Exploring Neural Networks and Optimization Algorithms

Suraj Rohira Lucas
Comp Sci & Elec Eng
University of Surrey
Guildford, Surrey, United Kingdom
suraj.rohira0@gmail.com

***Abstract*—Image classification is a well known problem in computer vision. Deep learning and neural networks have greatly developed in recent years, and have shown great performance on this type of a problems. Selecting the correct neural network architecture is still a challenge, and not only the architecture, but choosing an optimization algorithm to train the network is as important as the architecture itself. Optimizing the parameters of a neural network is a complex problem due to the high-dimensional search space and non-convexity nature of the problem, for which gradient-descend methods and backpropagation are commonly used. In this paper, we propose a neural network architecture for image classification, and a memetic optimization algorithm to train it. The performance is compared with RMSprop and genetic algorithm using the CIFAR-10 data set. Finally, we also address the problem as a bi-objective problem.**

I. Introduction

This section provides an overview of neural network (NN) architectures for image classifications, optimization algorithms, and unresolved problems. Additionally, a brief summary of our approach will be given.

A. Neural Network Architectures for Image Classification

The use of deep learning (DL) as a machine learning (ML) algorithm for image classification has risen due to the progress in hardware technology. The use of different NN architectures, such as Recursive Neural Networks (RvNNs), Recurrent Neural Networks (RNNs), or Autoencoders for different purposes are common, however, the NN architecture that has shown the best performance for image classification has been the convolutional neural network (CNN) [11].

The main advantage of a CNN compared to its predecessors are that it automatically detects features from an image using the convolutional layer. It offers other benefits like parameter sharing, as the same convolutional kernel is shared between all the pixels, which makes the operation simple and efficient, and allows it to operate on very large-scale data sets. It also greatly reduces the amount of training parameters and decreases the risk of over fitting [15].

The first CNN model, LeNet-5, was presented by Lecun et al. [1] however due to the lack of large scale training data, computational power, and theoretical knowledge on hyper-parameter tuning and training algorithms, the model only had good results on classifying handwritten numbers. Based on this architecture, Krizhevsky et al. [7] proposed a new model, AlexNet, the most significant changes were the use of rectified linear unit (ReLU) instead of sigmoid as the activation function, dropout and data augmentation were added to control the model complexity and avoid over fitting. With the success of AlexNet, several works are proposed to improve its performance, VGGNet [8] which has a simpler architecture with small filters and deeper layers, Network in Network (NIN) [9], or GoogLeNet Inception V1/V2/V3/V4 which introduced inception modules. The main contributions of the mentioned architectures and the reason for their increase in performance was achieved by increasing the depth of the networks, normally by adding convolutional and pooling layers. Increasing the depth led to problems such as the vanishing gradient problem. ResNet [12] is an architecture that solved this problem by adding residual blocks that skip connections, and allows the gradient to be passed.

Autoencoders are a type of NN used to learn efficient data encoding in an unsupervised way. An autoencoder is useful for feature extraction, noise removal and other similar tasks because it converts high dimensional data into low dimensional data. A sparse autoencoder is typically used for image classification. It has the benefit of filtering out noise and irrelevant features in the encoder, so they learn important features of the input. However, the computational complexity is increased due to the sparsity constraint, and the hyper-parameters chosen affect the performance significantly, so if they have not been tuned optimally it can harm the network. Denoising autoencoder can be used as a form of data augmentation. Choosing how much noise to add and the type of noise might require domain knowledge and this process can impact the accuracy because the noise added could result in a loss of important information from the original input. Convolutional autoencoders are used in CNNs as their building blocks. They can compress high dimensional data into low dimensional data which improves storage and transmission efficiency. The convolutional encoder can handle images with slight variations (such as position or orientation) and can also reconstruct missing parts of the image. The main disadvantage is that they can lead to over fitting, regularisation techniques need to be used to counteract this.

Vision transformers (ViTs) [30] is a recently proposed type of NN also used for image classification, based on the Transformer architecture [31]. Numerous advantages of ViTs include its ability to learn global features of images, which is capable because they focus on all specific parts of an image no matter the location. This is greatly beneficial for problems where object detection or scene understanding are involved. Also in comparison to a CNN, ViTs are less sensitive to data augmentation; the models global understanding of an images overall structure makes ViTs more robust to local changes brought on by data augmentation, meaning they can be suitably trained on smaller data sets. On the contrary, ViTs are less efficient than a CNN for image processing. The reason being that they focus on all parts of an image even if a section is not important for the task being carried out. Additionally they are expensive to train; the self-attention layers have a quadratic cost in the number of pixels, so large data sets require high computational power.

B. Optimization Algorithms

Optimization algorithms have a crucial role in the process of developing a NN model, optimization algorithms are used to update the learnable parameters (i.e. weight and bias) of a model.

We will start discussing gradient descent (GD) methods, as it is the most common algorithm used to optimize the parameters of a neural network. GD has three main variants: batch GD, stochastic gradient descent (SGD) and mini-batch GD.

