Meta-Heuristics vs. Backpropagation: A Fresh Look at CNN Model Parameter Optimization for Image Classification

Utkarsh Mathur
umathur, Pixel Swarm
CSE 573 Project Milestone 3
University at Buffalo
umathur@buffalo.edu

Mahammad Iqbal Shaik
mahammad, Pixel Swarm
CSE 573 Project Milestone 3
University at Buffalo
mahammad@buffalo.edu

***Abstract*—Most of the Neural Network models are trained using gradient based backpropagation technique to optimize model parameters which are prone to be stuck at local optimal value rather than reaching the globally optimal parameters. There are various techniques to improve the simple Stochastic Gradient Descent (SGD) like Learning Rate Scheduling and Momentum, but these techniques does not resolve the aforementioned limitation. In this project, we aim to compare Backpropagation with Meta-Heuristic Optimization Algorithms by analyzing their performance on training CNN models for Image Classification tasks. There are a few population-based meta-heuristic algorithms like Particle Swarm Optimization (PSO) and Grey Wolf Optimization (GWO) which can achieve globally optimal parameters and so, we aim to check the feasibility of such optimization techniques in CNN model architectures. https://github.com/datamathur/pixel_swarm_CV_project**

I. Introduction

Image Classification is a very important task in the field of Computer Vision as it was the first predictive task in an otherwise analytic field. In 1998, Yann LeCun et al. introduced the first Convolutional Neural Network LeNet-5 which introduced the concept of convolution layer and marked as the beginning of modern day Deep Learning. Over the years, various advancements in the field has significantly improved the quality of Image Classification models owing to the increment in research and computational resources.

The training of such Image Classification models as well as other Deep Learning models predominantly uses gradient based backpropagation technique. These optimizers have been extremely efficient in training Deep Learning models as well as generalization over test data. However they have the tendency to converge at local stable points (point of local minima) rather than converging at the global stable points. To address this issue, we decided to experiment with other optimization techniques in order to find a better optimization algorithm for Deep Learning paradigm.

Meta-heuristic algorithms are famous for finding global optimal solutions in low-dimensional complex spaces. Our aim here is to use some of the meta-heuristic algorithms to train Convolutional Neural Network and compare it's results with backpropagation so as to better understand the critical and unique role it has in the field of Deep Learning.

In our experiments, we have used Particle Swarm Optimizer (PSO) and Grey Wolf Optimizer (GWO) to train LeNet-5 model on MNIST and CIFAR-10 datasets and compared their results with that of Stochastic Gradient Descent (SGD). There are other meta-heuristic algorithms like Genetic Algorithm (GA), Ant Colony Optimizer (ACO), Differential Evolution (DE), and Firefly Algorithm (FA) which have proven potent in various optimization tasks however, due to the shortage of time we focussed on comparing the aforementioned algorithms and the expirement can be extended to other algorithms as well.

The general theme here is to analyze how these meta-heuristic algorithms perform in high-multidimesional complex space as Deep Learning paradigm generally consists of millions of trainable parameters and has surpassed over a billion parameters in the more recent models.

II. Background

A. LeNet - 5

LeNet-5 was originally designed for the task of digit recognition, specifically for processing images from the MNIST database, which contains handwritten digits.

The architecture of LeNet-5 (Fig. 1) consists of seven layers, not including the input layer. The input for LeNet-5 is a 32x32 pixel image. The first layer is a convolutional layer with six feature maps or filters, each using a 5x5 kernel, which produces feature maps of size 28x28. This is followed by a subsampling layer, also known as an average pooling layer, that reduces the size of each feature map to 14x14.

Subsequent layers alternate between convolutional layers and subsampling layers, progressively reducing the dimensionality of the spatial data while increasing the depth of the feature maps. The final layers of the network are fully connected layers, culminating in a softmax classifier that outputs the probabilities of the image belonging to one of the ten digit classes.

