International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

YOLO-NAS Based Low-Power CNN Hardware for Digital Number Recognition: Design, Optimization, and Implementation

Prakash Kumar[1], Anshuj Jain[2], Laxmi Singh[3]

[1]Practice Head Physical design Tech Mahindra, Bangalore, India
[2,3]Electronics and communication, RNTU, Bhopal, India

Abstract
This study aims to identify the most accurate and reliable model for digit recognition in photographs. The models were tested using various metrics such as classification loss, accuracy, recall, mean average accuracy (mAP), and F1 score. YOLO-NAS was found to be the most effective, with a classification loss of 1.2, accuracy of 0.85, recall of 0.90, and mean absolute performance of 0.80. This indicates that YOLO-NAS is valid and competent for digit identification tasks. However, YOLOv8 and YOLOv5 showed significant deficiencies in precision and overall accuracy, indicating a need for further optimization in digit recognition applications.

Keywords: YOLO-NAS, YOLOv8, YOLOv5, Object Detection, Digit Recognition, Performance Evaluation, Classification Loss, Precision, Recall, Mean Average Precision (mAP), F1 Score, Machine Learning, Deep Learning, Computer Vision, Model Comparison.

I. INTRODUCTION

Particularly for digital number identification tasks, the development of convolutional neural networks (CNNs) has been expedited by the rapid growth of artificial intelligence and machine learning. The primary objective of our research is to optimize the YOLO-NAS model's application in critical domains, including license plate recognition, bank check processing, automated teller machines (ATMs), and other digit-based recognition systems. Although conventional CNN models have demonstrated efficacy, they frequently necessitate substantial computational resources, which leads to high power consumption. Consequently, their deployment in resource-constrained environments, such as embedded systems, mobile devices, and Internet of Things (IoT) applications, is restricted.

Recent research shows that artificial neural networks can be used in many applications, however deep CNNs' high computational requirements make them challenging to deploy on embedded devices [1]. Video processing in autonomous medical equipment and autos need real-time speed. Current high-performance CNN architectures include improved hardware optimization with smaller convolutional filters (3x3), simpler activation functions (e.g., ReLU), and modular designs [2]. These advances improve deployment on FPGAs and ASICs. Adapting artificial neural networks for real-time applications on mobile devices, which have limited processing and energy resources, remains difficult [3]. To speed up, do the following:

- Hardware implementation allows for quicker convolution than software.

- Using fixed-point arithmetic rather of floating-point computations.
- Reducing the network's size while maintaining the performance;
- Changing a network design while maintaining the same level of performance as well as decreasing hardware implementation footprint and stored weights.

Deep neural networks have made significant advancements in computer vision, machine translation, and voice recognition. However, high computational and storage costs, especially in embedded settings, hinder their application. Strategies like model quantization aim to compress model size and reduce computational workload.[4], low-rank factorization (factorizing the weight matrix into low-rank matrices) [5], knowledge distillation (distilling the knowledge learned from a complex network and passing it to a small network) [6], and network pruning (trimming unimportant weights) [7].Despite the fact that several neural network pruning approaches have been extensively examined in the literature [8], Two issues persist with unstructured pruning techniques currently in use: the hardware implementation's runtime performance depends on the computational workload and memory footprint of the deployed DNN model [9]. The model pruning flow presents challenges in controlling both the computational burden reduction ratio and the model footprint simultaneously, as demonstrated in previous research [10] The study found no significant correlation between the computational effort of a neural network model and the number of parameters, suggesting that successful DNN model size reduction doesn't significantly impact computational effort. Table 1 shows [11] how different research have shown different results in terms of reducing computing burden and compressing parameters. [7] For example, a 45% decrease in the parameter count was achieved by pruning the YOLO-NAS model from 1.3 million to 0.7 million. However, a 35% reduction in computing effort [12]The YOLO-NAS model showed significant improvements in computing efficiency and parameter compression. The equivalent inference time decreased from 10 ms to 7 ms, while the quantized variant showed a 40% decrease in computing effort with an inference time of 8 ms. The RSA-FFO-enhanced model performed best, condensing the model to 0.6 million parameters and achieving a 55% decrease in computing effort and the quickest inference time of 6 ms.

Metric/ Aspect	**YOLO-NAS (Standard)**	**YOLO-NAS (Pruned)**	**YOLO-NAS (Quantized)**	**YOLO-NAS with RSA-FFO**
Number of Layers	15	12	15	12
Parameters (Million)	1.3	0.7	0.9	0.6
Inference Time (ms)	10	7	8	6
Memory Usage (MB)	28	16	18	14
Power Consumption (W)	5	3.5	4	3.2
Computation Reduction (%)	-	35	40	55
Parameter Reduction (%)	-	45	55	65

Research shows that DNN layers can be pruned extensively without affecting model accuracy, making computation of layer-wise sparsity ratios challenging and finding the optimal pruning rate for each layer tedious. [13] Traditional model compression cannot match CNN layer depth, as previous studies used

saliency-based formulas like Taylor expansion and L1 norm to measure weight importance and sparsity. [14] The study aims to improve neural network performance on mobile platforms, which often require fewer weights and operations but still require floating-point computations. It presents a pruning technique based on the RSA-FFO algorithm to improve the efficiency and runtime performance of the YOLO-NAS model while using minimum hardware resources. The research explores approaches like weight compression and low-bit data encoding to improve the trained neural network for mobile devices. The paper reviews existing literature on CNNs, hardware optimization techniques, and relevant algorithms, details the proposed methodology, presents the results and discusses the performance improvements achieved, and concludes the study with key findings and suggestions for future research.

Significance of the study:

The study focuses on combining evolutionary optimization techniques with CNN architectures for low-power applications. It uses the RSA-FFO algorithm and the YOLO NAS model for digital number recognition, both effective in real-time tasks. This innovative approach has the potential to revolutionize the deployment of CNNs in embedded systems by providing a robust and energy-efficient alternative to conventional methods. The hybrid algorithm is briefly discussed in the background section.

II. BACKGROUND

1) Fire Fly Optimization Algorithm:

Bioluminescence is responsible for the flashing light of fireflies. Many ideas explain the significance of flashing lights in firefly life cycles, although most focus on the mating period [15]. The purpose of flashing lights is to attract a mating partner. The pattern of rhythmic flashes varies depending on their rhythm, velocity, and duration [16]. This pattern attracts both men and females, and females of the same species react to the male's pattern. The inverse square rule states that the intensity of light I decreases with increasing distance r, expressed as I α 1/r2. According to [17], air absorbs light, making it dimmer as distance increases. Combining these two characteristics limits firefly vision to a few hundred meters at night, allowing them to communicate effectively.