Batch GD, optimises its parameters after evaluating the entire training data set. This presents two important problems, which are that it can converge really slowly, and if the data set is large then it is possible for the data set not to fit in memory. Another aspect to consider is the initial starting point, as it will determine the basin of the search which will probably be a local minimum. In SGD the parameters are updated after every training sample, this makes it converge faster, however due to frequent parameter updates, the loss function fluctuates much more compared to batch GD, this fluctuation however makes it possible for it to arrive to potentially better basin of parameters where there is a better local minimum or even global minimum. Mini-batch GD updates the parameters for every mini-batch of n training samples, this allows for lower fluctuations of the parameter updates leading to more stable convergence [13].

Some limitations with these three GD methods are the following. The selection of learning rate, if set too low it will converge slowly, and if set too high it could oscillate between the minimum. Having a fixed learning rate can also make convergence slower, as it is commonly preferable having a larger learning rate at the start whilst lower ones towards the end. The same learning rate for all parameters might not be desirable if the data set is sparse. Finally, a NN optimization problem is highly non-convex, so avoiding getting trapped in a sub-optimal local minimum or saddle point is a challenge.

To overcome these problems alternatives such as adding momentum [10] exist, we can also find adaptive learning first-order-gradient-based methods, which extend on the GD method. AdaGrad [20] decreases the importance of the learning rate selection as it automatically adapts the learning rate of each parameter, it naturally incorporates regularisation and shows better performance on sparse data sets, but in non-convex problems and dense gradients performance decreases because the learning rate diminishes which makes it become stuck in saddle points or local minimum. RMSProp combines Rprop [21] to adapt the learning rate for avoiding both exploding and vanishing gradients. Adam [18] is one of the most commonly used training algorithms and improves RMSProp towards the end of the optimization.

All of the methods mentioned until now are gradient-based methods, these present various limitations, for example, the gradient does not always provide information about the direction of the optimal solution and can be noisy, the architecture of the neural network might create a vanishing or exploding gradient problem, for multi-objective problems it may be stuck at local minimums, and calculating the gradient is not always possible [24].

Meta-heuristic (MH) [17] are gradient-free methods that are characterized by being global search, non-deterministic search methods, that introduce stochastic components to explore the search space and exploit the best solution, aiming to find the global minimum. They have been shown to be beneficial for solving large problems fast, and for making a more robust algorithm. They are also simple to design and implement [27].

They can be divided into single-solution-based for example simulated annealing (SA) [26], or population-based, which are further divided into evolutionary algorithms, for example genetic algorithms (GA) [2], differential evolution (DE) [4], or evolution strategies (ES) [3], and swarm intelligence, for example particle swarm optimization (PSO) [5], or whale optimization algorithm (WOA) [6].

NNs and DL architectures are considered ”black boxes” for which evolutionary and swarm intelligence algorithms are a better optimization approach. DE is one of the most popular MH algorithms due to the simpler implementation compared to other MH algorithms, it has shown to work well on non-linearity and multi-modal problems, and have been on the top three-best performing optimization algorithms in most CEC competitions. GA, and PSO have been used for training NNs and DL and have also shown success. However, according to the No Free Lunch (NFL) [29] theorem each problem is unique, so not every MH algorithm might be useful, and the only way to find is by experimentation. Due to MH searching the problem space, it is hard to determine if a MH algorithm is the best method for a problem. MHs have to find a suitable trade-off between exploration and exploitation which might be difficult, and changing algorithm variables does not have an influence in some cases. [28].

C. Our contribution

In this work, we have used a memetic algorithm to optimize the last layer of a CNN for image classification. To benchmark our algorithm, it has been compared to RMSprop and GA, using the CIFAR-10 data set. From the literature reviewed, we have not found related work on training a CNN for image classification using a memetic algorithm, as this is commonly done with a gradient-based approach, however, our work aims to evaluate a high-dimensional problem like this using a memetic algorithm that combines gravitational search algorithm (GSA) and Rprop. The results show that it outperforms GA, and performs equally as good as state-of-the-art algorithms such as Rprop. The problem is also addressed as a bi-objective problem using Non-Dominated Sorting Algorithm II (NSGA-II) which does not show better performance.

II. Neural Network Architecture

The selected NN architecture was an adaptation of the residual NN (ResNet) presented in [12]. Adaptations where made to make it suitable for the CIFAR-10 data set, and our specific problem.

A. Architecture

The architecture is shown in Figure 1. The input is a 32x32x3 image. The convolutional layers have all 3x3 filters, stride and padding of 1, and are followed by batch normalization and a rectified linear unit (ReLU) activation function, the number of output channels are 64, 128, 128, 128, 256, 64, 64 and 64 respectively. The architecture can be divided into nine blocks.

The first block (blue block) does not change the height and width of the image. The second, fifth and sixth blocks (green blocks) have an additional max pooling layer after the ReLU activation function, with a kernel size of 2x2 and padding of 0, after each of these max pooling layers, the height and width of the output image will be half of the height and width it had in the input. The third, fourth, seventh and eighth blocks (orange blocks) contain the residual connections, which add the original input back to the output feature map obtained by passing the input through two convolutional layers. These do not contain a pooling layer, so the input and output height and width are the same. The ninth and final block (purple block) has a max pooling layer of 4x4 filter, flattening, and a 10-way fully-connected layer that gives the probability of the image belonging to each of the classes.

