SAPIENZA
UNIVERSITÀ DI ROMA

Evaluating the Firefighter Optimization Algorithm for Hyperparameter Tuning in CNN-Based Brain Tumor MRI Classification

Facoltà di ingegneria dell'informazione, informatica e statistica

Corso di Laurea in Applied Computer Science and Artificial Intelligence

Candidate

Malak Matar

ID number 2079203

Thesis Advisor

Prof. Luigi Cinque

Academic Year 2024/2025

Contents

1 Background and Related Work **3**

1.1 Artificial Intelligence and Machine Learning Foundations 3

1.2 Deep Learning and Convolutional Neural Networks 6

1.3 Medical Imaging and Brain Tumor Classification 10

1.4 Hyperparameter Optimization in Deep Learning 13

1.5 Metaheuristic Algorithms for Optimization 16

1.6 Particle Swarm Optimization (PSO) 18

1.7 The Firefighter Optimization Algorithm (FFO) 22

2 Methodology **27**

2.1 Introduction 27

2.2 Dataset Description 27

2.3 Preprocessing Pipeline 29

2.4 Model Architecture 31

2.5 Hyperparameters and Search Space 33

2.6 Metaheuristic Optimization Setup 35

2.7 Experimental Protocol 38

2.8 Computational Environment 39

3 Results **41**

3.1 Convergence Behaviour of FFO and PSO 41

3.2 Per-Run Analysis of Optimization Behaviour 43

3.3 Final Best-Hyperparameter Training 47

3.4 Aggregate Comparison of Optimizers 56

4 Conclusion and Future Work **58**

References **60**

Abstract

Deep learning has become a cornerstone of modern medical image analysis, enabling automated detection and diagnosis through data-driven feature extraction. Among deep architectures, **Convolutional Neural Networks (CNNs)** have demonstrated remarkable success in brain tumor classification from Magnetic Resonance Imaging (MRI) scans. However, CNN performance is critically dependent on hyperparameter configuration, and conventional optimization methods—such as manual tuning, grid search, or Bayesian optimization—are often computationally expensive and poorly scalable.

This thesis investigates the use of the **Firefighter Optimization Algorithm (FOA)**, a recently proposed human-inspired metaheuristic, for systematic hyperparameter tuning of a **VGG16-based CNN** in brain tumor MRI classification. The algorithm's effectiveness is rigorously compared with the well-established **Particle Swarm Optimization (PSO)** under strictly identical experimental conditions. Both optimizers operate on the same search space encompassing six hyperparameters—dense layer units, dropout rate, learning rate, batch size, L_2 weight decay, and the number of unfrozen convolutional blocks—and share a fixed function-evaluation budget aligned across runs to ensure methodological fairness.

Experiments were conducted on the publicly available Kaggle Brain Tumor MRI dataset, comprising four diagnostic categories: *glioma*, *meningioma*, *pituitary tumor*, and *no tumor*. Empirical results demonstrate that FOA achieved higher peak performance, reaching a test accuracy of 98.6% under its best configuration, while PSO exhibited smoother convergence and superior probability calibration. These findings confirm that metaheuristic search constitutes an efficient and reproducible strategy for CNN hyperparameter optimization in medical image analysis, with FFO offering competitive performance relative to established swarm-based optimizers.

1 Background and Related Work

1.1 Artificial Intelligence and Machine Learning Foundations

Artificial Intelligence (AI) is a branch of computer science dedicated to creating systems capable of performing tasks that typically require human cognition, such as perception, reasoning, and decision-making. AI systems rely on mathematical models and algorithms to process data, recognize patterns, and generate adaptive responses. In healthcare, AI plays an increasingly central role due to its ability to analyse complex, high-dimensional data and assist clinicians in diagnosis, prognosis, and treatment planning. In particular, AI has become integral to *medical imaging*, which involves the acquisition, processing, and interpretation of visual representations of internal body structures to support clinical evaluation. The use of AI in this domain enhances the reproducibility and precision of diagnostic procedures and reduces inter-observer variability among specialists.

Machine learning (ML) is a core subfield of AI that focuses on enabling algorithms to improve performance automatically by identifying statistical patterns in data rather than following explicitly programmed rules [1, 2]. In ML, models learn from examples and generalize this experience to unseen data. ML is broadly categorized into three paradigms:

- **Supervised learning**, in which models are trained on labeled datasets containing input–output pairs. The algorithm learns a mapping from input variables x (features) to target variables y (labels), allowing it to predict outputs for new inputs. Typical tasks include *classification* and *regression*.

- **Unsupervised learning**, which operates on unlabeled data to uncover hidden structures or relationships. Common goals include *clustering*—grouping similar samples—and *dimensionality reduction*, which compresses data while preserving essential information.

- **Reinforcement learning**, in which an agent interacts with an environment by performing actions and receiving feedback in the form of rewards or penalties. Over time, the agent learns a policy that maximizes cumulative reward, a principle often applied in control and decision-making systems.

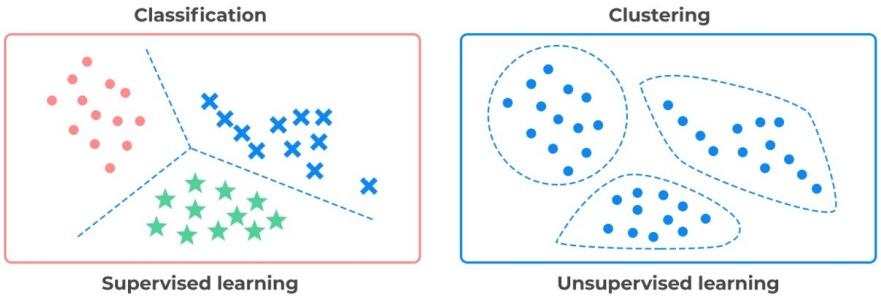


Figure 1: Illustration of supervised vs. unsupervised learning, reproduced from [3].

In medical image analysis, **supervised learning** is the most relevant paradigm because it allows direct mapping from imaging data to diagnostic categories. Early supervised models achieved moderate success but relied heavily on **handcrafted features**—manually engineered descriptors that capture visual cues such as texture, intensity, or edge statistics. These approaches were limited in generalization, as they failed to model the high-level abstractions and spatial dependencies inherent in medical images such as MRI scans.

Classical supervised learning algorithms include **decision trees**, **support vector machines (SVMs)**, and **ensemble methods**. A *decision tree* recursively partitions data according to feature thresholds, forming internal decision nodes and terminal leaves that represent class outcomes. Decision trees are interpretable but prone to overfitting on noisy or high-dimensional data. An *SVM* is a margin-based classifier that identifies the optimal separating hyperplane between classes in a high-dimensional feature space. It employs *kernel functions* to capture nonlinear relationships. *Ensemble methods*, such as Random Forests and Gradient Boosting, combine multiple weak learners to enhance predictive accuracy and robustness. Random Forests reduce variance by averaging decorrelated trees, whereas Gradient Boosting sequentially trains weak models to correct residual errors. While effective on structured data, these traditional algorithms perform poorly on raw image data, where pixel relationships carry essential spatial information.

The growing availability of annotated medical datasets and advances in computational hardware—particularly in graphical processing units (GPUs)—have catalyzed the development of **deep learning**, a subset of ML based on **artificial neural networks (ANNs)**. In deep learning, models learn hierarchical representations directly from raw data, eliminating the need for handcrafted features. An ANN comprises interconnected computational units called *neurons*, each performing a weighted sum of its inputs followed by a nonlinear *activation function*. By stacking multiple layers of neurons, deep networks capture progressively abstract data patterns: early layers detect edges and simple shapes, while deeper layers model high-level concepts such as anatomical structures or tumor regions (Figure 2) [4, 5, 6, 7].

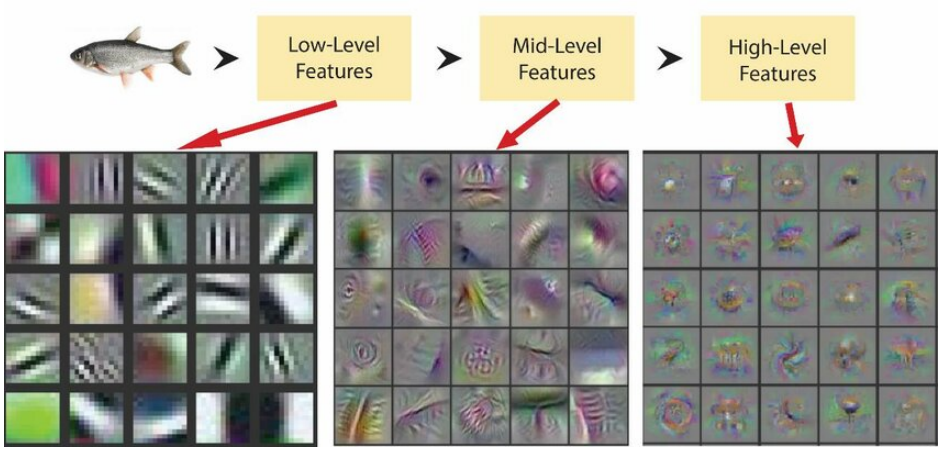


Figure 2: How layers learn data in CNNs, reproduced from [8].

This transition from manually engineered to data-driven representations has transformed computer vision and medical imaging alike. Deep networks automatically infer the most informative features for a given task, enabling them to capture the complex variability of MRI data—differences in tumor morphology, texture, and spatial distribution—that classical ML models cannot represent efficiently. In clinical practice, deep learning has achieved near-expert performance in disease classification, lesion segmentation, and anomaly detection.

Convolutional Neural Networks (CNNs) are a specialized deep-learning architecture tailored for image data. CNNs use small sliding filters, or *convolution kernels*, to detect spatially localized patterns while sharing weights across the input field, reducing the number of parameters and improving generalization. They incorporate operations such as *pooling*, which downsample feature maps to enhance translation invariance, and nonlinear activations such as the Rectified Linear Unit (ReLU) that prevent gradient saturation. These architectural features make CNNs highly effective for visual recognition tasks, including brain-tumor classification and segmentation.