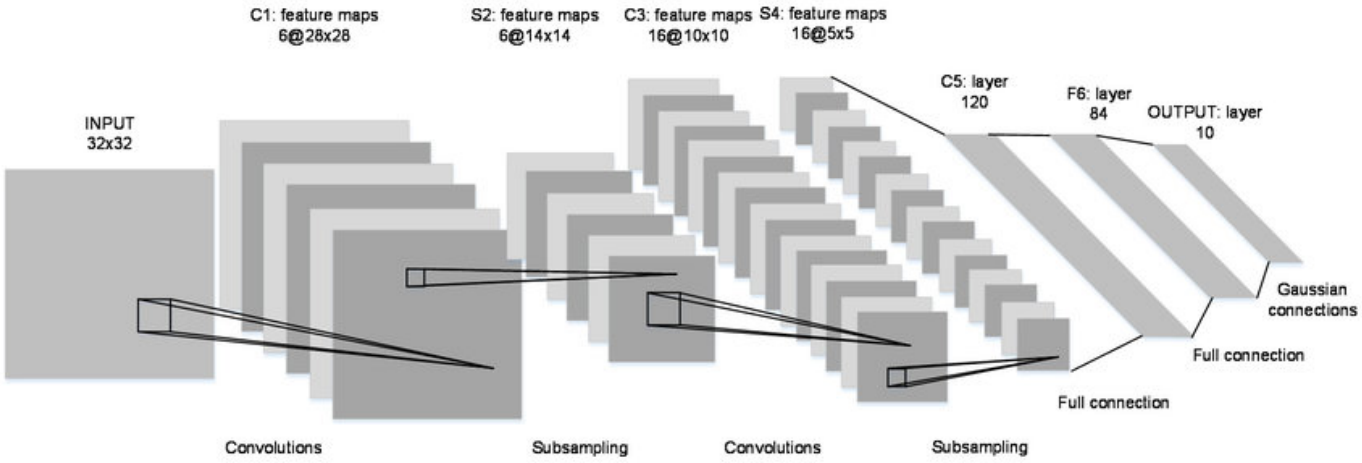


Fig. 1. LeNet Architecture

LeNet-5's design embodies several key innovations that have become standard in the design of CNNs for visual tasks, including the use of convolutional layers to capture spatial hierarchies, pooling layers to reduce spatial dimensionality, and fully connected layers to perform classification. The network's relatively simple structure and its effectiveness in digit recognition have made it a suitable choice for benchmarking optimization algorithms, such as the Particle Swarm Optimization (PSO) for the CIFAR dataset, which extends the principles of LeNet-5 to more complex image recognition tasks involving color and greater image detail.

B. Stochastic Gradient Descent

Gradient descent is a widely used optimization algorithm, particularly for neural networks. Modern Deep Learning libraries include optimization algorithms for gradient descent. These algorithms are commonly used as black-box optimizers due to the lack of clear explanations for their strengths and weaknesses.

Gradient descent is a way to minimize an objective function $J(\theta)$ parameterized by a model's parameters $\theta \in \mathbb{R}^d$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta)$ w.r.t. to the parameters. Stochastic Gradient Descent is one such widely used optimizer.

Stochastic gradient descent (SGD) performs a parameter update for *each* training example $x^{(i)}$ and label $y^{(i)}$:

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i)}, y^{(i)})$$

Because it only uses one training example at a time to update parameters, which leads to significant fluctuations in the objective function. These fluctuations enable SGD to break out of local minima and explore other, possibly better solutions, but they might complicate convergence to the precise minimum by frequently overshooting. It has been discovered that gradually lowering the learning pace helps to lessen this. By using a technique called simulated annealing, SGD can attain a local or global minimum, depending on the kind of optimization problem, and achieve convergence comparable to other gradient descent methods.