B. Justification

A CNN architecture was chosen because the convolutional layers allow the NN to discover and extract multiple feature from the images. It has other benefits like sparse connectivity and weight sharing which reduces the number of trainable network parameters and in turn helps the network to enhance generalization and to avoid over fitting [14]. Batch normalization is added as it makes optimization easier by smoothing the landscape of the problem [16]. It also allows using larger learning rates to increase the convergence rate, without causing the exploding gradient problem [19]. With respect to the addition of residual connections, these were added as they solve various problems encountered by deep NNs. A deep NN has problems like vanishing gradient problem and the curse

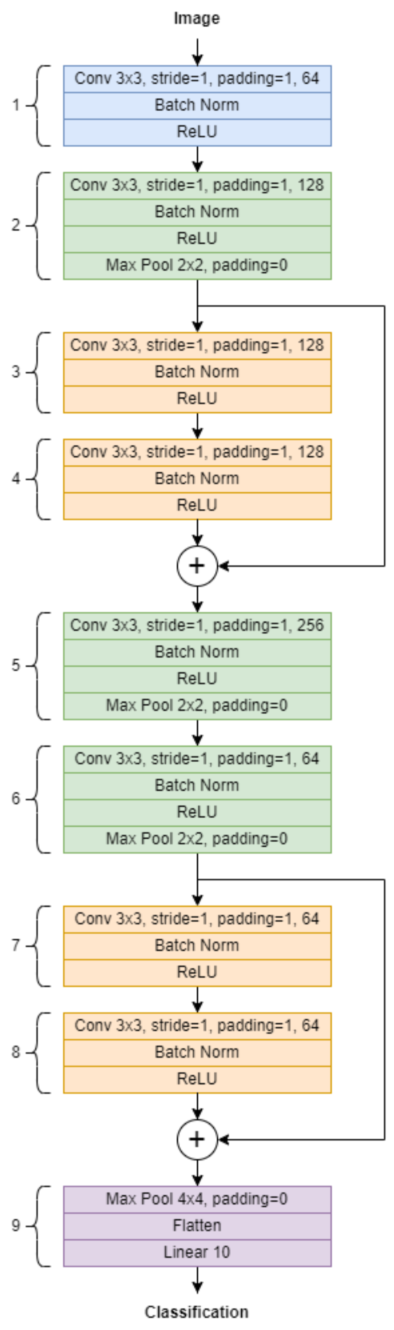


Image
1
Conv 3x3, stride=1, padding=1, 64
Batch Norm
ReLU
2
Conv 3x3, stride=1, padding=1, 128
Batch Norm
ReLU
Max Pool 2x2, padding=0
3
Conv 3x3, stride=1, padding=1, 128
Batch Norm
ReLU
4
Conv 3x3, stride=1, padding=1, 128
Batch Norm
ReLU
+
5
Conv 3x3, stride=1, padding=1, 256
Batch Norm
ReLU
Max Pool 2x2, padding=0
6
Conv 3x3, stride=1, padding=1, 64
Batch Norm
ReLU
Max Pool 2x2, padding=0
7
Conv 3x3, stride=1, padding=1, 64
Batch Norm
ReLU
8
Conv 3x3, stride=1, padding=1, 64
Batch Norm
ReLU
+
9
Max Pool 4x4, padding=0
Flatten
Linear 10
Classification

Fig. 1. Neural network architecture

of dimensionality, also it may not be able to learn simple functions. A deep NN is harder to optimise, this has been shown using CIFAR-10 and ImageNet data sets, once a certain number of iterations are passed, the training and testing error of deep NNs start to increase again. This is not an over fitting problem, as it is not only the testing error the one that is increasing, but the both the training and testing errors. This is known as the degradation problem, where the first layers of the network are learning better than the deeper layers [12]. Adding residual connections between layers has the effect of skipping these, with the result of performing at least as good as the previous layer, which solves the degradation problem. Also, larger gradients can be propagated to the initial layers, solving the vanishing gradient problem. A last consideration is that the residual connections allows having an architecture with more layers for the reasons just mentioned, so even if the optimal number is not known, the architecture itself will skip those layers that are not useful, this behaviour was desired for our problem as we did not have the time to test various architectures.

III. Our Optimization Algorithm

A. Algorithm Description

The proposed optimization algorithm consists of a memetic algorithm that combines a meta-heuristic, GSA [22], and a local search with linear convergence rate, Rprop [21]. This is an adaptation of the memetic gravitational search algorithm (MGSA) presented in [23].

This algorithm uses a meta-heuristic approach to search the global space, and when it detects that an unexplored promising neighbourhood has been found it starts a local search method. We say that a promising neighbourhood of unexplored local minima has been detected if the following two conditions are met:

1) The best solution in the current generation (current best) has a better fitness value than the best solution found until now (global best).
2) The distance between the current best position is far from the global best position.