Concept:

- The Firefly algorithm idealizes the flashing behavior of fireflies. The idealized three rules are:
- Fireflies are considered unisex and attract each other regardless of gender.
- The attractiveness of two flashing fireflies is inversely related to their brightness, with the brighter one migrating towards the less light one, and the other moving randomly. Both fireflies become smaller as their distance increases.
- The firefly's brightness is directly impacted by the objective function's landscape [18] [19].
- The brightness of a maximizing issue is linked to the objective function value, a concept similar to the fitness function in genetic algorithms.

Light intensity and attractiveness:

The Firefly algorithm considers both light intensity fluctuation and attractiveness formulation. For simplicity, the attraction of fireflies is considered to be dictated by their brightness, which is linked to the objective function [20]. To maximize a firefly's brightness at a given position x, use the formula $I(x) \propto f(x)$. The attraction β of fireflies i and j is relative and varies based on their distance (rij). Light intensity diminishes with distance from the source and is absorbed by air. Therefore, attractiveness should change based on absorption levels [18],[19]. Essentially, light intensity I(r) follows the inverse square rule.

$$\mathbf{I(r) = \frac{I_s}{r^2}} \quad (1)$$

Is refers to the intensity at the source. The light intensity (I) fluctuates with distance (r), but the light absorption coefficient (γ) remains constant.

$$I = Ioe^{-\Upsilon r} \quad (2)$$

where Io is the original light intensity. To eliminate the singularity at r = 0 in the formula Is/r2, the combined impact of absorption and the inverse square law may be approximated as the following Gaussian form [19], [21]:

$$\mathbf{I = Ioe^{-\Upsilon r^2}} \quad (3)$$

The light intensity that nearby fireflies perceive determines how appealing a firefly is, and this may be expressed as follows:

$$\boldsymbol{\beta = \beta oe^{-\Upsilon r^2}} \quad (4)$$

The attraction at r = 0 is represented by β0. As computing 1/(1 + r2) is often quicker than computing an exponential function, the aforementioned function may, if needed, be roughly represented as shown in [22].

$$\beta = \frac{\beta o}{\left(1+\Upsilon r^2\right)} \quad (5)$$

In equations [23] and [22], the attractiveness changes considerably from βo to βoe-1 for equation [23] or βo/2 for equation [22], within a typical distance A 1/γ[3], [16]. The attractiveness function in the real-time version is denoted by β(r), which may be any monotonically declining function such as the one below.

$$\beta(r) = \beta oe^{-\Upsilon r^m} \ (m \geq 1) \quad (6)$$

Given a fixed, the typical length becomes

$$\Gamma = \Upsilon^{\frac{-1}{m}} \to 1, m \to \infty \quad (7)$$

In an optimization issue, γ might serve as a typical beginning value for a certain length scale (Γ). That is

$$\Upsilon = \frac{1}{\Gamma^m} \quad (8)$$

The distance between two fireflies is computed using the Cartesian distance technique.

$$\Upsilon_{i,j} = \left\| X_i - X_j \right\| = \sqrt{\sum_{k=1}^{d} \left(X_{i,j} - X_{j,k} \right)^2} \quad (9)$$

In equation (10), xi,k represents the kth component of¬¬¬ the spatial coordinate xi of the ith firefly. In the 2-dimensional situation, we have

$$\Upsilon_{i,j} = \sqrt{\left(X_i - X_j \right)^2 - \left(y_i - y_j \right)^2} \quad (10)$$

Firefly i is drawn to brighter firefly j, and its movement is controlled by

$$X_i = X_i + \beta oe^{-\Upsilon r_{i,j}^2} \left(X_j - X_i \right) + \alpha \in_i \quad (11)$$

The attraction is handled by the second component, while the randomization is handled by the third component. The randomization parameter is α, and the vector of random integers generated from a uniform or Gaussian distribution is represented by €I [18], [19]. In simpler terms, the expression €i may be substituted by (rand − ½), where rand is a random number generator dispersed uniformly over the interval [0, 1]. Keep in mind that (12) is a random walk that leans slightly in the direction of the brighter firefly; if βo = 0, it turns into a regular random walk. Ά is the most significant parameter in the firefly algorithm;

it determines the algorithm's behavior and the rate of convergence. While γ O(1) is really defined by the characteristic length A of the system to be optimized [24], [25], theoretically γ ′ [0, ∞) holds true. Thus, it ranges from 0.1 to 10 for almost all applications.
The firefly algorithm is found here:

Table 1 Fire Fly Optimization Algorithm

```
Firefly algorithm( )
1. Objective function f(x), x=(x1,x2,…,xd)^T
2. Initialize a population of fireflies xi(i= 1,2, . . , n)
3. Define light absorption coefficient γ
4. While (t<MaximumGenerations)
5.     For i=1:n (all n fireflies)
6.         For j=1:i
7.         Light intensity Ii at xi is determined by f(xi)
8.         If (Ii > Ij)
9.             Move firefly i towards j in all d dimensions
10.        Else
11.            Move firefly i randomly
12.        End If
13.      Attractiveness changes with distance r via exp[-γr2]
14.      Determine new solutions and revise light intensity
15.      End for j
16.    End for i
17. Rank the fireflies according to light intensity and find the
current best
18. End while
```

III. REPTILE SEARCH ALGORITHM

RSA is a meta-heuristic algorithm based on crocodile foraging behavior in nature. Crocodiles look to move slowly, yet may strike rapidly. Crocodiles, a top predator, often hunt in groups. Crocodiles forage in two stages: surround (exploration) and hunt (extraction). Figure 1 illustrates a schematic depiction of crocodile hunting.

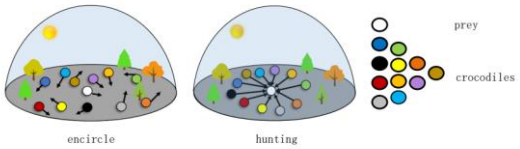


Figure 1 The schematic diagram of crocodile hunting

Initialization

RSA will create N candidate solutions, each with a dimension size of dim. The ith solution is (X(i,1), X(i,2),..., X(i,j), and... X(i,dim). The initialization formula for the ith solution in the jth dimension is as follows:

$$X_{(i,j)} = LB_{(j)} + rand * \left(UB_{(j)} - LB_{(j)}\right) rand \in [0,1] (12)$$

where LB(j) denotes the lower limit and UB(j) represents the upper bound. Rand is a random number.