C. Particle Swarm Optimizer (PSO)

Particle Swarm Optimization (PSO) is a population-based meta-heuristic algorithm inspired by swarm behavior observed in nature such as fish and bird schooling. PSO is a simulation of a simplified social system. In nature, any of the bird's observable vicinity is limited to some range. However, having more than one birds allows all the birds in a swarm to be aware of the larger surface of a fitness function. Using this intuition, the swarm activity can be mathematically modelled as follows:

1. Randomly initialize a population of N particles X_i.
2. Select hyperparamter values of **inertial weight (w)**, **cognitive coefficient (c1)**, and **social coefficient (c2)**. In our experiment we applied Adaptive Particle Swarm Optimizer where these hyperparameters are controlled by the current iteration and total number of iterations

$$\begin{aligned} w &= 0.9 - 0.5 * (current\ iteration\ /\ total\ iterations) \\ c1 &= 3.5 - 3 * (current\ iteration\ /\ total\ iterations) \\ c2 &= 0.5 + 3 * (current\ iteration\ /\ total\ iterations) \end{aligned}$$

3. Follow the below algorithm to get optimal value:

Particle Swarm Optimization

0: Initialize swarm
0: **for** $Iter$ **in** $range(max\_iter)$ **do** {Loop max_iter times}
0: **for** i **in** $range(N)$ **do** {For each particle}
0: Compute new velocity of i^{th} particle:
0: $swarm[i].velocity = w \times swarm[i].velocity + r1 \times c1 \times (swarm[i].bestPos - swarm[i].position) + r2 \times c2 \times (best\_pos\_swarm - swarm[i].position)$
0: Compute new position of i^{th} particle using its new velocity:
0: $swarm[i].position += swarm[i].velocity$

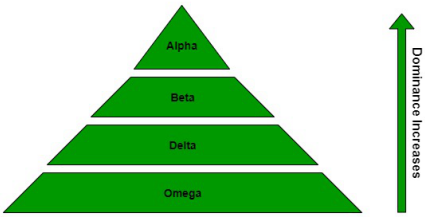


Fig. 2. Grey Wolf Social Hierarchy

0: If position is not in range $[minx, maxx]$ then clip it:
0: **if** $swarm[i].position < minx$ **then**
0: $swarm[i].position = minx$
0: **else if** $swarm[i].position > maxx$ **then**
0: $swarm[i].position = maxx$
0: **end if**
0: Update new best of this particle and new best of swarm:
0: **if** $swarm[i].fitness < swarm[i].bestFitness$ **then**
0: $swarm[i].bestFitness = swarm[i].fitness$
0: $swarm[i].bestPos = swarm[i].position$
0: **end if**
0: **if** $swarm[i].fitness < best\_fitness\_swarm$ **then**
0: $best\_fitness\_swarm = swarm[i].fitness$
0: $best\_pos\_swarm = swarm[i].position$
0: **end if**
0: **end for**
0: **end for**=0

4. Return the best positions amongst all particles. PSO has various advantages like it is derivative free, has very few hyperparamaters, is very efficient in global search and is insensitive to scaling of design variable. However, it is slow convergence in the refined search stage causing it to have weak local search ability or poor exploitation.

D. Grey Wolf Optimizer (GWO)

Grey wolf optimizer (GWO) is a population-based meta-heuristics algorithm that simulates the leadership hierarchy and hunting mechanism of grey wolves in nature, and it's proposed by Seyedali Mirjalili et al. in 2014. Grey wolves are very strict to their social hierarchy (Fig 2 [21]) which makes this algorithm very easy to design and execute.

Alpha (α) - It is the dominant wolf with best position (fitness value) amongst the pack. The pack follows alpha wolf. **Beta** (β) - It is the subordinate wolf with second best position (fitness value) amongst the pack and helps in decision-making along with **Alpha**. **Delta** (δ) - It is the wolf with third best position (fitness value) amongst the pack and they are responsible to keep **Omegas** in line while following orders from **alpha** and **beta**. **Omega** (ω) - They are the least significant members of the pack and are often used as spacegoats or baits.