This can be formulated as shown in (1) and (2).

$$f(\mathbf{g}_{best}) - f(\mathbf{x}^t_{best}) > \varepsilon \tag{1}$$

$$\|\mathbf{g}_{best} - \mathbf{x}^t_{best}\| > \gamma \tag{2}$$

When this conditions are met, the local search starts and updates the *positions* until condition (3) and (4) are satisfied.

$$\|\mathbf{w}_{k+1} - \mathbf{w}_k\| < tol \tag{3}$$

$$|f(\mathbf{w}_{k+1}) - f(\mathbf{w}_k)| < tol \tag{4}$$

where $\mathbf{w}_{k+1}$ and $\mathbf{w}_k$ are the new solution and previous solution found during the local search respectively.

An initial *population* of *positions* (each *position* being a potential set of NN parameters) is initialized and after each generation the *position* is updated. The way in the *positions* are updated is by calculating the *mass* of each *position*, according to (5), which takes into consideration the fitness value (a low fitness value will have a high *mass*). The fitness value is the loss of a *position* after calculating the cross entropy loss using one mini batch from the training data set.

$$M^t_i = \frac{m^t_i}{\sum_{j=1}^{N} m^t_j} \tag{5}$$

where m^t_i is given by (6)

$$m^t_i = \frac{f(\mathbf{x}^t) - worst^t}{best^t - worst^t} \tag{6}$$

where $best^t$ is the fitness value of the global best position, and $worst^t$ is the worst fitness value at this generation.

Using the *mass*, the *acceleration* of each *position* at each dimension is calculated according to (7). The direction of the *acceleration* will be towards *positions* (parameters) with a higher *mass* (lower loss), this is what makes the algorithm converge.

$$\mathbf{a}^t_i = G^t \cdot \sum_{\substack{j \neq i \\ j=1}}^{N} rand_j \cdot \mathbf{F}^t_{ij} \tag{7}$$

where $\mathbf{F}^t_{ij}$ is given by (8) and G^t is given by (9)

$$\mathbf{F}^t_{ij} = rand_j \cdot M^t_j \cdot \frac{\mathbf{x}^t_j - \mathbf{x}^t_i}{\|(\mathbf{x}^t_i - \mathbf{x}^t_j)\| + \epsilon} \tag{8}$$

$$G_t = G_0 \cdot e^{-\alpha \frac{t}{T}} \tag{9}$$

where G_t is the gravitational decay, G_0 the gravitational constant, α the learning rate, t the current generation, and T the total number of generations.

Once the *acceleration* is calculated the *velocities* and then the *positions* are updated according to (10) and (11) respectively.

$$\mathbf{v}^{t+1}_i = rand_i \cdot \mathbf{v}^t_i + \mathbf{a}^t_i \tag{10}$$

$$\mathbf{x}^{t+1}_i = \mathbf{x}^t_i + \mathbf{v}^{t+1}_i \tag{11}$$

The pseudocode of the algorithm is shown in Algorithm 1. And a flowchart is shown in Figure 2.

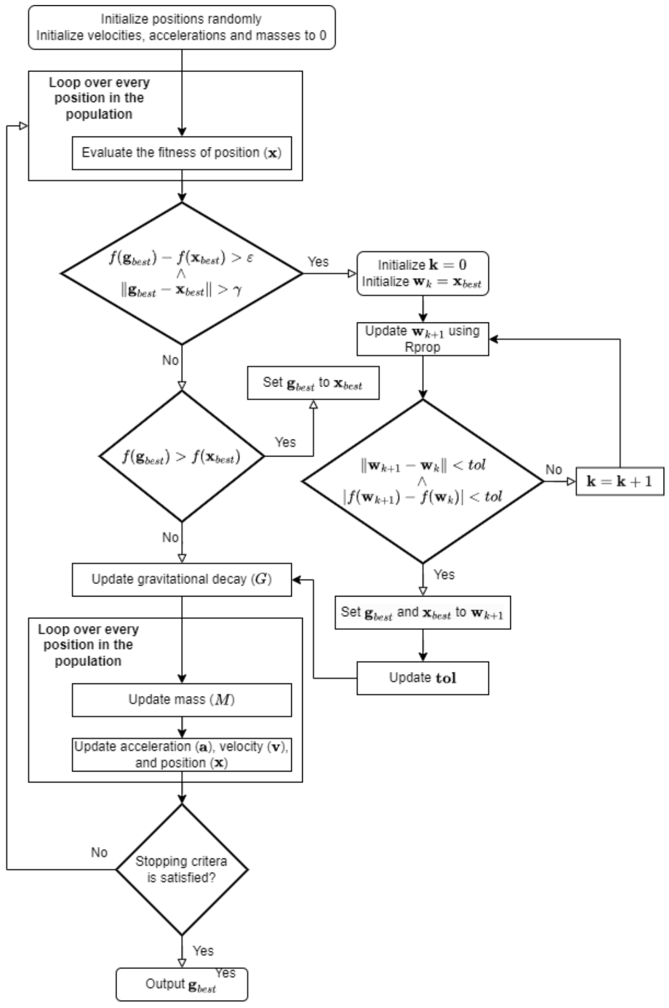
B. Algorithm Justification

The use of a memetic algorithm combines the exploration aspect of a global and stochastic search method such as GSA, with the exploitative characteristics of a local gradient-descent search method like Rprop. Having an exponential decaying term (G_t) manages a balance between exploration and exploitation, being more exploratory at the start and becoming more exploitative towards the end. The *tol* parameter regulates the intensity of the local search, by increasing it on each local search we again allow for the desired exploration/exploitation trade-off.