The introduction of CNNs has revolutionized medical image analysis by enabling models to reach diagnostic performance comparable to that of human experts across several domains. Rather than replacing clinicians, CNNs act as decision-support systems that provide reproducible, objective, and scalable assessments, improving the accuracy and consistency of clinical evaluation.

From a theoretical standpoint, supervised learning can be formalized as the minimization of the **expected risk**:

$$f^* = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{(x,y)\sim\mathcal{D}}\big[L(f(x), y)\big],$$

where f is a predictive function within the hypothesis space $\mathcal{F}$, $\mathcal{D}$ is the joint data distribution, and L denotes a loss function that measures prediction error. Because the true distribution $\mathcal{D}$ is unknown, training uses a finite sample $\{(x_i, y_i)\}_{i=1}^n$ and minimizes

the **empirical risk**:

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} L\big(f(x_i), y_i\big) \,,$$

typically optimized through **stochastic gradient descent (SGD)** or adaptive variants such as the **Adam optimizer**, which dynamically adjusts learning rates for each parameter [9]. The values controlling this process, known as **hyperparameters**, determine learning behaviour and model generalization.

The **No-Free-Lunch Theorem** in optimization states that no single algorithm performs best across all problems [10]. Consequently, effective model development depends on selecting appropriate optimization strategies and tuning procedures for the data at hand. These considerations motivate the exploration of more adaptive approaches to optimization, which are elaborated upon in later sections.

1.2 Deep Learning and Convolutional Neural Networks

Artificial Neural Networks (ANNs) are computational models inspired by the interconnected structure of biological neurons. Each **neuron** receives input values x_i, multiplies them by associated weights w_i, adds a trainable offset known as the **bias** b, and applies a nonlinear **activation function** $\sigma(\cdot)$ to generate its output:

$$a = \sigma\left(\sum_i w_i x_i + b\right) .$$

This nonlinearity enables ANNs to approximate complex, non-linear relationships between inputs and outputs.

Common activation functions include:

- **Rectified Linear Unit (ReLU):** $\mathrm{ReLU}(z) = \max(0, z)$. It outputs zero for negative inputs and passes positive values unchanged. ReLU mitigates the *vanishing-gradient problem*, in which gradients become too small to update weights effectively, thereby improving training speed and stability.

- **Sigmoid:** $\sigma(z) = 1/(1 + e^{-z})$. It maps values to the range $[0, 1]$, useful for probabilistic outputs, but may saturate for extreme inputs.

- **Hyperbolic tangent (tanh):** $\tanh(z) = (e^{z} - e^{-z})/(e^{z} + e^{-z})$. It produces zero-centred activations in $[-1, 1]$ and often yields faster convergence than the sigmoid.

Stacking multiple layers of neurons allows **hierarchical feature learning**, where successive layers capture increasingly abstract patterns: early layers detect simple gradients or edges, intermediate layers identify textures or shapes, and deeper layers encode semantic features such as organ boundaries or tumor regions in MRI data [4].

Backpropagation and gradient-based optimization. Training an ANN involves adjusting its parameters (weights and biases) to minimize a loss function that quantifies prediction error. The **backpropagation** algorithm computes the gradient of this loss with respect to every parameter by applying the *chain rule* of calculus from the output layer backward through the network. These gradients are then used by a **gradient-based optimizer** to iteratively refine parameters and reduce the loss.

Two of the most common optimizers are:

- **Stochastic Gradient Descent (SGD):** updates parameters using small random batches of data, introducing stochasticity that helps the model escape shallow local minima.
- **Adam:** an adaptive algorithm that maintains running estimates of both the mean and variance of gradients, enabling per-parameter learning-rate adjustment for faster, more stable convergence.

The **learning rate** controls the magnitude of each parameter update: overly large values can cause divergence, while very small ones slow convergence.

Convolutional layers. **Convolutional Neural Networks (CNNs)** are specialized ANNs designed to exploit the spatial structure of image data. They rely on two principles:

- **Spatial locality**—nearby pixels often contain related information.
- **Parameter sharing**—a single filter, or *kernel*, is applied across multiple regions of the image, drastically reducing the number of parameters.

The fundamental operation in a CNN is the **convolution**, which combines an input f with a small kernel g to produce an output feature map:

$$(f \star g)(i,j) = \sum_m \sum_n f(i+m, j+n)\, g(m,n).$$

Each kernel detects a specific local pattern, such as an edge or texture. Important configuration parameters include the **stride** (the number of pixels the kernel moves each step), **padding** (zero-values added around the borders), and the **number of filters**, which determines the depth of the output. By reusing the same filters across spatial locations, CNNs achieve **translation invariance**, recognizing patterns regardless of position—a crucial property in medical imaging where tumor size and location vary widely [11].

Pooling and downsampling. To reduce computational complexity and improve robustness to small spatial shifts, CNNs include **pooling layers** that summarize local neighbourhoods:

- **Max pooling:** selects the largest activation in each region, preserving the most salient feature.
- **Average pooling:** computes the average activation, yielding smoother representations.

After several convolution–pooling blocks, feature maps are often **flattened** or aggregated using **global pooling** and passed to fully connected layers for classification. The final layer typically employs the **softmax** activation:

$$P(y = k \mid \mathbf{x}) = \frac{\exp(z_k)}{\sum_{j=1}^{K} \exp(z_j)},$$

which converts the model's raw outputs (*logits*) into a normalized probability distribution across K categories such as *glioma*, *meningioma*, *pituitary tumor*, and *no tumor*.

Loss functions. A **loss function** measures the discrepancy between predicted and true labels. For multi-class problems, the standard choice is **categorical cross-entropy**:

$$\mathcal{L} = -\sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log \hat{y}_{ic},$$

where y_{ic} is the true label (one-hot encoded) and $\hat{y}_{ic}$ the predicted probability for class c. Training seeks to minimize $\mathcal{L}$ through repeated forward and backward passes.

Generalization, overfitting, and data augmentation. **Generalization** refers to a model's ability to perform well on unseen data. **Overfitting** occurs when a network memorizes the training set, capturing noise rather than meaningful patterns. **Data augmentation** combats overfitting by synthetically expanding the training set through transformations such as rotation, scaling, flipping, brightness variation, and noise injection. These operations encourage invariance to orientation and illumination. Figure 3 illustrates typical augmentation methods used in brain-tumor MRI classification, which have been shown to improve robustness and accuracy [11].

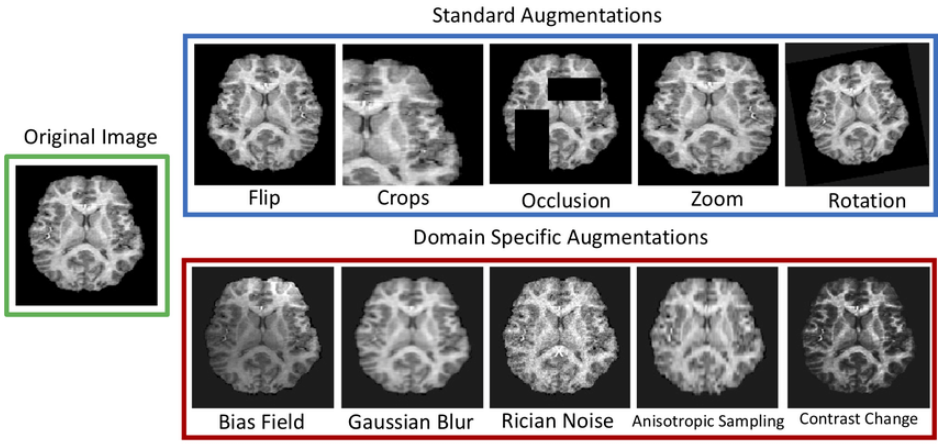


Figure 3: Examples of standard and domain-specific MRI augmentations applied to brain scans, reproduced from [12].

Transfer learning and fine-tuning. **Transfer learning** leverages representations learned by a model pretrained on a large, generic dataset (e.g., ImageNet) to improve performance on a smaller, domain-specific dataset. Lower layers capture generic features such as edges and textures, while higher layers encode task-specific details. Two main strategies are common [13]:

- **Feature extraction:** the pretrained convolutional base is kept fixed while new classifier layers are trained for the target task.
- **Fine-tuning:** deeper layers are unfrozen and retrained with a smaller learning rate, allowing adaptation to domain-specific structures such as tumor morphology.

Transfer learning is particularly advantageous in medical imaging, where annotated datasets are limited. Studies have shown that pretrained CNNs like **VGG16** yield substantially higher accuracy for brain-tumor MRI classification than networks trained from scratch [14].

VGG16 architecture. **VGG16**, introduced by Simonyan and Zisserman [15], follows a simple and uniform architecture built from sequential 3×3 convolutional layers and 2×2 pooling layers (Figure 4). It consists of 13 convolutional and 3 fully connected layers, totalling 16 weight layers.

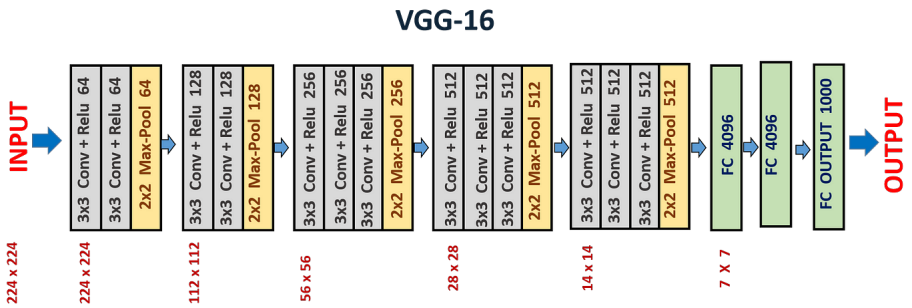


Figure 4: VGG16 processing model, reproduced from [16].