The pack hunts by tracking, chasing, and approaching the prey following which the pack pursues, encircles, and harasses the prey until it stops moving. This is when the **alpha** and/or **beta** wolf attacks the prey. The aforementioned hunting technique is used to optimize parameters and can be mathematically modeled as follows:

Grey Wolf Optimization

0: Randomly initialize the population of grey wolves X_i $(i = 1, 2, \ldots, n)$
0: Initialize the value of $a = 2$, A, and C (using Eq. 3)
0: Calculate the fitness of each member of the population
0: X_α = member with the best fitness value
0: X_β = second best member (in terms of fitness value)
0: X_δ = third best member (in terms of fitness value)
0: **for** $t = 1$ to $Max\_number\_of\_iterations$ **do**
0: Update the position of all the omega wolves by Eq. 4, 5, and 6
0: Update a, A, C (using Eq. 3)
0: $a = 2(1 - t/T)$
0: Calculate Fitness of all search agents
0: Update X_α, X_β, X_δ
0: **end for**
0: **return** X_α =0

III. Methodology

Most of the deep learning libraries like PyTorch, TensorFlow, and MxNet donot have functions available for meta-heuristic algorithms. Therefore, we developed these optimizers for model training in PyTorch. The literature around meta-heuristic algorithms deals with execution of these optimizers for scalars and 1D vectors, however most of the Convolutional Neural Networks have parameters in 4-dimensional tensors. We address this by using model parameter and candidate parameters as collection of tensors depicting parameters of a single convolution layer.

Since we aimed to develop meta-heuristic optimizers for PyTorch, we inherited torch.optim.Optimizers module to create optimizers that functions similar to the in-built PyTorch optimizers like Stochastic Gradient Descent and Adam.

By defining a callable closure() function, we we able to pass a function to calculate loss associated with each set of parameters as and when required by the optimization process. This reduces memory usage by avoiding making unnecessary instances of model class.

A. Particle Swarm Optimization

The PSO class inherits Optimizers module from PyTorch and defines the step() function which is used in defining the process of selecting the best particle from a swarm of particles based on its fitness (loss) over the given model and dataset. These particles are defined by another class Particle which also has its own step() function whose aim is to perform one iteration of PSO on a single particle in the swarm.

B. Grey Wolf Optimization

The GWO class defines Grey Wolf Optimization for PyTorch models by inheriting Optimizers module from PyTorch. The step() in GWO function is used to define and perform one iteration of training based on the fitness (loss) of all the member of the pack. In the end we store the 3 best positions instead of just one as it is the requirement of the algorithm (this depicts the social hierarchy of wolfs).

C. Model Training

Models are trained in PyTorch library using the CPU as device and Critical Cross Entropy as loss function to optimize parameters of LeNet-5 architecture.

IV. Experiments & Results

A. Particle Swarm Optimization

We performed grid search hyperparameter tuning with 5 hyperparameters of PSO. Using the Adaptive Particle Swarm Optimization algorithm [23] [24] we were able to control 3 hyperparameter by using number of epochs as a hyperparamter (as discussed above in the Background):

1. **Population Size** (N): Population size determines the size of the swarm of particles. We assumed [10, 20, 50, 100] values for population size.

2. **Inertial Weight** (w): Inertial weight is a hyperparameter that determines the weightage or momentum carried by position and velocity of the particle at previous step. We used $0.9 - 0.5 * \frac{current\ epoch}{total\ epochs}$ values for inertial weight which is recommended in the range of 0.1-1.

3. **Cognitive Coefficient** (c1): It is the hyperparamter that determines the amount of influence of current position of the particle in the next position. We assumed $3.5 - 3 * \frac{current\ epoch}{total\ epochs}$ for cognitive coefficient which is recommended in the range of 1.0-2.

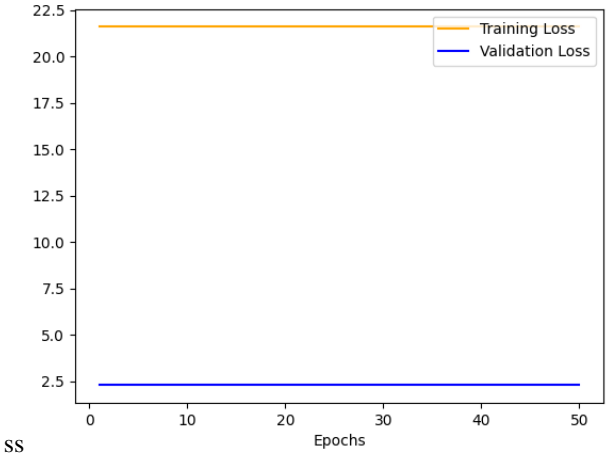
4. **Social Coefficient** (c2): This hyperparameter determines the amount of influence of the global best position (position of the fittest particle in the swarm) in the new position of the particle. We assumed $0.5 + 3 * \frac{current\ epoch}{total\ epochs}$ values for Social coefficient which is recommended in the range of 1.0-2.0.