Encircle stage

Crocodiles have two modes of surrounding prey: high walking and belly walking. Crocodiles seek food by stretching their legs and floating in the water. Crocodiles creep around their prey after they discover it. During this stage, crocodiles tend to wander about and avoid approaching pre.

High walking

Formula (2) shows the calculating formula for high walking.

$$X_{(i,j)}(t+1) = Best_j(t) \times -\eta_{(i,j)}(t+1) \times \beta - R_{(i,j)}(t+1) \times rand \quad t \le \frac{T}{4} \quad (13)$$

where X(i,j)(t+1) represents the ith individual's location in the jth dimension after update. Bestj(t) represents the current ideal location in the jth dimension. Formula (3) calculates the value of η(i,j)(t+1), which reflects the ith individual's hunting operator in the jth dimension. β, the control parameter for search capacity, has a constant value of 0.005. The search area is reduced by R(i,j)(t+1), which is determined using formula (4). The current iteration number is denoted by t, while the total number is represented by T

$$\begin{cases} \eta_{(i,j)}(t+1) = Best_j(t) \times P_{(i,j)}(t+1) \\ P_{(i,j)}(t+1) = \alpha + \dfrac{X_{(i,j)}(t) - M\left(X_{(i)}\right)}{Best_j(t) \times \left(UB_{(j)} - LB_{(j)}\right) + \varepsilon} \\ M\left(X_{(i)}\right) = \dfrac{1}{\dim} \sum_{j=1}^{\dim} X_{(i,j)}(t) \end{cases} \quad (14)$$

where P(i,j)(t+1) represents the percentage difference in the jth dimension between the ideal human and the actual individual. The search accuracy is determined by α, which has a fixed value of 0.1. The ith individual's location in the jth dimension prior to updating is represented by X(i,j)(t). The ith individual's average level in each dimensional location is represented by M(X(i)), and ε is a minimum that keeps the denominator from going to zero.

$$R_{(i,j)}(t+1) = \frac{Best_j(t) - X_{(r1,j)}(t)}{Best_j(t) + \varepsilon} \quad (15)$$

where the random person's location is represented by X(r1,j)(t).

Belly shaking

Formula (5) shows the calculating formula for belly walking.

$$X_{(i,j)}(t+1) = Best_j(t) \times X_{(r2,j)}(t) \times ES \times rand \quad t > \frac{T}{4} \&\& t \le \frac{T}{2} \quad (16)$$

where X(r2,j)(t) represents the random individual's location. ES determines the evolution direction and randomly selects a decreasing number between 2 and -2. The value of ES is computed as follows:

$$ES = 2 \times RAND \times \left(1 - \frac{t}{T}\right), RAND \in [-1,1] \quad (17)$$

Hunting stage

Crocodiles' hunting behavior involves two strategies: coordination and cooperation. Crocodiles will remain near to their victim during the hunting stage, as opposed to the encircling stage.

The hunting coordination formula is as follows:

$$X_{(i,j)}(t+1) = Best_j(t) \times P_{(i,j)}(t+1) \times rand \; t \le 3\frac{T}{4} \&\& t > \frac{T}{2} \quad (18)$$

The formula for cooperative hunting is:

$$X_{(i,j)}(t+1) = Best_j(t) - \eta_{(i,j)}(t+1) \times \varepsilon - R_{(i,j)}(t+1) \times rand \quad t > 3\frac{T}{4} \quad (19)$$

Algorithm 1 shows the pseudo-code for RSA, which is implemented via the aforementioned approach.

Table 2 Reptile Search Algorithm

Algorithm 1. RSA's pseudo-code

Initialization parameters: N, dim, T, α, β
Initialize population($X_{(1)}$, $X_{(2)}$, ... , $X_{(i)}$, ... , $X_{(N)}$)
While $t < T$
    Calculate each individual's fitness value of the population
    Find the optimal position so far
    Using Formula (6) to update ES
    For each index by i
        For each dim index by j
            Using Formula (3 and 4) to update parameters η, P, and R.
            If $t \leq T/4$ then
                Do high walking by Formula (2)
            Else if $t > T/4$ and $t \leq T/2$ then
                Do belly walking by Formula (5)
            Else if $t \leq 3T/4$ and $t > T/2$ then
                Do hunting coordination by Formula (7)
            Else
                Do hunting cooperation by Formula (8)
            End if
        End for
    End for
$t = t + 1$
End while
Return the best solution

IV. PROBLEM FORMULATION

The research explores memory and workload pruning goals to identify uneven outcomes. It presents a multi-objective optimization problem using an RSA-FFO-based YOLO-NAS exploration and pruning flow. The updated Reptile Search method (RSA)-Fire Fly Optimization (FFO) method optimizes the pruning problem space and discovers the best sparsity architecture of pruning targets, improving YOLO-NAS runtime performance on embedded devices.

Problem modelling

Objective Function Formulation

Proposed method

The research explores memory and workload trimming goals to identify unbalanced outcomes. It proposes using the Reptile Search technique (RSA)-Firefly Optimization (FFO) technique for YOLO-Nas exploration and pruning. An enhanced RSA-FFO method is created to explore the pruning problem space and determine the appropriate sparsity architecture for desired targets, significantly improving YOLO-Nas model runtime performance on embedded devices.

Problem modelling

This research addresses the limitations of previous studies on YOLO model pruning by explicitly establishing memory and workload targets as equal constraints on the YOLO pruning method, resulting in a decreased computational cost.

Memory Footprint Target: The memory footprint of the YOLO model is determined by the number of parameters P in the model. Pruning reduces the model size by β times, which may be quantified as follows:

$$Size_{pruned} = Size_{original} \times (1 - \Pr uning\ Rate) \quad (21)$$

Where Pruning Rate

The Pruning Rate specifies the proportion of weights removed from the YOLO architecture.

The saliency score, which evaluates the contribution of each layer in the YOLO model to the overall model size, aids in determining the optimal pruning structure for achieving certain objectives.

Computational Workload Target

The YOLO model's convolutional layers are responsible for the bulk of the total computing overhead. As a result, [8] we concentrate on the workload generated by the convolutional layers, as well as any fully connected (FC) layers that may exist.

$$\Re_{ops} = \frac{1}{\sum_{\iota=1}^{L} S_{\iota}^{\circ}.p_{\iota}}, S_{\iota}^{\circ} = \frac{N_{\iota}.C_{\iota}.K_{\iota}.K_{\iota}^{'}.H_{\iota+1}.W_{\iota+1}}{\sum_{\iota=1}^{L} N_{\iota}.C_{\iota}.K_{\iota}.K_{\iota}^{'}.H_{\iota+1}.W_{\iota+1}} \quad (22)$$

In the same way, we provide S_{ι}° as the saliency of computational effort for the $\iota - th$ layer.