Although more recent architectures such as **ResNet**—which employs *residual connections* to alleviate vanishing gradients—and **DenseNet**—which promotes *feature reuse* through dense inter-layer connections—achieve higher performance on large datasets, VGG16 remains a reliable backbone in biomedical imaging due to its stability, interpretability, and strong transfer-learning performance. Its straightforward design makes it ideal for controlled experiments investigating learning behaviour and generalization.

CNN effectiveness depends strongly on model configuration and training dynamics. Appropriate choices of learning rate, regularization strength, and network depth are essential for stable convergence. Later sections address how optimization strategies and adaptive search methods can be employed to refine these configurations and further enhance model performance in medical image classification.

1.3 Medical Imaging and Brain Tumor Classification

Medical imaging encompasses non-invasive techniques for visualizing the internal structures and physiological functions of the human body. It has become an indispensable tool for modern diagnostics, treatment planning, and disease monitoring. The principal modalities include **Computed Tomography (CT)**, **Positron Emission Tomography (PET)**, **X-ray imaging**, **Magnetic Resonance Imaging (MRI)**, and its vascular extension, **Magnetic Resonance Angiography (MRA)** (Figure 5). Each modality provides complementary information depending on the physical principle used for image formation and tissue contrast.

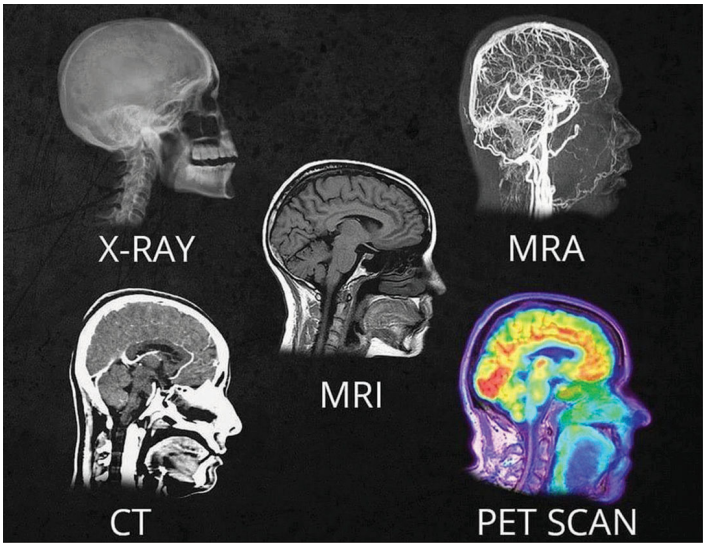


Figure 5: Medical imaging modalities, reproduced from [17].

Computed Tomography (CT) reconstructs volumetric anatomical information from multiple two-dimensional X-ray projections captured at different angles. It is fast and widely accessible but limited in soft-tissue contrast because it measures X-ray attenuation differences. **Positron Emission Tomography (PET)** provides functional information by detecting gamma photons emitted from injected radiotracers that accumulate in metabolically active regions. Although highly sensitive to metabolic activity, PET involves exposure to radioactive materials. **X-ray imaging** produces planar projections based on tissue absorption of ionizing radiation and remains essential for skeletal and thoracic applications but lacks the soft-tissue contrast required for neuroimaging. **Magnetic Resonance Imaging (MRI)** is the preferred modality for brain studies. It uses strong magnetic fields and radiofrequency pulses to align and perturb hydrogen nuclei in water molecules, measuring emitted signals as they relax to equilibrium. MRI yields high-resolution images with excellent soft-tissue contrast while avoiding ionizing radiation, making it ideal for characterizing intracranial abnormalities. **Magnetic Resonance Angiography (MRA)** is an MRI technique optimized to visualize blood vessels and vascular abnormalities such as tumor-induced angiogenesis, assisting in pre-surgical assessment and treatment monitoring [11].

MRI can employ various **pulse sequences**, each emphasizing specific tissue characteristics. **T1-weighted** images provide high anatomical detail and are useful for assessing normal brain structures. **T1 with contrast enhancement (T1c)** involves intravenous administration of gadolinium-based agents that accumulate in areas with a disrupted blood–brain barrier, highlighting active tumor regions. **T2-weighted** scans are sensitive to water content, facilitating detection of edema and cystic lesions, while the **Fluid-Attenuated Inversion Recovery (FLAIR)** sequence suppresses cerebrospinal-fluid signals to improve lesion visibility near the ventricles. By combining these complementary sequences (Figure 6), clinicians can derive a comprehensive view

of tumor morphology, infiltration, and peritumoral edema. This multimodal capability is a major advantage of MRI in neuro-oncology, supporting accurate tumor delineation, subtype differentiation, and surgical planning [18].

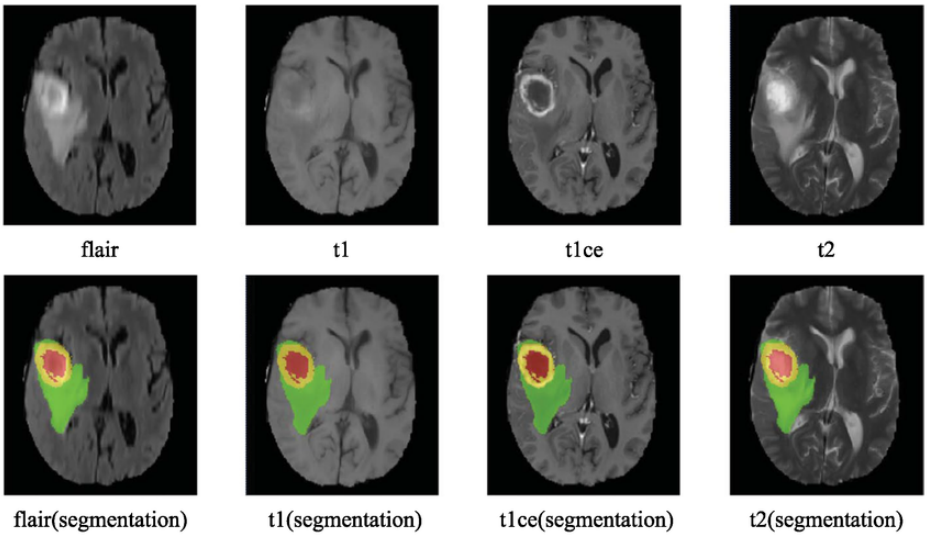


Figure 6: Illustration of MRI modalities, reproduced from [19].

Despite its diagnostic power, MRI data pose challenges for automated analysis. Brain tumors exhibit pronounced **intra- and inter-subject heterogeneity**: a single lesion may contain necrotic cores, proliferative regions, and surrounding edema, each with distinct signal properties. Image quality and contrast vary with scanner type, magnetic-field strength, and acquisition protocol, while motion artifacts and noise introduce additional variability. Manual annotation is labour-intensive and prone to inter-observer variability. These factors highlight the need for computational models capable of generalizing across heterogeneous data and imaging conditions.

Early computational approaches. Initial approaches to automated brain-tumor classification relied on **handcrafted feature extraction**. Descriptors such as **texture statistics**, **wavelet coefficients**, and **histogram-based measures** were engineered to capture local intensity or frequency patterns. These handcrafted features were then used with classical machine-learning algorithms. For example, **Support Vector Machines (SVMs)** locate optimal separating hyperplanes between classes in a high-dimensional space, while **k-Nearest Neighbours (k-NN)** classify new samples based on proximity to known examples. Soltaninejad et al. (2017) combined statistical texture descriptors with SVMs for tumor grading, achieving good results on specific datasets but limited generalization to other scanners or protocols [11]. Because such descriptors depend heavily on acquisition settings, handcrafted pipelines typically lack robustness across institutions.

Deep learning in brain-tumor analysis. The advent of **deep learning** overcame many of these limitations by enabling models to learn discriminative features directly from data. **Convolutional Neural Networks (CNNs)** automatically capture spatial and contextual information relevant to tumor localization and classification, eliminating the need for manual feature design. Pereira et al. (2016) trained a multimodal CNN achieving a Dice coefficient above 0.88 for tumor segmentation, outperforming feature-based methods [20]. Havaei et al. (2017) developed an end-to-end CNN architecture capable of simultaneous segmentation and classification, setting new benchmarks on the BRATS dataset [21]. Subsequent research introduced **transfer learning**, where pretrained networks such as VGG16 are fine-tuned on medical images, achieving over 90 % accuracy in multi-class brain-tumor classification [14]. These results established data-driven representation learning as a cornerstone of modern neuro-imaging analysis.

Need for systematic optimization. Although CNNs achieve strong performance, their effectiveness depends on a complex interplay of architectural design choices and training settings. Parameters such as learning rate, batch size, and regularization strength strongly influence convergence behaviour and final accuracy. Manual tuning or exhaustive grid search is inefficient and non-reproducible, motivating the adoption of **automated optimization** approaches. Recent research explores adaptive search strategies, including **metaheuristic algorithms**, which employ stochastic, population-based mechanisms to balance exploration of the search space with local refinement. These methods have shown promise in discovering well-performing configurations efficiently and improving model robustness in medical-image classification tasks.

1.4 Hyperparameter Optimization in Deep Learning

A fundamental distinction in deep learning lies between **model parameters** and **hyperparameters**. Model parameters—such as the weights and biases of a neural network—are internal variables automatically learned during training through backpropagation and gradient descent. **Hyperparameters**, by contrast, are external configuration settings chosen before training that define how the learning process unfolds. Examples include the learning rate (which determines step size during optimization), batch size (the number of samples processed per update), dropout probability (the fraction of neurons temporarily deactivated during training), regularization strength (the weight decay applied to model parameters), and the number of training epochs. Because these settings control *how* the model learns rather than *what* it learns, they have a decisive influence on both convergence stability and generalization performance.