By controling these 4 hyperparameters we wanted to experiment with search space exploration v/s exploitation tradeoff. However, the best validation accuracy for MNIST dataset was 26.37% (N=100, epochs=100) and for CIFAR-10 was 18.97% (N=50, epoch=100) which is very low in comparison to SGD optimizer. (Table I)

B. Grey Wolf Optimization

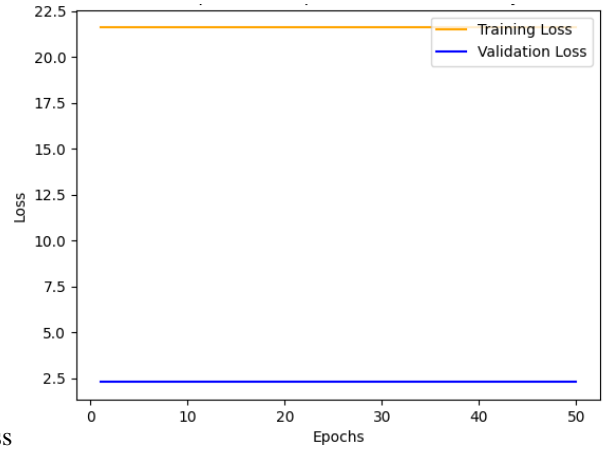
We performed Grid search Hyperparamter tuning with 2 hyperparamters for GWO:

1. **Encircling Coefficient** (a = 2*(1-(i/num_epochs)): This hyperparamter determines the rate at which each wolf in the pack encircles or searches the search space for solution. It is determined by the number of epochs such that its value decreases uniformly from 2 to 0 over the training process.



SS

Fig. 3. Loss vs Epochs: MNIST trained by PSO (N=100, epochs=100)



SS

Fig. 4. Loss vs Epochs: CIFAR-10 trained by PSO (N=100, epochs=100)

2. **Pack Size** (N): This hyperparamter determines the population size of the pack and is extremely crucial in the exploration and exploitation of search space.

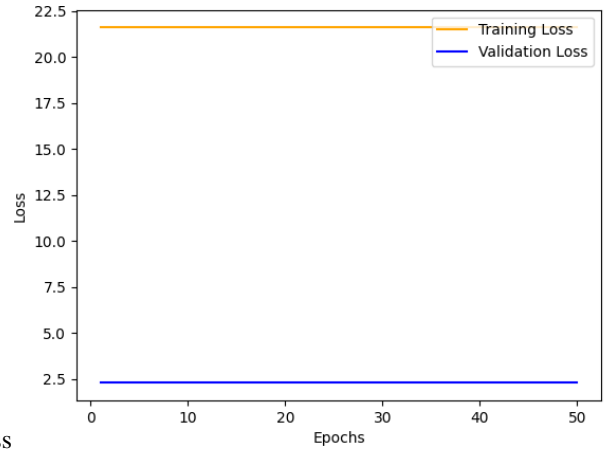
The best validation accuracy for MNIST dataset was 16.52% (num_epochs = 50, N = 5) and for CIFAR-10 was 12.35% (num_epochs = 20, N = 10) which is very low in comparison to SGD optimizer. (Table I)

TABLE I
Comparing Validation Loss for PSO, GWO, and SGD

Optimizer	MNIST		CIFAR10	
	Test Accuracy	Test Loss	Test Accuracy	Test Loss
PSO	14.29%	2.318	14.9%	2.312
	(N = 100, epochs=100)		(N = 100, epochs=100)	
GWO	9.69%	2.303	10.12%	2.303
	(N = 5, epochs=50)		(N = 10, epochs=20)	
SGD	98.99%	1.4722	64.4%	1.0433