Various problems appear when using only a gradient-based method for optimization, which our optimization algorithm (OOA) reduces. Gradient-based methods assume that the gradient provides useful information about the target we want to reach, however, there may be problems where the gradient provides little information [24], for this reason, OOA does not

Algorithm 1 Our Optimization Algorithm

```
$\varepsilon \leftarrow 0$                $\alpha \leftarrow 20$
$\gamma \leftarrow 1$                $N \leftarrow 200$
$tol \leftarrow 0.01$             $T \leftarrow 50$
$G_0 \leftarrow 100$             $t \leftarrow 0$

randomly initialize $N$ $\boldsymbol{x}$
initialize $N$ $M$, $\boldsymbol{v}$, and $\boldsymbol{a}$ to $0$
$\mathbf{g}_{best} \leftarrow \mathbf{x}_{best}$

while $t < T$ do
    calculate the fitness value for all positions

    if $f(\mathbf{g}_{best}) - f(\mathbf{x}^t_{best}) > \varepsilon \wedge \|\mathbf{g}_{best} - \mathbf{x}^t_{best}\| > \gamma$ then
        $k \leftarrow 0$
        $\mathbf{w}^k \leftarrow \mathbf{x}^t_{best}$
        $condition1 \leftarrow False$
        $condition2 \leftarrow False$
        while $!condition1 \wedge !condition2$ do
            update $\mathbf{w}^k$ using Rprop
            $condition1 \leftarrow \|\mathbf{w}^{k+1} - \mathbf{w}^k\| < tol$
            $condition2 \leftarrow |f(\mathbf{w}^{k+1}) - f(\mathbf{w}^k)| < tol$
        end while
        $\mathbf{v}^t_{best} \leftarrow \mathbf{w}^{k+1} - \mathbf{g}_{best}$
        $\mathbf{x}^t_{best} \leftarrow \mathbf{w}^{k+1}$
        $\mathbf{g}_{best} \leftarrow \mathbf{w}^{k+1}$
        $tol \leftarrow tol/10$

    else if $f(\mathbf{g}_{best}) > f(\mathbf{x}^t_{best})$ then
        $\mathbf{g}_{best} \leftarrow \mathbf{x}^t_{best}$

    end if

    update $G_t$ according to (9)
    update $M_t$ according to (5)
    update $\boldsymbol{a}_t$ according to (7)
    update $\boldsymbol{v}_t$ according to (10)
    update $\boldsymbol{x}_t$ according to (11)

    $t \leftarrow t + 1$
end while
```



Initialize positions randomly
Initialize velocities, accelerations and masses to 0
Loop over every position in the population
Evaluate the fitness of position $(\mathbf{x})$
$f(\mathbf{g}_{best}) - f(\mathbf{x}_{best}) > \varepsilon \wedge \|\mathbf{g}_{best} - \mathbf{x}_{best}\| > \gamma$
Yes
No
Initialize $\mathbf{k} = 0$
Initialize $\mathbf{w}_k = \mathbf{x}_{best}$
Update $\mathbf{w}_{k+1}$ using Rprop
Set $\mathbf{g}_{best}$ to $\mathbf{x}_{best}$
$f(\mathbf{g}_{best}) > f(\mathbf{x}_{best})$
Yes
No
$\|\mathbf{w}_{k+1} - \mathbf{w}_k\| < tol \wedge |f(\mathbf{w}_{k+1}) - f(\mathbf{w}_k)| < tol$
No
$\mathbf{k} = \mathbf{k} + 1$
Yes
Update gravitational decay (G)
Set $\mathbf{g}_{best}$ and $\mathbf{x}_{best}$ to $\mathbf{w}_{k+1}$
Loop over every position in the population
Update mass (M)
Update $\mathbf{tol}$
Update acceleration $(\mathbf{a})$, velocity $(\mathbf{v})$, and position $(\mathbf{x})$
No
Stopping criteria is satisfied?
Yes
Output $\mathbf{g}_{best}$
Yes

Fig. 2. Our optimization algorithm flowchart

only rely on the gradient information to find better solutions. In [25], GA and GSA are combined, and demonstrate better solutions than a gradient-based back propagation method.

