learning Applied to Document Recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

- [5] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” in *International Conference on Learning Representations (ICLR)*, 2015.
- [6] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.
- [7] M. S. Hanif and M. Bilal, “Competitive residual neural network for image classification,” *ICT Express*, vol. 6, pp. 28–37, 2020, doi: <https://doi.org/10.1016/j.icte.2019.06.001>.
- [8] O. M. Awad *et al.*, “Improving ResNet-9 Generalization Trained on Small Datasets,” *arXiv preprint*, 2023, Available: <https://arxiv.org/abs/2309.03965>
- [9] D. P. Kingma and J. L. Ba, “ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION,” in *International Conference on Learning Representations (ICLR)*, 2015. Available: <https://arxiv.org/pdf/1412.6980>
- [10] L. Bottou, “Large-Scale Machine Learning with Stochastic Gradient Descent,” in *Proceedings of COMPSTAT’2010*, Y. Lechevallier and G. Saporta, Eds., Heidelberg, Germany: Physica-Verlag HD, 2010, pp. 177–186. doi: https://doi.org/10.1007/978-3-7908-2604-3_16.
- [11] J. Kennedy and R. Eberhart, “Particle Swarm Optimization,” in *IEEE International Conference on Neural Networks*, 1995, pp. 1942–1948.
- [12] R. Storn and K. Price, “Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces,” *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.
- [13] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer, “Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems,” *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 6, pp. 646–657, Dec. 2006.
- [14] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, “Comprehensive learning particle swarm optimizer for global optimization of multimodal functions,” *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 3, pp. 281–295, Jun. 2006, doi: <https://doi.org/10.1109/tevc.2005.857610>.
- [15] Z.-H. Zhan, J.-Y. Li, and J. Zhang, “Evolutionary deep learning: A survey,” *Neurocomputing*, vol. 483, pp. 42–58, Apr. 2022, doi: <https://doi.org/10.1016/j.neucom.2022.01.099>.
- [16] E. Galván and P. Mooney, “Neuroevolution in Deep Neural Networks: Current Trends and Future Challenges,” *IEEE Transactions on Artificial Intelligence*, vol. 2, no. 6, pp. 476–493, Dec. 2021, doi: <https://doi.org/10.1109/TAI.2021.3067574>.
- [17] Y. Jin, T. Okabe, and B. Sendhoff, “Neural Network Regularization and Ensembling Using Multi-objective Evolutionary Algorithms,” in *IEEE Congress on Evolutionary Computation (CEC)*, 2004, pp. 1–8.
- [18] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
- [19] T. Elsken, J. H. Metzen, and F. Hutter, “Neural Architecture Search: A Survey,” *Journal of Machine Learning Research*, vol. 20, pp. 1–21, Mar. 2019.
- [20] T. M. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, “Meta-Learning in Neural Networks: A Survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 5149–5169, Sep. 2022, doi: <https://doi.org/10.1109/TPAMI.2021.3079209>.
- [21] J. Martens and R. Grosse, “Optimizing neural networks with Kronecker-factored approximate curvature,” *arXiv preprint*, 2015, Available: <https://arxiv.org/abs/1503.05671>
- [22] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” 2015. Available: <https://arxiv.org/pdf/1502.03167.pdf>
- [23] V. Nair and G. E. Hinton, “Rectified Linear Units Improve Restricted Boltzmann Machines,” in *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010. Available: <https://www.cs.toronto.edu/~fritz/absps/reluICML.pdf>
- [24] M. Lin, Q. Chen, and S. Yan, “Network In Network,” *arXiv*, 2014, Available: <https://arxiv.org/abs/1312.4400>
- [25] E. Real *et al.*, “Large-Scale Evolution of Image Classifiers,” 2017. Available: <https://arxiv.org/pdf/1703.01041>
- [26] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?,” 2014. Available: <https://arxiv.org/pdf/1411.1792>
- [27] T. Dang, T. T. Nguyen, C. F. Moreno-Garcia, E. Elyan, and J. McCall, “Weighted ensemble of deep learning models based on comprehensive learning particle swarm optimization for medical image segmentation,” in *IEEE Congress on Evolutionary Computation (CEC)*, Krakow, Poland: IEEE, 2021, pp. 744–751. Available: <https://doi.org/10.1109/CEC48532.2021.9504929>
- [28] A. K. Qin and P. N. Suganthan, “Self-adaptive differential evolution algorithm for numerical optimization,” in *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC 2005)*, IEEE, 2005, pp. 1785–1791. Available: <https://doi.org/10.1109/CEC.2005.1554904>
- [29] J. Cao, Z. Lin, and G.-B. Huang, “Self-Adaptive Evolutionary Extreme Learning Machine,” *Neural Processing Letters*, vol. 36, no. 3, pp. 285–305, 2012, doi: <https://doi.org/10.1007/s11063-012-9236-y>.
- [30] J. Ku and K. Xing, “Self-adaptive differential evolutionary extreme learning machine and its application in facial age estimation,” in *Proceedings of the 2017 International Conference on Computer Network, Electronic and Automation (ICCNEA 2017)*, IEEE, 2017, pp. 72–77.
- [31] M. Baiocetti, G. Di Bari, A. Milani, and V. Poggioni, “Differential Evolution for Neural Networks Optimization,” *Mathematics*, vol. 8, no. 69, pp. 69–69, 2020, doi: <https://doi.org/10.3390/math8010069>.
- [32] C. A. Coello Coello, “Evolutionary multi-objective optimization: a historical view of the field,” *IEEE Computational Intelligence Magazine*, pp. 28–36, 2006.

APPENDIX

1. *Danish Whelan Bin Zamri*

- Contributed to the development of code. I.e (development of algorithms/Hyperparameter tuning/Development of figures/Debugging and etc)
- Co-authored section 3 with Mohammad Shayaan on elaborating on choice of algorithms, with justifications.
- Co-Authored Section 4 with Mohammad Shayaan on the description of the experimental setup and results of algorithms.
- Assisted in Documentation for other sections.

2. *Eleana Vrachimi*

- Contributed to the decision-making process for selecting the model architecture and training algorithms.
- Co-authored the Introduction and Model Architecture Sections in collaboration with Elpida Dimitriou.
- Wrote the discussion comparing single-objective and multi-objective optimisation.
- Assisted in Documentation for all sections.
- Authored the conclusion.

3. *Elpida Dimitriou*

- Helped select and justify the model architecture and training algorithms.
- Co-authored the Introduction and Model Architecture Sections in collaboration with Eleana Vrachimi.
- Wrote the explanation of the NSGA-II algorithm, including the bi-objective formulation and experimental setup.
- Authored the abstract
- Assisted in Documentation for all sections.

4. *Mohammad Shayaan Aslam Khan*

- Worked on all sections of the Code, introduced optimizations in the runtime to allow for quicker results along with fine tuning and bug fixing
- Co-authored Section 3 with Danish Whelan on the explanation and elaboration about the choice of algorithms used.
- Co-authored Section 4 with Danish Whelan to describe the initial and experimental setups as well as the result of the algorithms.