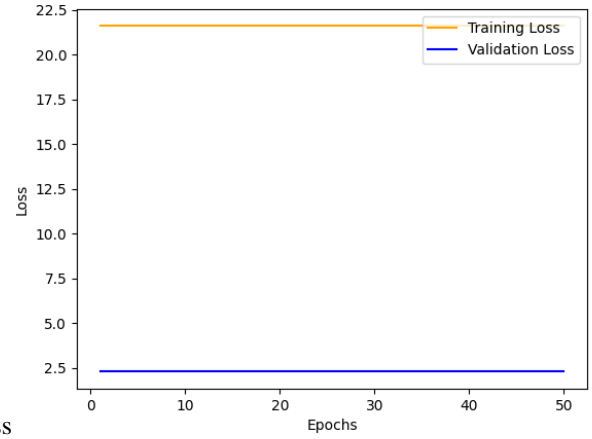
V. Observations Conclusions

After training the LeNet-5 model using PSO and GWO we made some observations: 1. For both PSO and GWO, the training loss and



ss

Fig. 5. Loss vs Epochs: MNIST trained by GWO (N=5, epochs=50)



ss

Fig. 6. Loss vs Epochs: CIFAR-10 trained by GWO (N=10, epochs=20)

validation loss hardly changed over the epoches however the parameter values were updated. This indicates that in high-dimensional search spaces (in the scale of thousands or million) the exploration capabilities of PSO and GWO face mammoth challenges. On the other hand, backpropagation avoid such bottlenecks. 2. The accuracy levels of PSO and GWO trained LeNet-5 models were very close to 10% which is the probability of correctly guessing the output of MNIST and CIFAR-10 images. Whereas, the accuracy of SGD over these dataset is significantly high. This indicated that due to lack of exploration, these algorithms were not able to exploit the explored space as well.

Through these experiments we can safely concur that backpropagation is a better algorithm for training CNN models. SGD and other gradient based methods have a distinctive advantage over metaheuristic optimizers that they consider atomic units of the models (namely convolution layers and dense layers) one at a time and so making transmission of feedback from loss, which is calculated at the very last layer, through the network visible at each layer. This encourage the similarity of feature processing in consecutive layers boosting the key characteristics of convolutional neural networks. Our implementation of metaheuristic algorithms to train the model parameters considers the entire set of parameters a single particle hence diminishing the defining characteristics of CNN and Deep Learning models. These algorithms are very powerful in finding globally optimal solution but in CNN paradigm they fail mainly because of difficulty of exploration and exploitation in high-dimensional search space. However, metaheuristic algorithms could be effective in performing Image Classification and Object Detection tasks under a different approach to the task where the search space is lower dimensional.

Apart from training the entire model, metaheuristic algorithm are often used to boost the performance of CNN models by employing them for hyperparameter tuning, weight initializations, and model architecture optimization [26] [27].

REFERENCES

[1] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996. doi: https://doi.org/10.1007/978-3-662-03315-9.

[2] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," Multimedia Tools and Applications, vol. 80, no. 5, Oct. 2020, doi: https://doi.org/10.1007/s11042-020-10139-6.

[3] J. Kennedy and R. Eberhart, "Particle swarm optimization," Proceedings of ICNN'95 - International Conference on Neural Networks, vol. 4, pp. 1942–1948, 1995, doi: https://doi.org/10.1109/icnn.1995.488968.

[4] T. Alam, S. Qamar, A. Dixit, and M. Benaida, "Genetic Algorithm: Reviews, Implementations, and Applications," Jun. 2020, doi: https://doi.org/10.48550/arxiv.2007.12673.

[5] A. G. Gad, "Particle Swarm Optimization Algorithm and Its Applications: A Systematic Review," Archives of Computational Methods in Engineering, vol. 29, no. 5, pp. 2531–2561, Apr. 2022, doi: https://doi.org/10.1007/s11831-021-09694-4.