The combination of GSA with qN search directions has already demonstrated to outperform Rcr-JADE, PSO, GA, and COBIDE in the XOR problem [23]. Also, the combination of a chaotic GSA (CGSA) and qN has demonstrated to address the curse of dimensionality [23] which is something desired in the case of training a layer with 650 parameters.

IV. Results

Three optimization algorithms were tested, a gradient-descent method, Rprop, a population-based method, GA, and OOA.

A. Experimental Setup

All the layers of the NN were first trained for 50 epochs, using mini batch GD with a mini batch size of 128. This gave an initial accuracy of 80%. To perform the three experiments, the mini batch size was increased to mini batches of 1000. For each experiment, all the layers except for the last layer were froze, and the last layer was randomized, meaning that the training was only done on the last layer of the network.

The hyper-parameters for OOA are shown in Table III, the NN was trained for 50 generations, on each generation, the entire *population* was evaluated using one single mini batch from the training data set, the mini batch was chosen randomly after each generation and the *position* with the best fitness value (i.e. lowest loss) at each generation was recorded. If the local search begun, then the NN was trained as normal (i.e. iterating over the entire training data set in mini batches until the stopping condition is met), and the loss was recorded after each mini batch.

The hyper-parameters for RMSprop are shown in TableI, the NN was trained for the same number of iterations as the number of recordings that OOA had, for example, if OOA performed 50 generations, and additionally performed 60 iterations using the local search method, then RMSprop would be trained for 110 (50 + 60) iterations. One iteration means evaluating the loss function of one mini batch. The loss after each mini batch was recorded.

The hyper-parameters for GA are shown in Table II, the number of generations was determined the same way as how the number of iterations was determined for RMSprop. On each generation, the entire population was evaluated using one single mini batch from the training data set, the mini batch was chosen randomly after each generation and the *individual* with the best fitness value (i.e. lowest loss) at each generation was recorded.

Hyper-parameter	Value
Iterations	175
Mini batch size	1000
Learning rate	0.01
Alpha	0.99
Weight decay	0
Momentum	0

TABLE I
RMSPROP HYPER-PARAMETERS

Hyper-parameter	Value
Generations	175
Mini batch size	1000
Bounds	[0.0, 1.0]
Population size	200
Bits per gene	10
Crossover probability	0.9
Uniform crossover probability	0.5
Mutation probability	0.1
Bit mutation flip probability	1.0 / (650 * 10)
Elitism	5.0

TABLE II
GENETIC ALGORITHM HYPER-PARAMETERS

Hyper-parameter	Value
Generations	50
Mini batch size	1000
Bounds	[0.0, 1.0]
Population size	200
ε	0
γ	1
tol	0.01
G_0	100
α	20

TABLE III
OUR ALGORITHM HYPER-PARAMETERS

B. Experimental Results

The evaluation metrics used were the accuracy, given by the amount of correctly classified images over the the total number of images. The accuracy was measured on the test data set (10,0000 images), the results are shown in Table IV.

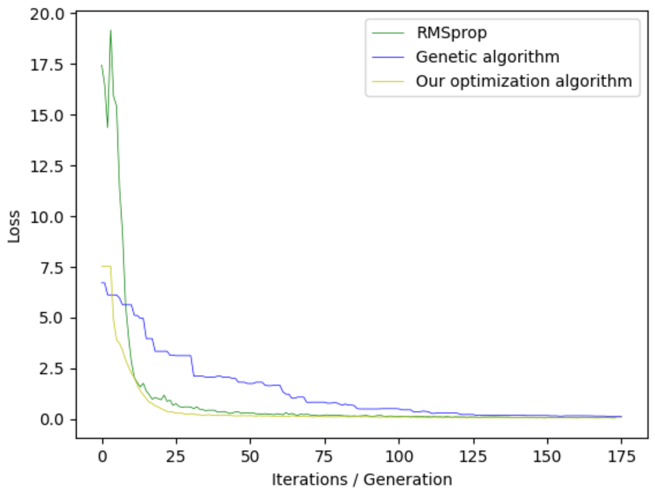


Fig. 3. Convergence trend

Additionally, the convergence trend (i.e. the loss against the number of iterations/generations) is shown in Figure 3.

Optimization Algorithm	Accuracy (%)
RMSprop	75
GA	72
Our optimization algorithm	75

TABLE IV
ACCURACY FOR RMSPROP (GRADIENT-DESCENT METHOD), GENETIC ALGORITHM (POPULATION-BASED) AND OUR OPTIMIZATION ALGORITHM

From Table IV we can observe how from the three methods, the gradient-based (RMSprop) and OOA give the best accuracy of 75%, followed by the population-based (GA) with 72%.

By observing the curves on Figure 3 we can arrive to various conclusions.

RMSprop and OOA have the fastest convergence, whilst GA has continuous iterations of no improvement, leading to a slower convergence. Reasons for this slower convergence are that GA uses a stochastic method to search for solutions, meaning that the parameters are adjusted without using the loss function or gradient information to guide the search, however, in the case of the other two algorithms, using the gradient (during the local search in case of OOA) allows them to update the parameters towards a solution with a lower loss on each iteration, there are yet situations where the loss increases which may be due to the learning rate adjustment.