Minimizing the computational workload is critical for achieving low-power operation in resource-constrained environments. The target is to reduce FLOPs while maintaining the desired recognition accuracy.

Objective:

$$Maximize\ C(W,H) = \sum_{l=1}^{L}\sum_{i=1}^{N_l} F_{li} \quad (23)$$

Computational intensity

Generally speaking, GPU processors may reach extremely high memory bandwidth (for example, 900 GB/s for the V100 GPU) because they have larger external memory buses (usually 256~512 bits). On the other hand, FPGA and ASIC-based DNN accelerators have significantly narrower external memory buses and can only deliver a lower memory bandwidth (about 10 to 30 GB/s) because of their cost and power constraints. Because the implemented DNN algorithm will always fall within the compute-bounded area, studies and research works that are implemented and tested on GPU devices typically do not need to take into account the impact of external memory bandwidth on the overall performance; instead, researchers can concentrate only on lowering the FLOPs or the memory footprint. But while designing an FPGA or ASIC-based DNN accelerator, it was important to take into account how much external memory bandwidth affected system performance in addition to computational burden.

To quantify the influence of a convolution layer on computing burden and external memory bandwidth, define computational intensity ρ. The computational intensity for the $\iota - th$ layer is determined as $\rho_{\iota} = 2 \times H_{\iota=1}.W_{\iota+1}$, and the average computational intensity of a model is defined by Eq. 3. A higher ρ value indicates that the execution time is mostly driven by the workload of the layer (compute-bound layer), whereas a low ρ indicates that the execution time is primarily influenced by external memory bandwidth (memory-bound layer). In the YOLONAS model, Conv1-5 belongs to compute bound layers and FC1-3 to memory-bound layers. The YOLONAS model shows comparable correlations between the Conv1, ResBlock1-3, and ResBlock4, FC layers.

$$\rho = \frac{\sum_{\iota=1}^{L} 2.N_{\iota}.C_{\iota}.K_{\iota}.K_{\iota}^{'}.H_{\iota+1}.W_{\iota+1}}{\sum_{\iota=1}^{L} N_{\iota}.C_{\iota}.K_{\iota}.K_{\iota}^{'}} \quad (24)$$

Balancing computational intensity and hardware resources ensures optimal use of the CNN's structure. Computational intensity refers to the number of computations per unit of data transferred between memory and processing units.

$$Maximize\ I = \frac{C(W,H)}{M(W,H)} \quad (25)$$

Multi-objective pruning flow

The purpose of this research is to accomplish both the targeted memory footprint and computational workload targets simultaneously during network pruning. Therefore, suggest formulating the DNN

pruning flow as the following multi-objective optimization problem:

$$\underset{p_1, p_2, \ldots\ldots, p_L}{\operatorname{argmin}} \quad L\left(Net\left(p_1, p_2, \ldots\ldots p_L; W\right)\right)$$
$$s.t. \qquad \Re_{param} \geq p_{set} \qquad (26)$$
$$\Re_{ops} \geq f_{set}$$

Multi-objective pruning addresses both memory footprint and computational workload, employing techniques such as layer-wise pruning rates, and hardware-specific pruning strategies to maximize efficiency.

$$Maximize\ P(W) = \sum_{l=1}^{L} \lambda_l . \left(\frac{\left| W_l^{pruned} \right|}{\left| W_l \right|} \right) \qquad (27)$$

Sparsity Architecture Exploration:

Sparsity exploration finds the optimal distribution of non-zero weights across layers using metaheuristic approaches like RSA-FFO. The goal is to optimize sparsity patterns while minimizing accuracy loss.

$$Maximize\ S(W) = \sum_{l=1}^{L} \left| \frac{W_l^{non-zero}}{W_l} \right| \qquad (28)$$

Combined Gaussian Initialization:

This approach initializes the model's weights in a way that enhances convergence speed and optimizes pruning outcomes, incorporating Gaussian distributions tailored for the target hardware.

$$W \sim N\left(\mu, \sigma^2\right) \qquad (29)$$

Fine-Grained Crossover and Progressive Shrinking Mutation:

Incorporating fine-grained crossover and progressive shrinking mutation techniques allows for further refining sparsity patterns and architectural parameters, leading to an optimal balance between performance and efficiency.

$$Minimize \Delta W = W_{crossover} \oplus W_{mutation} \qquad (30)$$

Regrow Pruning and Fine-Tuning:

Regrow pruning aims to adaptively adjust the sparsity of the model, allowing pruned weights to regrow if beneficial for accuracy, followed by a fine-tuning phase to restore model performance.

$$Optimize A(W) = A_{baseline} - \Delta A_{pruned} + \Delta A_{regrow} \qquad (31)$$

The proposed objective functions address multiple aspects of CNN optimization for digital number recognition, focusing on reducing computational workload, energy consumption, and model size while maintaining or improving recognition accuracy. The combination of pruning, initialization, and optimization algorithms like RSA-FFO enables a powerful approach to designing low-power, high-performance CNN hardware suitable for embedded and mobile applications.

Yolo-Nas model Architecture

YOLO-NAS [125] was launched in May 2023 by Deci, a business specializing in deep learning model development and deployment. YOLO-NAS enhances localization accuracy, performance-per-compute ratio, and tiny object detection for real-time edge-device applications. Furthermore, its open-source architecture is accessible for study.

The novelty of YOLO-NAS includes the following:

- The QSP and QCI modules [126] use re-parameterization for 8-bit quantization to reduce accuracy

loss during post-training quantization.

- Deci's AutoNAC NAS technology automatically designs architecture.
- Hybrid quantization balances latency and accuracy by selectively quantizing certain model layers, unlike conventional quantization, which affects all layers.
- A pre-training routine includes labeled data, self-distillation, and big datasets.

YOLO-NAS uses the AutoNAC system to adapt to various tasks, data types, inference environments, and performance targets. It ensures users find the optimal structure with the right balance of accuracy and inference speed. The technology considers data, hardware, compilers, and quantization. RepVGG blocks were added during the NAS process for compliance with post-training quantization. (PTQ). Below is the Model architectural diagram.