[6] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," IEEE Computational Intelligence Magazine, vol. 1, no. 4, pp. 28–39, Nov. 2006, doi: https://doi.org/10.1109/MCI.2006.329691.

[7] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey Wolf Optimizer," Advances in Engineering Software, vol. 69, pp. 46–61, Mar. 2014, doi: https://doi.org/10.1016/j.advengsoft.2013.12.007.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, Jun. 2009, doi: https://doi.org/10.1109/cvpr.2009.5206848

[9] Li Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 141–142, Nov. 2012, doi: https://doi.org/10.1109/msp.2012.2211477.¡/li¿

[10] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," undefined, 2009, Accessed: May 11, 2022. [Online]. Available: https://www.semanticscholar.org/paper/Learning-Multiple-Layers-of-Features-from-Tiny-Krizhevsky/5d90f06bb70a0a3dced62413346235c02b1aa086¡/li¿

[11] 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, doi: https://doi.org/10.1109/5.726791.¡/li¿

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Communications of the ACM, vol. 60, no. 6, pp. 84–90, May 2012, doi: https://doi.org/10.1145/3065386.¡/li¿

[13] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv.org, Apr. 10, 2015. https://arxiv.org/abs/1409.1556

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," arXiv.org, Dec. 10, 2015. https://arxiv.org/abs/1512.03385

[15] R. Mohapatra, "rohanmohapatra/torchswarm," GitHub, Apr. 20, 2024. https://github.com/rohanmohapatra/torchswarm (accessed Apr. 20, 2024).
[16] A. P. Sansom, "Torch PSO," GitHub, Aug. 01, 2022. https://github.com/qthequartermasterman/torch_pso (accessed Apr. 20, 2024).
[17] H. Faris, "7ossam81/EvoloPy," GitHub, Apr. 20, 2024. https://github.com/7ossam81/EvoloPy/tree/master (accessed Apr. 20, 2024).
[18] Li Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 141–142, Nov. 2012, doi: https://doi.org/10.1109/msp.2012.2211477.
[19] A. Krizhevsky, "CIFAR-10 and CIFAR-100 datasets," Toronto.edu, 2009. https://www.cs.toronto.edu/ kriz/cifar.html (accessed Apr.20, 2024)
[20] "pytorch/pytorch," GitHub, Mar. 22, 2021. https://github.com/pytorch/pytorch (accessed Apr.20, 2024)
[21] "Grey wolf optimization - Introduction," GeeksforGeeks, Mar. 16, 2021. https://www.geeksforgeeks.org/grey-wolf-optimization-introduction/ (accessed Apr. 20, 2024).
[22] "Particle Swarm Optimization (PSO) - An Overview," GeeksforGeeks, Apr. 22, 2021. https://www.geeksforgeeks.org/particle-swarm-optimization-pso-an-overview/
[23] M. Clerc and J. Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space," IEEE Transactions on Evolutionary Computation, vol. 6, no. 1, pp. 58–73, 2002, doi: https://doi.org/10.1109/4235.985692.
[24] A. Agrawal and S. Tripathi, "Particle swarm optimization with adaptive inertia weight based on cumulative binomial probability," Evolutionary Intelligence, Nov. 2018, doi: https://doi.org/10.1007/s12065-018-0188-7.
[25] U. Khandelwal, "ujjwalkhandelwal/pso_particle_swarm_optimization," GitHub, Apr. 24, 2024. https://github.com/ujjwalkhandelwal/pso_particle_swarm_optimization/tree/main (accessed May 01, 2024).
[26] W.-C. Yeh, Y. Lin, Y.-C. Liang, and C.-M. Lai, "Convolution Neural Network Hyperparameter Optimization Using Simplified Swarm Optimization," Mar. 2021, doi: https://doi.org/10.48550/arxiv.2103.03995.
[27] S. K. Ladi, G. K. Panda, R. Dash, P. K. Ladi, and R. Dhupar, "A Novel Grey Wolf Optimisation based CNN Classifier for Hyperspectral Image classification," Multimedia Tools and Applications, Mar. 2022, doi: https://doi.org/10.1007/s11042-022-12628-2.