The sensitivity of deep learning models to these choices is well documented. A learning rate that is too high can cause the loss function to oscillate or diverge, while an excessively low rate slows training and increases computational cost. **Dropout**, a regularization technique that randomly disables a subset of neurons during training, mitigates overfitting by preventing co-adaptation among features; however, excessive dropout can suppress important signals. The **batch size** affects both efficiency and convergence: small batches introduce stochastic noise that may improve generalization, whereas large batches smooth the optimization trajectory but risk convergence to sharp local minima. Likewise, the **number of epochs** must balance underfitting (too few training passes) against overfitting (too many). These examples highlight why careful **hyperparameter optimization (HPO)** is critical in practice, particularly in high-stakes fields such as medical imaging where performance consistency and reproducibility are essential.

Formally, HPO can be expressed as a **black-box optimization problem**. Let $\theta \in \Theta$ represent a vector of hyperparameters within a predefined search space Θ. Given a model $f(x;\theta)$, the goal is to maximize a validation metric $\mathcal{M}$ (for example, accuracy or F1-score):

$$\theta^* = \arg\max_{\theta\in\Theta} \mathcal{M}\big(f(X;\theta), y\big).$$

Here, $\mathcal{M}$ is typically non-convex and stochastic, as each evaluation requires retraining the model and is affected by random initialization, data shuffling, and noise. Because training a deep neural network is computationally intensive, the objective function is also expensive to evaluate, making HPO one of the major computational challenges in modern deep learning [22, 23].

Traditional optimization strategies. Classical HPO methods address this problem through different strategies. **Manual tuning** relies on practitioner intuition and iterative trial-and-error, offering flexibility but lacking scalability and reproducibility. **Grid search** systematically evaluates all combinations in a discrete parameter grid, guaranteeing exhaustive coverage but suffering from the *curse of dimensionality*—its computational cost grows exponentially with the number of hyperparameters. **Random search** improves efficiency by sampling configurations uniformly at random, a method shown to outperform grid search in high-dimensional spaces where only a subset of parameters significantly impacts model performance [22]. **Bayesian optimization** refines this idea by building a probabilistic surrogate model—often a **Gaussian Process (GP)**—to approximate the relationship between hyperparameters and performance. It then uses an **acquisition function** (such as Expected Improvement) to prioritize regions of the search space likely to yield improvement [23]. While Bayesian optimization is sample-efficient, it scales poorly to the noisy, high-dimensional search spaces typical of deep networks, and its surrogate modeling incurs significant computational overhead. Figure 7 conceptually compares these approaches.

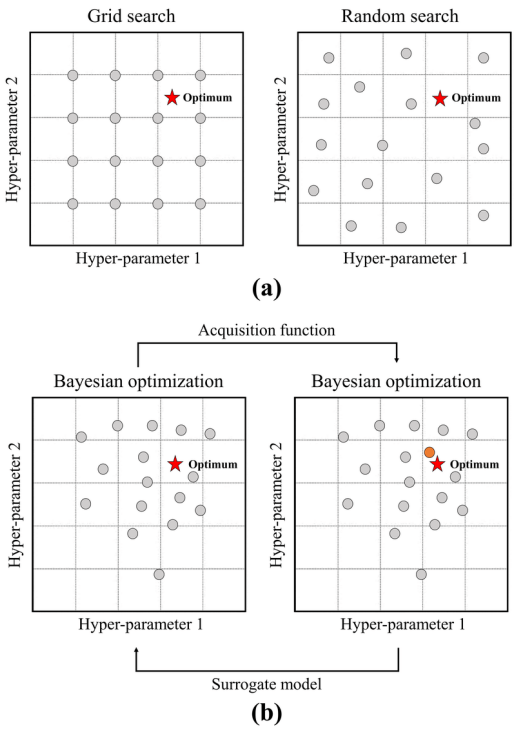


Figure 7: Visual comparison of grid search, random search, and Bayesian optimization for hyperparameter tuning, reproduced from [24].

Limitations of conventional approaches. Despite their conceptual appeal, conventional HPO strategies exhibit several drawbacks when applied to deep learning. Manual and grid search are prohibitively expensive for large networks; random search wastes many evaluations in low-potential regions of the search space; and Bayesian optimization, though elegant, degrades in efficiency as dimensionality and noise increase. Additionally, these methods implicitly assume a static objective function, while in reality the loss landscape of deep learning evolves dynamically as the network trains. This mismatch between assumption and practice limits their effectiveness. In medical imaging studies, inconsistently tuned hyperparameters have been identified as a major source of variability and reproducibility issues across reported CNN results [25, 26].

Metaheuristic approaches. These limitations have led to growing interest in **metaheuristic optimization algorithms**—stochastic, population-based methods that search complex spaces by combining random exploration with adaptive refinement. Unlike deterministic grid or probabilistic Bayesian methods, metaheuristics iteratively evolve a population of candidate solutions, allowing the search to escape local minima and adapt to non-stationary or multimodal landscapes. Their design is often inspired by natural or social processes, such as animal behaviour, swarm intelligence, or human decision-making. Well-known examples include **Particle Swarm Optimization (PSO)**, based on the social dynamics of flocking and schooling; the **Grey Wolf**

Optimizer (GWO), which models the cooperative hunting hierarchy of wolves; and the more recent **Firefighter Optimization Algorithm (FFO)**, a human-inspired method that models adaptive resource allocation during firefighting scenarios.

Because metaheuristics require only function evaluations rather than gradient information, they are well-suited for optimizing deep networks, whose loss functions are non-differentiable with respect to hyperparameters. Their inherent stochasticity promotes diversity in the search process, while population interactions enable convergence toward high-quality solutions under limited computational budgets. These properties make metaheuristic optimization a compelling framework for hyperparameter tuning in computationally demanding domains such as brain-tumor MRI classification.

1.5 Metaheuristic Algorithms for Optimization

Metaheuristic algorithms are high-level, problem-independent optimization strategies designed for complex search spaces where traditional deterministic methods become infeasible or inefficient. They are particularly effective for **non-convex**, **discontinuous**, and **high-dimensional** problems in which the objective function may be computationally expensive to evaluate or lack analytical gradients. Unlike gradient-based optimizers, metaheuristics rely on stochastic sampling rather than derivative information, making them robust against noisy or irregular search landscapes.

A defining feature of metaheuristics is the balance between two complementary processes: **exploration**, which guides the search toward new and unexplored regions of the solution space, and **exploitation**, which refines existing high-quality solutions. Maintaining this balance allows the algorithm to avoid premature convergence on suboptimal regions while progressively improving solution quality. Most metaheuristics employ a **population-based** design, in which multiple candidate solutions evolve cooperatively over successive iterations. This design promotes diversity, improves robustness to local minima, and facilitates parallel exploration—properties that are particularly beneficial for deep learning hyperparameter optimization.

Genetic Algorithms (GA). Among the earliest metaheuristics, the **Genetic Algorithm (GA)** is inspired by Darwinian principles of evolution and natural selection. Candidate solutions, or *chromosomes*, are encoded as vectors that represent potential parameter configurations. The algorithm evolves this population over generations using three key operators:

- **Selection:** individuals with higher fitness have a greater probability of being chosen as parents.
- **Crossover:** two parents combine portions of their chromosomes to generate offspring, encouraging the mixing of traits.

- **Mutation:** small random modifications are introduced to maintain population diversity and prevent stagnation.

GA is flexible and applicable to discrete, continuous, and combinatorial problems. However, its performance depends on the correct tuning of parameters such as mutation and crossover rates, and it may converge slowly in high-dimensional continuous spaces typical of CNN hyperparameter search.

Simulated Annealing (SA). **Simulated Annealing (SA)** is a probabilistic optimization algorithm inspired by the metallurgical annealing process, where materials are heated and gradually cooled to reach a low-energy crystalline structure. Unlike GA, SA operates on a single candidate solution at a time. At each iteration, a random neighbor of the current solution is generated, and its acceptance is determined probabilistically:

$$P = \exp\left(-\frac{\Delta E}{T}\right),$$

where ΔE represents the change in the objective function and T denotes the **temperature**, a control parameter that decreases over time. Initially, high values of T permit acceptance of worse solutions to encourage exploration; as T decreases, the algorithm becomes more conservative, emphasizing exploitation. SA's main strength lies in its ability to escape local minima, though it typically converges slowly and scales poorly with problem dimensionality.

Grey Wolf Optimizer (GWO). The **Grey Wolf Optimizer (GWO)**, introduced by Mirjalili in 2014 [27], models the social hierarchy and cooperative hunting behavior of grey wolves. The population is structured into four groups: *alpha*, *beta*, and *delta* wolves, which lead the search, and *omega* wolves, which follow. The positions of candidate solutions are updated based on encircling and attacking strategies defined by:

$$D = |C \cdot X_p(t) - X(t)|, \qquad X(t+1) = X_p(t) - A \cdot D,$$

where X_p denotes the position of the prey (the best solution found), and A and C are coefficient vectors that control convergence pressure and randomness. Early in the search, larger stochastic fluctuations promote exploration, while later iterations encourage exploitation near the best solutions. GWO is computationally efficient and has demonstrated strong performance across engineering and medical-imaging optimization problems, such as feature selection and tumor classification [18].

Comparison and relevance. Despite their differing inspirations—biological evolution, physical annealing, and social hunting—these algorithms share common design elements: stochasticity, adaptability, and iterative improvement. Each employs randomized search operators to explore complex landscapes and refine

candidate solutions toward optimality. Table 1 summarizes their conceptual principles, advantages, and limitations. Their success across various domains highlights their effectiveness as baselines and comparators for more recent approaches such as the **Firefighter Optimization Algorithm (FFO)**, which extends these principles through human-inspired resource allocation dynamics.

Table 1: Summary comparison of classical metaheuristic algorithms.