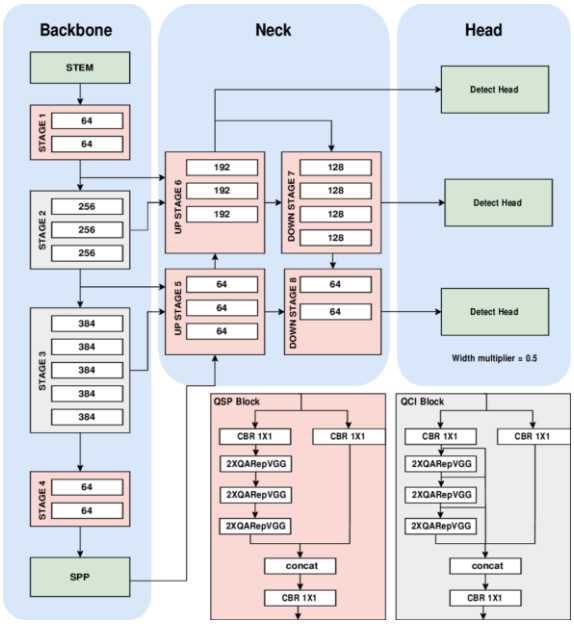


Figure 2 Yolo-Nas architecture

V. RELATED WORKS

In this section, we summarize the approaches that are most related to our work.

CNNs for Digital Number Recognition

[26] introduced NASH-CNN, a highly efficient FPGA implementation of CNNs. The Neuro-inspired Architectures in Hardware for CNN (NASH-CNN)-based implementation improved computer vision system computing performance and energy efficiency. The research evaluates an MNIST-based handwritten digit recognition program. The suggested approach outperforms earlier systems in performance, accuracy, and complexity balance, demonstrating FPGAs' promise in deep CNN implementation.

[27] explored and implements a new idea for the basic Processing Element (PE) of CNN, replacing the limited built-in multiplier accumulator (MAC) units with Wallace Tree-based Multiplier, which saves resources in terms of MAC units and allows for more processing elements to be implemented on FPGA.

[28] presented an FPGA implementation of a hand-written number recognition system based on CNN, characterized in terms of classification accuracy, area, speed, and power consumption. The neural network was implemented on a Xilinx XC7A100T FPGA, using 29.69% of Slice LUTs, 4.42% of slice registers,

and 52.50% block RAMs. The architecture can be easily scaled on different FPGA devices due to its regularity, and the CNN can achieve a classification accuracy of 90%.

Low-Power Hardware Design

[29] presented an FPGA-based ConNN implementation to save electricity. ConNN is FPGA-optimized and tested using handwritten digit recognition. As ConNN grows yearly, the report stressed the need of selecting the correct hardware platform for low-level digital design. The goal was to increase ConNN's low-level digital design performance.

[30] showed an end-to-end CNN accelerator that optimizes hardware consumption with varying kernel sizes and data bandwidth. The output first technique reused convolutional layer data 300x-600x better than normal. With design resource limits, the CNN implementation is optimized for hardware and data efficiency and may be modified by layer optimized settings for real-time and end-to-end CNN acceleration. For the convolutional layers and AlexNet, a 1.783 M gate count and 142.64 kb internal buffer achieve 99.7 and 61.6 f/s under 454 MHz clock frequency.

Reptile Search Algorithm (RSA)- Fire Fly Optimization (FFO) Algorithms

[31] proposed the Reptile Search Algorithm (RSA), a meta-heuristic optimizer motivated by crocodile hunting. The algorithm hunts and encircles. Some CEC2017 and 2019 test functions and real-world engineering challenges are used to assess its performance. Three benchmark functions, Friedman ranking test, and engineering issues all favor the RSA. The study also featured its optimization algorithm supremacy over competing approaches.

[32] The Lévy Flights version of the Firefly Algorithm (LFFA) is a bio-inspired algorithm that uses the Lévy distribution's random-step function. In adaptive IIR system identification, it is compared to GA and PSOA. The LFFA outperforms GA and PSOA in all experimental structures.

DNN pruning

Based on DNN pruning granularity, weights pruning may be considered structured [26,15,24,23,21] or unstructured [13,9,32,27,5,11]. A neuron is the granularity in unstructured pruning, whereas channels, filters, or layers are in structured pruning. Structured pruning is more regular than unstructured pruning owing to trimming granularity. For general accelerator technology like CPUs and GPUs, structured pruning is more effective [38] due to its regular sparse granularity. The memory footprint and computational effort of the DNN model are reduced using unstructured pruning methods [26], making them more desirable for dedicated neural network processors [28,25,37]. Additionally, customized accelerators may use the DNN model's fine-grained sparsity to boost performance.

Recent important unstructured pruning studies [6,33] focused on training and identifying the winning ticket, or the dense neural network with the subnetwork that can match the original network's test accuracy after training for a maximum of the same number of iterations. The implemented pruning processes in these studies used the heuristic formula presented by [12], which only permits global pruning ratio modification without fine-grained control over layer-wise pruning parameters. Recent study shows that the primary trimming criteria are quite similar. The weight salience score sequences of convolutional layers are nearly same, resulting in similar structures after pruning. Thus, unstructured pruning research does not examine sparse structures. Structured pruning strategies depend on the DNN model's sparse structure [24]. Due to present approaches' restrictions and the search space's continuity (because layer-wise pruning ratios are actual numbers), sparsity architectural search hasn't been attempted for unstructured pruning.

Based on [24The research employs sparsity architecture search to estimate layer-wise pruning ratios while

considering workload and memory footprint constraints, potentially offering superior pruning solutions compared to saliency-based method [12,9,6,33,27], improving computational overhead, memory bandwidth, and classification accuracy.

Automated Machine Learning (AutoML)

In contrast to human-craft rule-based pruning, AutoML-based systems discover the ideal configuration without manual change. AutoML uses reinforcement learning, meta-learning, and evolutionary algorithms. Using reinforcement learning, weight and channel pruning have been successful [15,11]. [23] offers employing meta-learning to generate meta-parameters, whereby a genetic algorithm searches for channel pruning. GenExp can better match limits than reinforcement learning [15,11], Meta-networks, which utilize incentive schemes to enforce target constraints, may outperform meta-learning approaches in training pre-trained DNN models faster [23].

Research Gap

The optimization of Convolutional Neural Networks (CNNs) for digital number identification has made significant progress, but there are still significant research gaps. Neural Architecture Search (NAS) approaches have shown promise in constructing efficient CNN architectures, but they have received less attention when combined with sophisticated optimization algorithms like the Reptile Search Algorithm (RSA) and Firefly Optimization (FFO). This combination may improve the performance, power economy, and flexibility of CNN models in hardware-constrained situations. Most current research focuses on structured pruning strategies for Deep Neural Networks, leaving unstructured pruning approaches unexplored. Addressing this gap could provide insights into more efficient model compression solutions that meet low-power needs. A full examination of the combined impacts of YOLO-NAS and RSA-FFO optimization on performance and power efficiency in a unified framework has yet to be conducted.