It is also noticed how RMSprop starts from a higher loss than the other two methods. The reasons for GA and OOA to have lower loss at the start is due to the random initialization of solutions, and the constraints on the minimum and maximum bounds of the decision variables (i.e. parameters), which were constrained to range between 0.0 and 1.0. Having this constraint, and generating 200 random solutions, gave a higher probability of generating a good solution.

If we consider the training time of each algorithm, RMSprop was the fastest (33.8s), followed by OOA (45min 42.4s), and finally (104m 19.7s). We can see a great difference between all of them, OOA and GA have such a high increase in time, as they have to evaluate every possible solution from the population for each generation.

In the case of RMSprop and OOA, they achieve a low loss within a few iterations, having such a low loss but only achieving a 75% accuracy can be due to different factors. First, it could be due to a limit on the NN architecture instead of the optimization algorithm, another reason might be over fitting, and finally, it could be due to being stuck in a local optimum. For RMSprop escaping this can be challenging, however, in the case of OOA, this could be avoided by adjusting the hyper-parameters such as the tolerance (tol) or learning rate (α), or gravitational constant (G_t), to make it more exploratory rather than exploitative as such early stages.

The main advantages of RMSprop are the fast training time and the almost guarantee of optimizing the parameters, however, it can become stuck in a local minimum or be over fitting if it is not stopped early. In the case of GA and OOA, the benefits come from doing a global search, and therefore having a greater probability of finding better solutions than a gradient-based method, however, GA has various drawbacks. First, it requires a significantly higher training time, it does not guarantee convergence, and it can have difficulties for high-dimensionality problems, as the update of the solutions is random, so having more dimensions to optimize decreases the probabilities of finding good solutions. In the case of OOA, the training time is also much higher, however it does not have the high-dimensionality problem, as tuning the local search correctly can help it guide the search.

V. Bi-Objective Problem

The training algorithm was also addressed as a bi-objective problem using NSGA-II. The first objective was minimizing the loss function, and the second objective was minimizing the sum of the square of the weights.

The hyper-parameters chosen are shown in Table V.

Hyper-parameter	Value
Generations	175
Bounds	[0.0, 1.0]
Mini batch size	1000
Population size	200
Bits per gene	10
Crossover probability	0.9
Uniform crossover probability	0.5
Mutation probability	0.1
Bit mutation flip probability	1.0 / (650 * 10)

TABLE V
NSGA-II HYPER-PARAMETERS

The loss function was the cross-entropy loss, and on each generation, the loss function was calculated by evaluating the individual over one mini batch (i.e. 1000 images) from the training data set. After each generation, a different mini batch was randomly chosen, so all the individuals from the same generation were evaluated using the same mini batch. The sum of the square of the weights was calculated considering only the weight and bias from the last layer, as this was the only layer that was being trained.

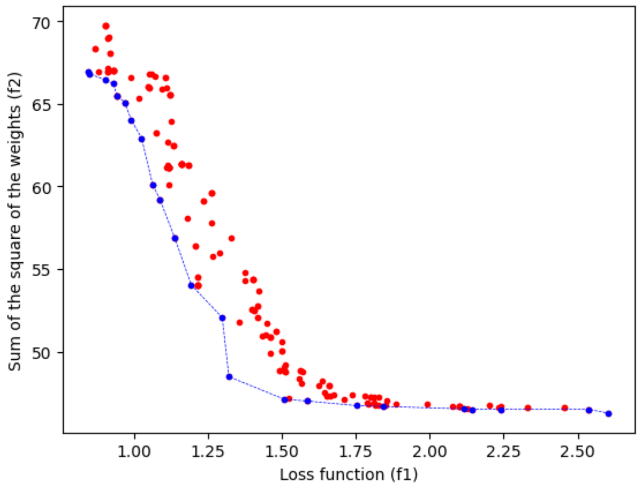


Fig. 4. Pareto front

The results gave a Pareto front of all the non-dominated solutions found during all the generations, the Pareto front and the final population is shown in Figure 4. The accuracy for each of the solutions in the Pareto from were calculated, the results are shown in Table VI.

Non-Dominated Solution	F1	F2	Accuracy (%)
1	2.60	46.27	56
2	2.54	46.50	57
3	2.24	46.50	55
4	2.14	46.51	55
5	2.11	46.53	53
6	1.84	46.68	54
7	1.75	46.72	54
8	1.58	47.01	53
9	1.51	47.11	51
10	1.32	48.48	51
11	1.30	52.06	47
12	1.19	54.04	47
13	1.14	56.88	46
14	1.09	59.21	45
15	1.06	60.11	39
16	1.03	62.92	40
17	1.00	64.03	36
18	0.97	65.09	36
19	0.94	65.50	38
20	0.93	66.22	36
21	0.90	66.45	32
22	0.85	66.84	34
23	0.85	66.98	33

TABLE VI
ACCURACY AS A BI-OBJECTIVE PROBLEM

From the results, we observe that training the neural network parameters as a bi-objective problem provides worse accuracy compared to the single-objective optimization methods used in section IV. In this case, the first objective looks at minimizing the loss, i.e. fitting the model to the training data set, and the second objective is aimed at reducing the complexity of the model to avoid over fitting. From VI we clearly see the effect of regularization of the second minimization objective, as we obtain a higher accuracy for a solution with a higher loss but smallest sum of the square of the weights. The way in which the complexity of the model is measured is by calculating the sum of the square of the weights, therefore, to minimize the complexity large weights are penalised, and the weight values will tend towards zero. A problem this might introduces is that the layers will tend towards zero, therefore it could make some important layers inactive, eliminating what was learned.