Algorithm	**Inspiration and Strategy**	**Strengths**	**Limitations**
Genetic Algorithm (GA)	Evolutionary biology: selection, crossover, and mutation	Versatile across domains, supports both discrete and continuous variables, maintains diversity	Sensitive to parameter tuning, slow convergence in large continuous spaces
Simulated Annealing (SA)	Thermodynamics: probabilistic acceptance governed by temperature schedule	Simple implementation, effective at escaping local minima	Operates on a single candidate, slow convergence in high-dimensional problems
Grey Wolf Optimizer (GWO)	Social hierarchy and hunting coordination of grey wolves	Simple, balanced exploration and exploitation, effective for multimodal landscapes	Less robust in very high-dimensional settings; may require fine-tuning of control parameters

1.6 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is one of the most influential swarm intelligence algorithms, introduced by Kennedy and Eberhart in 1995 [28]. It draws inspiration from the collective behavior of bird flocks and fish schools, where individuals adjust their trajectories based on both personal experience and information shared within the group. Owing to its simplicity, computational efficiency, and adaptability, PSO has been widely applied across engineering, computer vision, and machine learning—particularly for **hyperparameter tuning** and feature selection.

Core concept. In PSO, each candidate solution is represented as a *particle* that moves through the search space. Every particle has two key attributes: a **position**

vector, encoding the candidate solution, and a **velocity vector**, determining how the position changes over time. Each particle remembers the best position it has visited (*personal best*, *pBest*) and communicates with others through the *global best* (*gBest*)—the best position found by any particle in the swarm. This dual influence of individual learning and collective knowledge enables efficient global search through cooperative interaction.

Mathematical formulation. The evolution of each particle follows two equations: one governing velocity and the other position. For particle i at iteration t,

$$v_i(t+1) = \omega v_i(t) + c_1 r_1 (pBest_i - x_i(t)) + c_2 r_2 (gBest - x_i(t)),$$

$$x_i(t+1) = x_i(t) + v_i(t+1),$$

where $x_i(t)$ and $v_i(t)$ are the current position and velocity, ω is the **inertia weight**, c_1 and c_2 are the **cognitive** and **social acceleration coefficients**, and $r_1, r_2 \sim U(0,1)$ are random values introducing stochasticity.

Each term serves a specific function:

- **Inertia term** ($\omega v_i(t)$): promotes momentum and exploration by encouraging continuation of the current trajectory.
- **Cognitive term** ($c_1 r_1 (pBest_i - x_i(t))$): models self-learning, drawing the particle back toward its own best experience.
- **Social term** ($c_2 r_2 (gBest - x_i(t))$): represents information sharing, pulling the particle toward the swarm's best-known solution.

These interacting forces govern the trade-off between exploration and exploitation. If exploration dominates, particles spread widely and may fail to converge; if exploitation dominates, the swarm can prematurely cluster around suboptimal solutions (Figure 8). Appropriate tuning of ω, c_1, and c_2 is thus critical to achieving balanced dynamics.

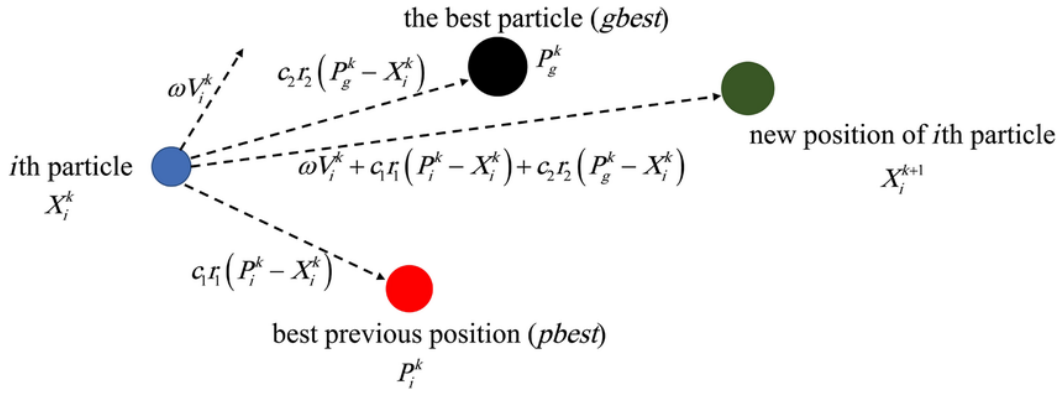


Figure 8: PSO velocity and position updates. Each particle is influenced by inertia, its own best-known position (*pBest*), and the swarm's global best (*gBest*), reproduced from [29].

Over successive iterations, particles exchange information and collectively converge toward promising regions of the search space, as illustrated in Figure 9.

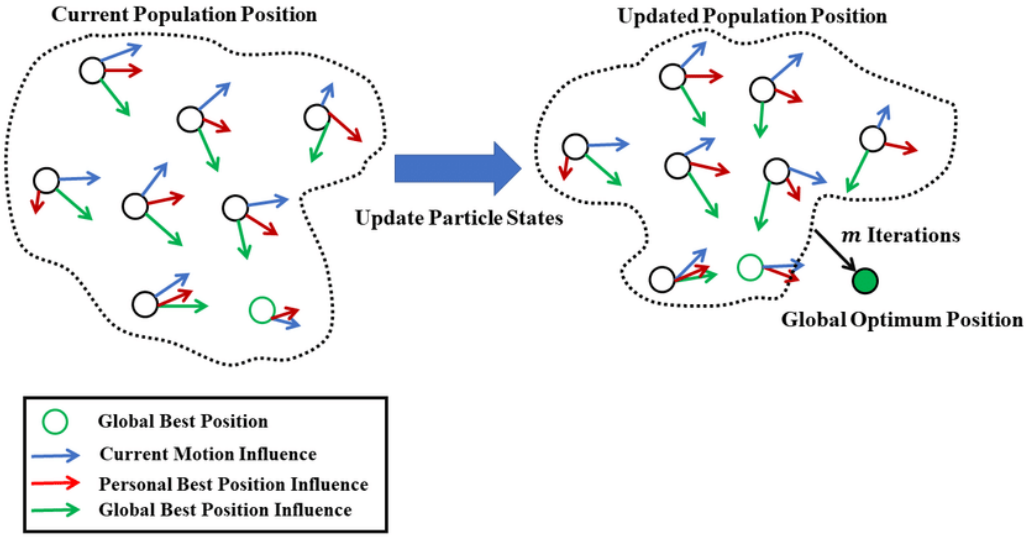


Figure 9: Swarm dynamics in PSO. Particles are influenced by inertia (blue), individual knowledge (red), and collective experience (green), reproduced from [30].

Variants and improvements. The original PSO formulation, while effective, can suffer from premature convergence when swarm diversity decreases too quickly. Two common refinements improve its stability:

- A **linearly decreasing inertia weight** ω: starts large to promote exploration early in the search and decreases gradually to enhance exploitation in later iterations.
- The **constriction factor** approach: scales the velocity update as

$$v_i(t+1) = \chi\big[v_i(t) + c_1 r_1(pBest_i - x_i(t)) + c_2 r_2(gBest - x_i(t))\big],$$

  where χ is derived from swarm-stability theory to ensure bounded particle trajectories and controlled convergence.

These refinements make PSO more resilient in high-dimensional and multimodal optimization problems, including deep learning hyperparameter tuning.

Algorithmic flow. The overall process is summarized in Algorithm 1.

Algorithm 1: Particle Swarm Optimization (PSO) [28]

Input: Swarm size N, max iterations T, inertia weight ω, cognitive coefficient c_1, social coefficient c_2

Output: Best solution $gBest$

Initialize positions x_i^0 and velocities v_i^0 randomly;
Evaluate fitness of all particles;
Set $pBest_i = x_i^0$ for all i;
Set $gBest$ as the best among $pBest_i$;
for $t = 1$ **to** T **do**
    for *each particle* i **do**
        Update velocity $v_i(t+1)$;
        Update position $x_i(t+1)$;
        Evaluate fitness of $x_i(t+1)$;
        if *fitness*$(x_i(t+1))$ *better than fitness*$(pBest_i)$ **then**
            Update $pBest_i$;
        end
    end
    Update $gBest$ if any $pBest_i$ improves;
end
return $gBest$;

Computational complexity. Let N denote the swarm size, T the number of iterations, and d the dimensionality of the search space. Each iteration evaluates N candidate solutions, leading to $M = N \times T$ total evaluations. If C_{eval} is the computational cost of evaluating one candidate (e.g., training a CNN for several epochs and computing validation accuracy), then

$$\text{Time}_{\text{PSO}} = O(M\, C_{\text{eval}}),$$

where vector updates contribute only $O(Nd)$ per iteration, negligible compared to training time. The memory requirement scales as $O(Nd)$ to store particle positions, velocities, and personal bests. In this work, all optimizers are compared under an equal evaluation budget $M^\star$, ensuring fair computational conditions.

Applications in machine learning and medical imaging. PSO has become a standard tool for machine learning optimization due to its simplicity and effectiveness in navigating complex, noisy landscapes. In CNN hyperparameter tuning, each particle typically encodes parameters such as learning rate, dropout probability, batch size, or the number of dense units, with model accuracy on a validation set serving as the objective function. In medical imaging, PSO has achieved strong results in multiple applications. El Amoury et al. (2025) used PSO to tune CNN hyperparameters

for brain-tumor MRI classification, reaching accuracies above 99% on the Kaggle dataset [26]. Saifullah et al. (2025) developed PSO-UNet, where PSO optimized U-Net hyperparameters for brain-tumor segmentation, achieving Dice scores above 0.95 [31]. Beyond neuro-oncology, PSO has also been applied to breast cancer histopathology, lung nodule detection, and radiomic feature selection [25].

Motivation for inclusion. PSO provides a strong and well-established benchmark for evaluating the **Firefighter Optimization Algorithm (FFO)**. Its maturity, interpretability, and extensive validation across deep learning tasks make it an ideal baseline. By comparing FFO directly against PSO, the performance and robustness of the proposed optimization framework can be assessed under consistent experimental conditions.

1.7 The Firefighter Optimization Algorithm (FFO)

The **Firefighter Optimization Algorithm (FFO)** is a recent addition to the family of metaheuristic optimizers, proposed by Naser and Naser in 2024 [32]. Unlike bio-inspired or physics-based methods, FFO is a **human-inspired** algorithm that models the strategies used by firefighters to contain and extinguish wildfires. In this analogy, the *fire* represents uncontrolled exploration of new regions in the search space, while the *firefighters* symbolize targeted exploitation—resources strategically allocated to promising areas. By simulating the dynamic tension between spread and suppression, FFO naturally achieves the core balance that defines all metaheuristics: wide-ranging exploration early in the search and focused refinement as superior solutions emerge.