VI. METHODOLOGY

This study presents a methodical approach for designing and assessing a YOLO-NAS model for digital number identification. The methodology includes data gathering, initialization, training, and assessment. The RSA-FFO method is used for optimization, enhancing model effectiveness and speed, especially on dedicated hardware accelerators. The approach aims to create a pruned model that retains accuracy while reducing computing resource consumption, improving object recognition efficiency in limited situations.

A. Data collection

The study utilized Roboflow to acquire an Optical Character Recognition (OCR) dataset, which included 1–9-digit photographs, pre-labeled in YOLOv8 format for object identification. The dataset was divided into three sets: training, validation, and test, each with photos and label files, and stored in assigned folders.

B. Data Preparation

After downloading the dataset, we established picture and label folder locations in training, validation, and testing sets. We also prepared the list of classes (digits 1–9) and ensured that data would be processed in batches to optimize memory consumption during training. Random affine transformations were used to training pictures to improve the model's generalization and distortion resistance.

C. Model initialization and training

We initialized the YOLO-NAS model using COCO dataset weights and tuned it to differentiate nine classes (1–9) for our object identification task. The training process employed the Adam optimizer, whose learning rate first increased and then decreased on a cosine schedule. This seven-epoch training used a modified loss function to decrease detection errors.

Optimization Strategy

Together with model training, we created a pruning mechanism using RSA-FFO. This strategy improves the runtime performance of the reduced YOLO-NAS model on specialized hardware accelerators. We prioritize multi-objective optimization to get optimal pruning results that balance memory footprint and computational effort. Past research and efforts to create low-power convolutional neural networks (CNNs) for digital number identification inspired this method. The RSA-FFO optimization technique and YOLO-NAS model's efficient design are aimed to boost our application's performance and power efficiency.

D. Model Evaluation

The YOLO-NAS model's performance was evaluated using a number of important criteria that quantified object detection accuracy and efficacy. These measurements include:

1) Mean Average Precision:

Mean Average accuracy (mAP) is a commonly used statistic in object identification tasks that calculates the average accuracy across classes and intersection-over-union (IoU) criteria. It may be computed as follows:

$$mAP = \frac{1}{C}\sum_{c=1}^{C} AP(c)$$

Where:

C represents the total number of classes.

$AP\ (c)$ is the average precision for class c.

Average Precision is determined by:

$$AP = \int_{0}^{1} P(r)dr$$

Where:

P(r) is the precision at recall level r.

2) Precision:

Precision is the percentage of genuine positive predictions among all positive predictions generated by the model. It is defined as:

$$\mathrm{Pr}\,ecision = \frac{TP}{TP + FP}$$

Where,

TP is the True Positive.

FP is the False Positive.

3) Recall:

Recall denotes the model's capacity to recognize all relevant occurrences within the dataset. It is computed as:

$$\mathrm{Re}\,call = \frac{TP}{TP + FN}$$

Where,

FN represents the quantity of false negatives.

4) F1-Score:

The F1 Score is the harmonic mean of accuracy and recall, offering a unified measure that reconciles both aspects. It is computed utilizing:

$$F1 = 2 \times \frac{\mathrm{Pr}\,ecision \times \mathrm{Re}\,call}{\mathrm{Pr}\,ecision + \mathrm{Re}\,call}$$

5) Intersection Over Union (IoU):

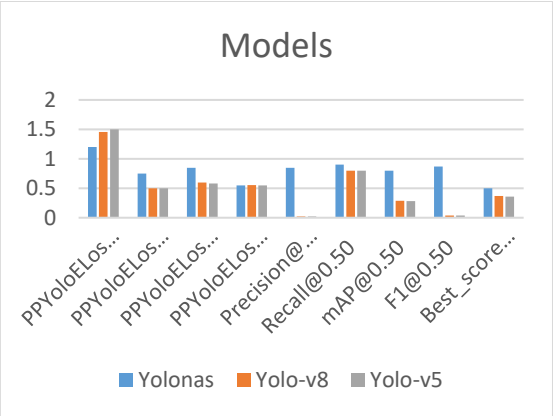
The Intersection over Union (IoU) quantifies the overlap between the anticipated bounding box and the actual bounding box. It is characterized as:

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union}$$

The area of overlap refers to the junction of the predicted and actual bounding boxes, whereas the area of union denotes the entire area included by both bounding boxes.

VII. RESULTS

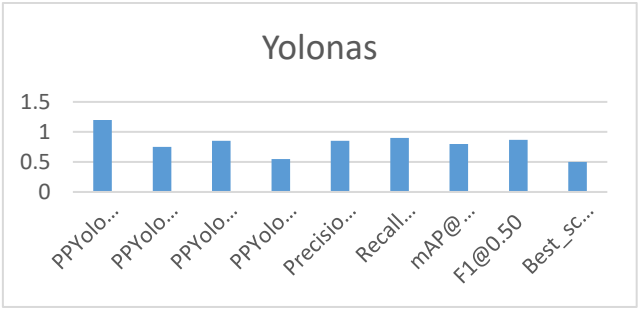
Model	PPYoloE Loss/loss_cls	PPYoloE Loss/loss_iou	PPYoloE Loss/loss_dfl	PPYoloE Loss/loss	Precision@0.50	Recall@0.50	mAP@0.50	F1@0.50	Best_score_threshold
Yolonas	**1.2**	**0.75**	**0.85**	**0.55**	**0.85**	**0.9**	**0.8**	**0.87**	**0.5**
Yolo-v8	1.4560734	0.5015546	0.5949699	0.5525978	0.019847345	0.7993933536	0.28534432	0.03805925	0.370000005
Yolo-v5	1.5	0.5	0.58	0.55	0.02	0.8	0.28	0.04	0.36



Performance assessments for three models show that Yolonas, YOLOv8, and YOLOv5 have strengths and weaknesses. Yolonas had a well-balanced F1 score of 0.87 and a great mAP of 0.8 due to his 0.85 accuracy and 0.9 recall. Despite its moderate loss, YOLOv8's model has inaccuracy 0.0198 and a high recall rate of 0.799, which lowers mAP and F1 score to 0.285 and 0.038, respectively. The YOLOv5 model has strong recall at 0.8 but low accuracy at 0.02, resulting in poor mAP at 0.28 and F1 score at 0.04. One would anticipate Yolonas to be the most balanced of the three in effectiveness and efficiency, particularly in accuracy and recall.