VI. Conclusion

In this paper we have reviewed the most relevant work on image classification and optimization algorithms. Then, we presented our neural network architecture based on ResNet and our optimization algorithm based on a memetic algorithm, it has been compared against RMSprop and GA by training the last layer of the neural network on the CIFAR-10 data set, and it has demonstrated to show equal accuracy and similar convergence rate than a state-of-the-art algorithm such as RMSprop. The main reasons for these results seem to be the combination of global and local search. We have also experimented using NSGA-II to train the last layer as a bi-objective problem using the loss function and sum of the square of the weights as the two minimization objectives, this approach however has not shown an improvement in accuracy.

We consider this a first approach to alternative optimization methods for training neural networks. On future work, it should be tested other memetic algorithm combinations such as PSO, GA, DE, etc, with other local search methods. Finding the optimum parameters is still a challenge in deep learning, but we are confident that following this approach is a good path to solving this problem.

References

[1] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[2] Grefenstette, J. J. (1993, August). Genetic algorithms and machine learning. In Proceedings of the sixth annual conference on Computational learning theory (pp. 3-4).

[3] Michalewicz, Z., & Michalewicz, Z. (1994). Evolution strategies and other methods. Genetic algorithms+ data structures= evolution programs, 167-184.

[4] Storn, R., & Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization, 11, 341-359.

[5] Eberhart, R., & Kennedy, J. (1995, October). A new optimizer using particle swarm theory. In MHS'95. Proceedings of the sixth international symposium on micro machine and human science (pp. 39-43). Ieee.

[6] Mirjalili, S., & Lewis, A. (2016). The whale optimization algorithm. Advances in engineering software, 95, 51-67.

[7] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25.

[8] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition.

[9] Lin, M., Chen, Q., & Yan, S. (2013). Network in network. arXiv preprint arXiv:1312.4400.

[10] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. Neural networks, 12(1), 145-151.

[11] Srinivas, S., Sarvadevabhatla, R. K., Mopuri, K. R., Prabhu, N., Kruthiventi, S. S., & Babu, R. V. (2016). A taxonomy of deep convolutional neural nets for computer vision. Frontiers in Robotics and AI, 2, 36.

[12] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[13] Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

[14] Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., ... & Farhan, L. (2021). Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. Journal of big Data, 8, 1-74.

[15] Chen, L., Li, S., Bai, Q., Yang, J., Jiang, S., & Miao, Y. (2021). Review of image classification algorithms based on convolutional neural networks. Remote Sensing, 13(22), 4712.

[16] Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. Advances in neural information processing systems, 31.

[17] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. Computers & operations research, 13(5), 533-549.

[18] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[19] Bjorck, N., Gomes, C. P., Selman, B., & Weinberger, K. Q. (2018). Understanding batch normalization. Advances in neural information processing systems, 31.

[20] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research, 12(7).

[21] Riedmiller, M., & Braun, H. (1993, March). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In IEEE international conference on neural networks (pp. 586-591). IEEE.

[22] Rashedi, E., Nezamabadi-Pour, H., & Saryazdi, S. (2009). GSA: a gravitational search algorithm. Information sciences, 179(13), 2232-2248.

[23] García-Ródenas, R., Linares, L. J., & López-Gómez, J. A. (2021). Memetic algorithms for training feedforward neural networks: an approach based on gravitational search algorithm. Neural Computing and Applications, 33, 2561-2588.

[24] Shalev-Shwartz, S., Shamir, O., & Shammah, S. (2017, July). Failures of gradient-based deep learning. In International Conference on Machine Learning (pp. 3067-3075). PMLR.

[25] Sheikhpour, S., Sabouri, M., & Zahiri, S. H. (2013, May). A hybrid Gravitational search algorithm—Genetic algorithm for neural network training. In 2013 21st Iranian Conference on Electrical Engineering (ICEE) (pp. 1-5). IEEE.

[26] Kirkpatrick, S., Gelatt Jr, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. science, 220(4598), 671-680.

[27] Talbi, E. G. (2009). Metaheuristics: from design to implementation. John Wiley & Sons.

[28] Kaveh, M., & Mesgari, M. S. (2023). Application of meta-heuristic algorithms for training neural networks and deep learning architectures: A comprehensive review. Neural Processing Letters, 55(4), 4519-4622.

[29] Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. IEEE transactions on evolutionary computation, 1(1), 67-82.

[30] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929.

[31] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.