Core concept. In FFO, each candidate solution is treated as a *fire source*, characterized by its location (a point in the search space) and its *intensity*, which corresponds to fitness. Fires spread outward to generate new candidate solutions, while firefighters are deployed to high-intensity areas to contain and refine them. This process models both global exploration and local intensification. To prevent premature convergence, FFO integrates **simulated annealing** acceptance and **genetic-inspired** crossover and mutation operators, which preserve diversity and allow the search to escape local minima.

Representation of agents. Each fire source represents a d-dimensional solution vector:

$$X_i^t = (x_{i1}^t, x_{i2}^t, \ldots, x_{id}^t),$$

where $i = 1, 2, \ldots, N$ indexes the population, d is the search-space dimensionality (number of hyperparameters), and t is the iteration index. The fitness $f(X_i^t)$ measures the quality of each solution—for instance, the validation accuracy achieved by the CNN

for a given hyperparameter configuration. Conceptually, FFO manages N ignition points, each evolving under both stochastic expansion (fire spread) and adaptive resource allocation (firefighting).

Fire spreading dynamics (exploration). The exploratory component of FFO models the outward spread of a fire:

$$X_i^{t+1} = X_i^t + \alpha R(X_{\text{best}}^t - X_i^t) + \beta\epsilon,$$

where X_{best}^t is the best solution at iteration t, α is a coefficient controlling attraction toward the global best, R is a random diagonal matrix defining random spread directions, and $\beta\epsilon$ introduces stochastic perturbation.

- The term $\alpha R(X_{\text{best}}^t - X_i^t)$ directs each fire toward promising areas (controlled exploitation).
- The random perturbation $\beta\epsilon$ promotes diversification and global search.

This mechanism maintains a dynamic balance between exploitation of strong solutions and continued exploration of new regions.

Resource allocation (exploitation). FFO models exploitation through the adaptive distribution of resources (firefighters). Fires with higher fitness attract more resources, which are then used to generate refined local candidates:

$$n_i = \left\lfloor \frac{F_i}{\sum_{j=1}^{N} F_j} \cdot R_{\text{total}} \right\rfloor,$$

where n_i denotes the number of resources assigned to fire i, F_i its fitness, and R_{total} the total available resources. This proportional allocation mimics real firefighting priorities, ensuring that stronger fires (better solutions) receive greater attention while weaker ones still contribute to maintaining population diversity.

Simulated annealing acceptance. To encourage diversification and avoid stagnation, FFO occasionally accepts inferior solutions according to a simulated-annealing rule:

$$P = \exp\left(-\frac{\Delta f}{T(t)}\right), \quad T(t) = T_0\gamma^t,$$

where Δf is the decrease in fitness, T_0 the initial temperature, and γ the cooling coefficient ($0 < \gamma < 1$). Early in the search, high $T(t)$ values increase the probability of accepting worse candidates (enhanced exploration), while later stages focus on exploitation as $T(t)$ decreases.

Genetic-inspired operators. To sustain diversity and adaptability, FFO integrates simple genetic mechanisms:

- **Crossover:** combines segments from two parent fires to create new candidates, facilitating knowledge transfer between solutions.
- **Mutation:** introduces random perturbations to a subset of parameters, promoting sudden jumps into previously unexplored regions.

These operators help FFO maintain a heterogeneous population, preventing convergence to local optima and enhancing its robustness on complex landscapes.

Algorithmic flow. The main steps of the FFO are summarized in Algorithm 2.

Algorithm 2: Firefighter Optimization Algorithm (FFO) [32]

Input: Population size N, max iterations T, total resources R_{total}, initial temperature T_0, cooling rate γ

Output: Best solution X_{best}

```
Initialize N fire sources X_i^0 randomly;
Evaluate fitness f(X_i^0) for all fires;
Set X_best as the best initial solution;
for t = 1 to T do
    Allocate resources n_i proportional to f(X_i^t);
    for each fire source i do
        Generate new candidates via fire spread;
        Apply crossover and mutation operators;
        Evaluate fitness of generated candidates;
        if candidate improves X_i^t then
            Update X_i^t;
    Update X_best if any improvement occurs;
    Apply simulated annealing acceptance;
    Reduce temperature: T(t+1) = γT(t);
return X_best;
```

Computational complexity. Let N denote population size, T the number of iterations, and C_{eval} the cost of one fitness evaluation (e.g., CNN training for a small epoch subset). FFO performs $O(N)$ base evaluations per iteration plus additional evaluations from crossover and mutation. With expected factors p_{x} (crossover probability), p_{m} (mutation probability), and L (local tries per mutation), the expected cost per iteration is

$$O\big(N(1 + \alpha p_{\text{x}} + p_{\text{m}} L)\, C_{\text{eval}}\big).$$

As in PSO, the vector operations themselves are negligible ($O(Nd)$). Under a fixed evaluation budget $M^\star$, the runtime scales as

$$\text{Time}_{\text{FFO}} = O(M^\star C_{\text{eval}}),$$

allowing fair comparison with PSO under equivalent resource constraints.

Reported performance. Preliminary studies [32] indicate that FFO achieves competitive or superior performance compared with established metaheuristics on benchmark problems, maintaining exploration for longer periods and reducing stagnation near local optima. The combination of fire spreading (global search), resource allocation (guided exploitation), and annealing-based acceptance (adaptive diversification) provides a hybrid mechanism that supports steady convergence, as illustrated in Figure 10.

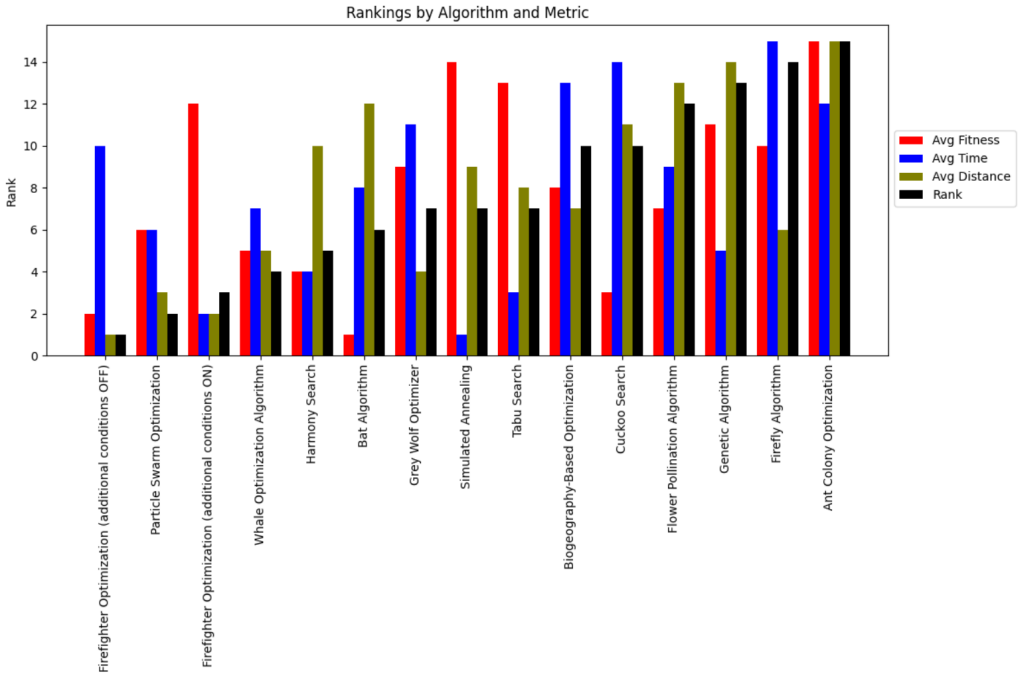


Figure 10: Reported comparative ranking of metaheuristics on two-dimensional benchmarks. FFO demonstrates strong convergence behaviour and exploration balance, reproduced from [32].

Motivation for inclusion. Although promising, FFO remains a novel algorithm with limited empirical evaluation beyond synthetic and engineering benchmarks. Its human-inspired structure offers an intriguing hybridization of exploration, exploitation, and adaptive resource allocation. Applying FFO to CNN hyperparameter optimization in medical imaging therefore represents a novel and valuable research direction. The present work investigates whether these mechanisms can improve the robustness and

reliability of hyperparameter selection compared with established optimizers such as PSO.

2 Methodology

2.1 Introduction

This chapter presents the methodology employed throughout the study, detailing the dataset, preprocessing pipeline, model architecture, hyperparameter search space, optimization algorithms, and evaluation protocol. The overarching goal is to establish a transparent, reproducible, and extensible experimental framework that allows independent verification and future extension of the results.

To this end, all stages of the workflow were implemented as modular Python scripts with well-documented **Command-Line Interfaces (CLI)**. This design facilitates flexible experimentation through parameterized execution, version control, and automation. Every stochastic process—such as data splitting, parameter initialization, and optimizer sampling—was initialized with explicit **random seeds** and logged to dedicated JSON files. This ensures full reproducibility of the results under identical configurations and allows precise tracing of each experiment's provenance.