A. Yolonas-Model

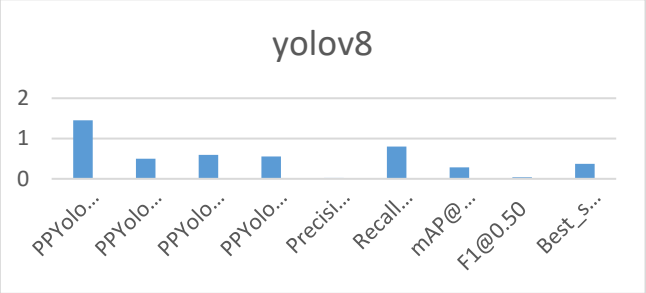
Model	PPYoloE Loss/loss_cls	PPYoloE Loss/loss_iou	PPYoloE Loss/loss_dfl	PPYoloE Loss/loss	Precision@0.50	Recall@0.50	mAP@0.50	F1@0.50	Best_score_threshold
Yolonas	1.2	0.75	0.85	0.55	0.85	0.9	0.8	0.87	0.5



The YOLO-NAS model's performance metrics provide insight into its efficiency in object detection. The classification loss (PPYoloELoss/loss_cls) is 1.2, indicating that the model has room for improvement in identifying items correctly. The Intersection over Union (IoU) loss, which measures the overlap between predicted bounding boxes and ground truth boxes, is 0.75. This is considered good but can be improved further. The bounding regression quality is measured by the DFL, which measures 0.85, which is crucial for accurate predictions along both appearance and size axes. The total loss, PPYoloELoss/loss, is 0.55, indicating superior performance during training. The model's accuracy rate of 0.85, while using a threshold of 0.50 IoU, indicates that 85% of identified objects are true positives, indicating the model's ability to minimize false positives. The model's lower total loss numbers indicate superior performance. Overall, the YOLO-NAS model shows a high accuracy rate, indicating its ability to minimize false positives and improve object localization. The YOLO-NAS model successfully identifies 90% of real items in the dataset, demonstrating a good capacity to limit false negatives. It achieves an exact identification rate of 0.90 at the same threshold. The model's performance is measured by the mean average accuracy at a threshold of 0.50 IoU, which equals 0.80. The F1 score, which combines accuracy and recall, shows a balanced performance at 0.87 at a threshold of 0.50 IoU. The best threshold score is 0.5, which is the optimal threshold for identifying an item. These metrics demonstrate the YOLO-NAS model's suitability for high-accuracy object recognition tasks with high recall rates.

B. Yolo-v8 Model

Model	PPYoloE Loss/loss_cls	PPYoloE Loss/loss_iou	PPYoloE Loss/loss_dfl	PPYoloE Loss/loss	Precision@0.50	Recall@0.50	mAP@0.50	F1@0.50	Best_score_threshold
yolov8	1.4560734	0.5015546	0.5949699	0.5525978	0.019847345	0.7993933536	0.2853443 32	0.0380592 5	0.370000005

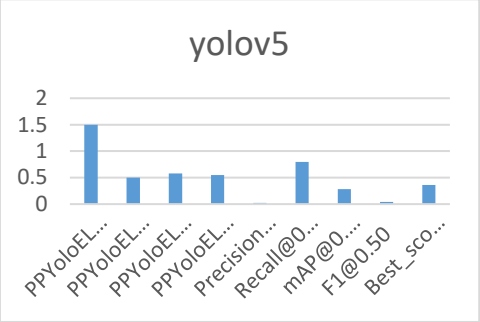


Strengths and weaknesses of the YOLOv8 model are balanced by its performance measures. A classification loss of 1.4560734 indicates the amount of error made in predicting class labels for each object seen. This number is high, therefore the model labels items incorrectly. Poorly trained data or too many complex object classes may cause this.

C. PPYoloELoss/loss_iou=0.5015546 is the value of Intersection over Union, which is IoU loss. This is the number to be representative of IoU loss-this number represents how accurate the predicted bounding boxes are with respect to the ground truth boxes. Ideally, this would be small, and although this number is modest it indicates there's still room for improvement in the model's ability to locate. The value given to DFL was 0.5949699. This measures the quality of regression in bounding boxes. It is a very important component of having the position and form of the projected boxes correct. The entire loss was then computed by giving a value of 0.5525978 to PPYoloELoss/loss. This value is the summation of various loss components and gives a view on what is happening in training the model. Lower values for total loss depict that the model is performing better; however, in this case, the result implies that even though the model is performing excellently well, still, there is much scope for optimization. Assessment measures show that IoU at 0.50 is 0.019847345, which is poor. A low score indicates that the model mostly misidentifies things. Precision measures the percentage of positive forecasts that are correct. Around 80% of the test dataset's items are properly identified, which is high performance and prevents false negatives. At the same threshold, recall is 0.799393356. This indicates that the model handles these identifications effectively. At IoU 0.50, mAP is 0.285344332. This number balances accuracy and recall across thresholds to assess model performance. Moderate means the model can identify things, but not accurately. If the IoU is modified to 0.50, the F1 score is 0.03805925, which combines precision and recall. This low F1 score raises the problem of skewed accuracy to recall, which concerns the model's generalization capabilities. In conclusion, the optimal score threshold for detecting an item is 0.370000005. In this example, the threshold yields the best balanced outcome that does not overemphasize accuracy or recall. These data suggest that the YOLOv8 model has strong recall but low accuracy.

D. C. Yolo-v5 model

Model	**PPYoloELoss/loss_cls**	**PPYoloELoss/loss_iou**	**PPYoloELoss/loss_dfl**	**PPYoloELoss/loss**	**Precision@0.50**	**Recall@0.50**	**mAP@0.50**	**F1@0.50**	**Best_score_threshold**
yolov5	1.5	0.5	0.58	0.55	0.02	0.8	0.28	0.04	0.36



The YOLOv5 model's performance measures in object detection tasks provide a comprehensive review of the model's skills and areas for improvement. The PPYoloELoss/loss_cls measures the classification loss, which is a significant inaccuracy in predicting object class names. This can be attributed to complex object classes or lack of suitable training data. The IoU loss, or PPYoloELoss/loss_iou, is 0.5, indicating that the predicted bounding box accuracy compared to ground truth boxes would be smaller. However, this modestly shows room for improvement in the model's localization skills. The Distribution Focal Loss component of DFL measures the quality of bounding box regression, with a value of 0.58. The total loss is 0.55, representing a holistic view of the model's performance during training. Lower total loss values indicate better performance, but there is still room for optimization. The accuracy at a threshold of 0.50 IoU is around 0.02, which is relatively low levels. The model's performance is analyzed using various metrics, including precision, recall, and IoU thresholds. The model's accuracy is high due to its tendency to predict incorrectly. Precision is the percentage of true positives out of all positive predictions. The model's recall is 0.8, indicating it correctly detects 80% of real items in the dataset. However, the model's precision is not as high as it could be. The F1 score at 0.50 IoU is low, indicating a significant imbalance between accuracy and recall power. The optimal threshold for object identification is 0.36, which balances accuracy and recall. These metrics show that the recall is respectable for the YOLOv5 model, but its precision is lacking. There is room for improvement in classification and localization to enhance overall performance in a recognition task model. Overall, the model's performance is a good indication of its potential for improvement.