2.2 Dataset Description

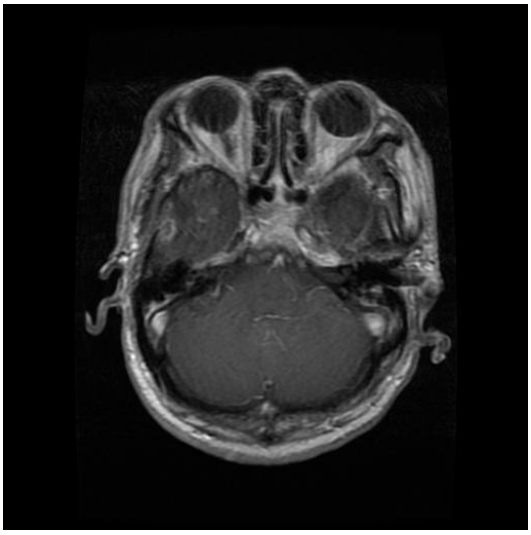
The experiments were performed on the publicly available **Kaggle Brain Tumor MRI** dataset [33], which contains a collection of T1-weighted magnetic resonance imaging (MRI) slices labeled according to tumor type. The dataset consists of a total of 3,264 images distributed across four diagnostic categories:

- **Glioma tumor** — 926 slices
- **Meningioma tumor** — 937 slices
- **Pituitary tumor** — 901 slices
- **No tumor (healthy)** — 500 slices

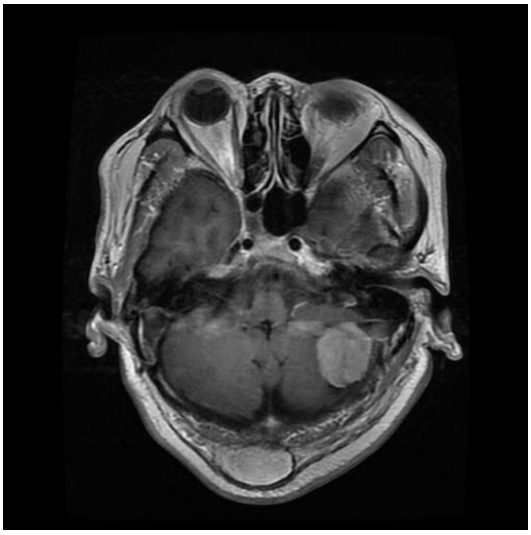
An alternative dataset—the **Figshare Brain Tumor Dataset**—was initially considered. It contains 3,064 T1-weighted contrast-enhanced images from 233 patients across three tumor types: meningioma (708 slices), glioma (1,426 slices), and pituitary tumor (930 slices). This dataset was ultimately discarded because it lacks the *no-tumor* category and provides slightly fewer samples overall. The inclusion of healthy MRI slices in the Kaggle dataset was deemed beneficial for evaluating the classifier's ability to distinguish pathological from non-pathological cases.[1]

Figure 11 presents representative MRI slices from each category, illustrating both inter-class and intra-class variability in tumor morphology and intensity profiles.

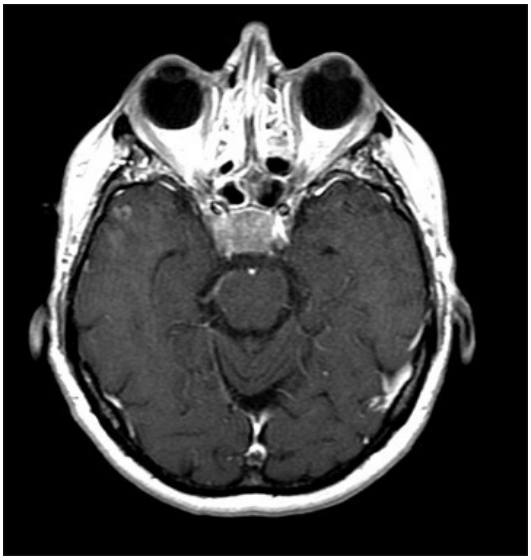
[1] Exact counts after stratified splitting are summarized in Table 2.



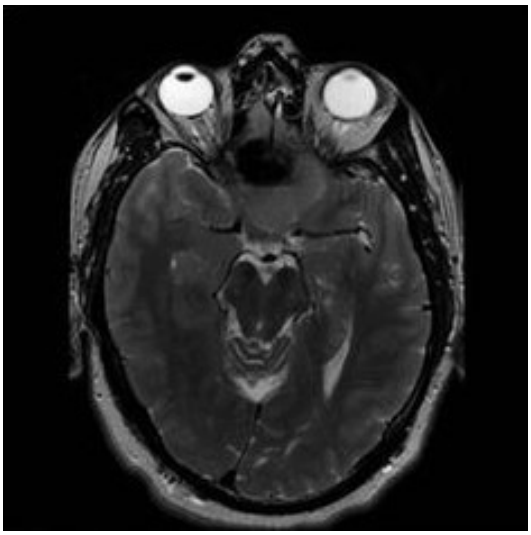
(a) Glioma tumor sample



(b) Meningioma tumor sample



(c) Pituitary tumor sample



(d) Healthy brain sample

Figure 11: Representative MRI slices from each diagnostic class, reproduced from [33].

Data splitting strategies. Two controlled dataset partitioning strategies were implemented within the preprocessing script to accommodate different evaluation needs:

1. **Kaggle-provided split:** uses the official folder structure, where images are already divided into `Training/` and `Testing/` subsets.
2. **Reproducible stratified 70/15/15 split:** merges all images and re-samples them into `train`, `val`, and `test` subsets using a fixed random seed. Stratification ensures that the class proportions remain consistent across all subsets.

Each splitting operation generates and stores its associated random seed in the files `split_seed.json` and `seeds_history.json`, enabling the exact replication of data partitions in subsequent runs. Stratification and seed logging are handled automatically during preprocessing, forming the foundation of experimental reproducibility throughout the project.

Although the dataset is relatively balanced among tumor classes, the *no-tumor* category is slightly underrepresented. This minor imbalance was deliberately retained to preserve the dataset's natural distribution and to evaluate how the optimization algorithms handle intrinsic class variability without artificial resampling or augmentation. This decision reflects the practical conditions under which clinical MRI datasets are typically collected.

Data Integrity and Split Audits

To prevent accidental **data leakage**—the unintentional reuse of test images during training or validation—the pipeline performs automated consistency checks after every split. Specifically, it verifies that the training, validation, and test sets are *mutually disjoint* and that the total number of images per class matches the expected counts from the dataset description.

During both the merging and splitting stages, the script computes per-class file counts and writes an integrity report containing the split statistics, dataset configuration, and random seed used to `experiment_config.json`. When the `--use-kaggle-test` flag is enabled, the test subset is copied directly from Kaggle's official `Testing/` directory, and its provenance is explicitly recorded in `test_source.txt`. These logs collectively form an auditable record of dataset preparation for each experimental run.

2.3 Preprocessing Pipeline

To ensure standardized and reproducible input data across all experiments, a dedicated preprocessing pipeline was implemented in `preprocess.py`. This pipeline converts the raw MRI dataset into consistent, machine-readable arrays suitable for deep learning. The core preprocessing steps are:

- **Resizing and color conversion:** Each MRI slice was resized to 224×224 pixels and converted to a three-channel RGB format to match the input specification of the VGG16 convolutional base. Although MRI data are originally grayscale, the conversion ensures compatibility with ImageNet-pretrained weights used during transfer learning.

- **Normalization:** Pixel intensity values were scaled to the range $[0, 1]$ according to
$$x' = \frac{x}{255},$$
where x denotes the raw pixel value (0–255). Normalization ensures numerical stability, accelerates gradient-based optimization, and prevents exploding or vanishing gradients during model training.

- **Storage:** The processed datasets were serialized into binary NumPy arrays (`.npy` format) for efficient I/O and consistent type handling across experiments. The resulting files (`train_X.npy`, `train_y.npy`, etc.) allow direct memory mapping into TensorFlow data pipelines, minimizing preprocessing overhead during training.

No data augmentation was performed at this stage in order to maintain strict fairness between optimization algorithms. Both the Firefighter Optimization Algorithm (FFO) and Particle Swarm Optimization (PSO) were thus evaluated under identical training conditions, ensuring that observed performance differences result exclusively from the optimization process rather than from altered data distributions.

Reproducibility flags. The preprocessing script supports several command-line flags that control its behavior:

- `--force-resplit`: forcibly regenerate the train/validation/test partitions, overwriting previous splits.

- `--regenerate-seed`: produce a new random seed and append it to `seeds_history.json`.

- `--use-kaggle-test`: use the Kaggle-provided test subset instead of resampling it.

All metadata—including file counts, split ratios, and random seeds—are stored in `experiment_config.json` and `test_source.txt`, enabling precise reconstruction of any experiment's preprocessing stage.

Scripted Interface and Outputs

Example CLI command.

```
python preprocess.py --force-resplit --regenerate-seed --use-kaggle-test
```

Generated outputs.

- **Arrays:** `train_X.npy`, `val_X.npy`, `test_X.npy`
- **Labels:** `train_y.npy`, `val_y.npy`, `test_y.npy`
- **Logs and metadata:** `split_seed.json`, `seeds_history.json`, `experiment_config.json`, `test_source.txt`

Directory structure after preprocessing.

```
raw_data/
  Training/ ...
  Testing/ ...
  Combined/ ...
data/
  train/<class>/*
  val/<class>/*
  test/<class>/*

train_X.npy, train_y.npy,
val_X.npy,   val_y.npy,
test_X.npy,  test_y.npy

experiment_config.json,
split_seed.json,
seeds_history.json,
test_source.txt
```

2.4 Model Architecture

The backbone of the classification model was the **VGG16** convolutional neural network [15], a 16-layer deep architecture originally trained on ImageNet for large-scale object recognition. The network was loaded from `tf.keras.applications` with `include_top=False` to exclude its original fully connected classifier and retain only the convolutional base. The convolutional layers were initialized with **ImageNet-pretrained weights** and initially frozen to implement transfer learning.

Transfer learning allows the network to leverage general-purpose feature extractors—such as edge and texture detectors—learned from natural images, which can be repurposed for medical imaging tasks where training data are more limited. By freezing these early convolutional blocks, the model retains the low-level spatial representations while learning new high-level discriminative patterns in the added classification head.

Custom classification head. A task-specific classifier was appended to the pretrained base, composed of the following layers:

1. **Flatten layer** — converts the final convolutional feature maps into a one-dimensional vector.

2. **Fully connected (dense) layer** — the number of neurons is treated as a tunable hyperparameter within the range 128−512.

3. **Dropout layer** — randomly deactivates a fraction of neurons during training to prevent overfitting; its rate is also a tunable hyperparameter within the range 0.2−0.6.