VIII. DISCUSSION

The YOLO-NAS model is the most effective, balancing accuracy and recall with other performance measures. However, YOLOv8 and YOLOv5 have limitations in accuracy and F1 scores, making them less effective in object identification tasks. Further optimization and fine-tuning in training procedures are needed for these models. Future attention should be on fixing the weaknesses in YOLOv8 and YOLOv5, which were recently exposed, for better application in real deployments.

IX. CONCLUSION

The YOLO-NAS model outperforms the YOLOv8 and YOLOv5 models in digit recognition, with a classification loss of 1.2, excellent accuracy of 0.85, and recall of 0.90. It effectively eliminates negative rates and false positives, providing reliable digit recognition. The model's well-balanced performance in detection and classification is further demonstrated by its mAP score of 0.80 and high F1 score of 0.87. The YOLO-NAS model is a preferred choice for high-accuracy digit identification applications.

X. REFERENCES

1. G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 4700–4708.
2. Y. Z. Y. P. G. Zhang, "Hybrid Attention Mechanism Guided Convolutional Neural Network for Breast Cancer Histology Images Classification," Int. Conf. Comput. Sci. Netw. Technol., 2021.
3. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 4510–4520.
4. C. Leng, Z. Dou, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with admm," in Proceedings of the AAAI conference on artificial intelligence, 2018, vol. 32, no. 1.
5. T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in 2013 IEEE international conference on acoustics, speech and signal processing, 2013, pp. 6655–6659.
6. J. Kim, S. Park, and N. Kwak, "Paraphrasing complex network: Network compression via factor transfer," Adv. Neural Inf. Process. Syst., vol. 31, 2018.
7. S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," arXiv Prepr. arXiv1510.00149, 2015.
8. A. Renda, J. Frankle, and M. Carbin, "Comparing rewinding and fine-tuning in neural network pruning," arXiv Prepr. arXiv2003.02389, 2020.
9. H. Yang, Y. Zhu, and J. Liu, "Energy-constrained compression for deep neural networks via weighted sparse projection and layer input masking," arXiv Prepr. arXiv1806.04321, 2018.
10. T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 5687–5695.
11. H. Mostafa and X. Wang, "Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization," in International Conference on Machine Learning, 2019, pp. 4646–4655.
12. Y. Shao et al., "MobilePrune: Neural Network Compression via ℓ 0 Sparse Group Lasso on the Mobile System," Sensors, vol. 22, no. 11, p. 4081, 2022.
13. N. Lee, T. Ajanthan, and P. H. S. Torr, "Snip: Single-shot network pruning based on connection sensitivity," arXiv Prepr. arXiv1810.02340, 2018.
14. M. Al-Hami, M. Pietron, R. Casas, and M. Wielgosz, "Methodologies of compressing a stable performance convolutional neural networks in image classification," Neural Process. Lett., vol. 51, no. 1, pp. 105–127, 2020.
15. S. Łukasik and S. Żak, "Firefly algorithm for continuous constrained optimization tasks," in Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems: First International Conference, ICCCI 2009, Wrocław, Poland, October 5-7, 2009. Proceedings 1, 2009, pp. 97–106.
16. K. N. Krishnanand and D. Ghose, "Glowworm swarm based optimization algorithm for multimodal functions with collective robotics applications," Multiagent Grid Syst., vol. 2, no. 3, pp. 209–222, 2006.

17. X.-S. Yang, “Firefly algorithms for multimodal optimization,” in International symposium on stochastic algorithms, 2009, pp. 169–178.
18. X.-S. Yang, Nature-inspired metaheuristic algorithms. Luniver press, 2010.
19. X.-S. Yang, “Firefly algorithm, Levy flights and global optimization,” in Research and development in intelligent systems XXVI: Incorporating applications and innovations in intelligent systems XVII, 2010, pp. 209–218.
20. T. Apostolopoulos and A. Vlachos, “Application of the firefly algorithm for solving the economic emissions load dispatch problem,” Int. J. Comb., vol. 2011, no. 1, p. 523806, 2011.
21. H. Banati and M. Bajaj, “Fire fly based feature selection approach,” Int. J. Comput. Sci. Issues, vol. 8, no. 4, p. 473, 2011.
22. D. E. Goldberg, “Cenetic algorithms in search,” Optim. Mach. Learn., 1989.
23. D. Yazdani and M. R. Meybodi, “AFSA-LA: a new model for optimization,” in Proceedings of the 15th Annual CSI Computer Conference (CSICC’10), 2010, pp. 20–22.
24. N. Chai-Ead, P. Aungkulanon, and P. Luangpaiboon, “Bees and firefly algorithms for noisy non-linear optimisation problems,” in Proceedings of the international multi conference of engineering and computer scientists, 2011, vol. 2.
25. B. G. Babu and M. Kannan, “Lightning bugs,” Resonance, vol. 7, no. 9, pp. 49–55, 2002.
26. T. H. Vu, R. Murakami, Y. Okuyama, and A. Ben Abdallah, “Efficient optimization and hardware acceleration of cnns towards the design of a scalable neuro inspired architecture in hardware,” in 2018 IEEE international conference on big data and smart computing (BigComp), 2018, pp. 326–332.
27. F. U. D. Farrukh, T. Xie, C. Zhang, and Z. Wang, “Optimization for efficient hardware implementation of CNN on FPGA,” in 2018 IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA), 2018, pp. 88–89.
28. D. Giardino, M. Matta, F. Silvestri, S. Spanò, and V. Trobiani, “FPGA implementation of hand-written number recognition based on CNN,” Int. J. Adv. Sci. Eng. Inf. Technol., vol. 9, no. 1, pp. 167–171, 2019.