4. **Dense output layer with softmax activation** — contains four output neurons corresponding to the diagnostic categories: glioma, meningioma, pituitary tumor, and no tumor.

This hybrid structure combines the representational strength of VGG16's convolutional base with the flexibility of a lightweight classifier head adapted to the target domain. Figure 12 illustrates the overall architecture.

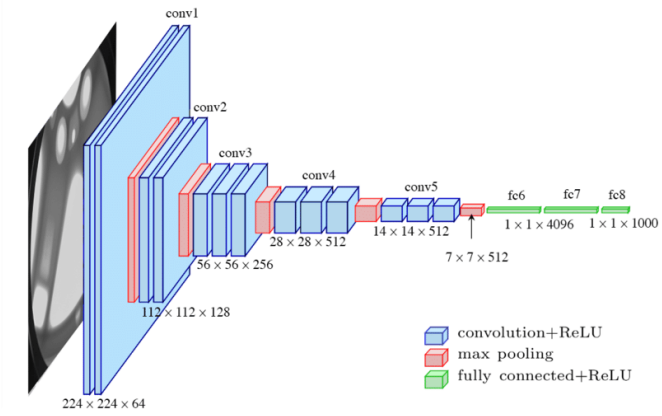


Figure 12: VGG16-based CNN architecture with custom classification head, reproduced from [16].

Output formulation. The final **softmax layer** converts raw logits z_k into class probabilities:

$$P(y = k \mid \mathbf{x}) = \frac{\exp(z_k)}{\sum_{j=1}^{K} \exp(z_j)},$$

where $K = 4$ denotes the number of diagnostic categories. The model was trained using the **categorical cross-entropy** loss:

$$\mathcal{L} = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{i,k} \log \hat{y}_{i,k},$$

where $y_{i,k}$ is the one-hot encoded ground truth for class k, and $\hat{y}_{i,k}$ is the predicted probability.

Training configuration. Inputs were normalized to the $[0, 1]$ range as part of the preprocessing pipeline, and the same scaling was applied internally within the training wrapper to maintain consistency across runs.

During the **hyperparameter search phase**, the convolutional base of VGG16 remained frozen to reduce variance and ensure stable transfer learning. Partial fine-tuning was optionally enabled through the categorical hyperparameter `unfrozen_blocks` $\in \{0, 1, 2\}$, corresponding to progressively deeper levels of the convolutional stack being unfrozen.

Optimization employed the **Adam** optimizer with a learning rate sampled logarithmically from the specified search space, while the dense layer size, dropout rate, and batch size were determined by the candidate configuration proposed by the optimizer.

To guarantee fair comparison between optimizers, each candidate model was trained for a fixed number of epochs without early stopping during the search stage. Early stopping was reserved exclusively for the final retraining phase (train+validation $\rightarrow$ test) to ensure that every candidate consumed an equal and predefined number of function evaluations, thereby isolating the effect of the optimization algorithm itself.

Rationale for VGG16 selection. VGG16 was selected due to its architectural simplicity, interpretability, and consistent performance in transfer learning studies within medical imaging. Its use of uniform 3×3 convolution kernels and moderate network depth strikes an optimal balance between representational power and computational efficiency. These characteristics make it particularly suitable for systematic hyperparameter optimization studies, where the goal is to evaluate optimizer behavior rather than to maximize absolute model complexity.

2.5 Hyperparameters and Search Space

The hyperparameter optimization focused on six key variables that jointly influence model capacity, regularization strength, convergence behavior, and transfer-learning depth. The corresponding search space $\mathcal{H}$ was defined as follows:

- **Dense units:** number of neurons in the fully connected layer, sampled uniformly from $[128, 512]$. This parameter controls model expressiveness and directly affects the classifier's capacity to model nonlinear decision boundaries.
- **Dropout rate:** probability of neuron deactivation during training, sampled continuously from $[0.25, 0.55]$. Dropout reduces co-adaptation of neurons and mitigates overfitting, especially under limited data conditions.

- **Learning rate:** base step size for gradient updates, sampled on a logarithmic scale from $[10^{-5}, 10^{-4}]$. Logarithmic sampling provides better coverage across multiple orders of magnitude, enabling both exploratory and fine-grained learning behavior.

- **Batch size:** number of samples per gradient update, selected from the discrete set $\{16, 32\}$. Smaller batches improve generalization by introducing gradient noise, while larger batches improve training stability and GPU efficiency.

- L_2 **weight decay:** coefficient for L_2 regularization applied to trainable parameters, sampled from $[10^{-6}, 10^{-4}]$. This penalty constrains weight magnitudes to reduce overfitting and promotes smoother optimization landscapes.

- **Unfrozen convolutional blocks:** categorical variable `unfrozen_blocks` $\in \{0, 1, 2\}$ controlling the extent of fine-tuning. When set to 0, the pretrained convolutional base remains fully frozen; when 1 or 2, the last one or two convolutional blocks are unfrozen, respectively, allowing gradual adaptation of deeper features to the MRI domain.

The **number of training epochs** was initially considered for optimization but was later fixed at 8 epochs per evaluation to ensure a consistent computational budget across algorithms. Allowing optimizers to vary epochs could bias the results, as longer training durations might artificially improve validation accuracy. This uniform training schedule ensures that each hyperparameter configuration consumes an equal number of evaluations, maintaining fairness between optimization algorithms.

The search bounds were selected based on prior studies in medical image classification using VGG16-based architectures [26, 18, 14], as well as empirical pilot runs and GPU memory constraints. The resulting configuration space $\mathcal{H}$ provided sufficient diversity to evaluate optimizer performance under realistic yet computationally feasible conditions.

Evaluation Wrapper and Objective Function

All optimization algorithms—Particle Swarm Optimization (PSO) and Firefighter Optimization (FFO)—shared a unified evaluation entry point implemented as:

`evaluate_model(hparams, train_X, val_X).`

This function dynamically builds the CNN according to the hyperparameter set `hparams`, trains the model on the training subset, validates it on the validation subset, and returns the resulting validation accuracy as a scalar fitness value.

The optimization objective was defined as a black-box maximization problem:

$$\max_{\theta \in \mathcal{H}} \ \mathrm{Acc}_{\mathrm{val}}(\theta),$$

where θ represents a candidate hyperparameter vector sampled from the search space $\mathcal{H}$, and Acc_{val} denotes the mean validation accuracy achieved under θ.

By centralizing all training, validation, and logging operations within a single evaluation wrapper, the implementation ensures strict modularity and reproducibility. Each optimizer—PSO and FFO—interacts identically with this interface, guaranteeing that performance differences reflect only the efficiency of the search mechanism and not discrepancies in model construction or data handling.

2.6 Metaheuristic Optimization Setup

Particle Swarm Optimization (PSO). In PSO, each candidate solution is represented as a "particle" in the search space, with a position x_i (the current hyperparameter vector) and a velocity v_i (the search step). At every iteration, the velocity update balances three forces:

- **Inertia term** ωv_i^t: preserves part of the previous velocity, encouraging momentum and large-scale exploration. A higher ω promotes global search, while a smaller value favors local refinement.
- **Cognitive component** $c_1 r_1 (p_i - x_i^t)$: pulls the particle toward its own historically best position p_i; it reflects exploitation of past personal experience.
- **Social component** $c_2 r_2 (g - x_i^t)$: pulls the particle toward the global best position g found by the swarm; it ensures information sharing and collective convergence.

The update rule is:

$$v_i^{t+1} = \omega v_i^t + c_1 r_1 (p_i - x_i^t) + c_2 r_2 (g - x_i^t), \quad x_i^{t+1} = x_i^t + v_i^{t+1},$$

where $r_1, r_2 \sim U(0, 1)$ inject stochasticity.

For this thesis, parameters were set to:

- **Swarm size:** 10 particles. Each iteration evaluates 10 distinct candidate hyperparameters.
- **Iterations:** 10. The search is capped at 100 updates plus the initial swarm evaluation, resulting in about 110 function evaluations.
- **Inertia weight:** $\omega = 0.7$, a commonly used balance between exploration and exploitation.
- **Acceleration coefficients:** $c_1 = c_2 = 1.5$, giving equal weight to personal and social experience, encouraging both independent exploration and convergence toward the best solutions.

Firefighter Optimization Algorithm (FOA). Unlike PSO, FOA integrates several operators:

- **Population of agents:** 10 candidate solutions are maintained per iteration.
- **Crossover (probability 0.5):** pairs of agents exchange parts of their solution vectors, analogous to genetic algorithms, enabling exploration of new combinations of hyperparameters.
- **Mutation/local search (probability 0.7):** each agent may undergo Gaussian perturbations, with multiple "tries" per mutation, which allow the algorithm to refine solutions and escape local optima.
- **Annealing schedule:** acceptance of worse solutions is governed by a simulated annealing probability, with $T_{t+1} = \gamma T_t$ and $\gamma = 0.95$. This gradually reduces randomness, shifting from exploration to exploitation.
- **Stagnation perturbations:** if no improvement occurs after 12 iterations, perturbations are added to agents relative to the global best, increasing diversity and preventing premature convergence.

Fairness and Function Evaluation (FE) Budget. Both FOA and PSO were applied to the same objective function and search space, with identical seeds to ensure reproducibility. To avoid bias, termination was governed not just by the product of agents and iterations but also by a global function evaluation (FE) budget. This was essential because FOA tends to generate many more evaluations per iteration than PSO due to its crossover and mutation operators, which could otherwise give it an unfair advantage by simply testing more models.

The expected evaluation count can be derived analytically. For a population size N, the Firefighter Optimization (FOA) step evaluates:

$$\underbrace{N}_{\text{baseline population}} + \underbrace{N\,p_{\mathrm{x}} \cdot 2}_{\text{two crossover children}} + \underbrace{N\,p_{\mathrm{m}} \cdot (1 + \texttt{tries})}_{\text{local search: current+proposals}} = N\big[\,1 + 2p_{\mathrm{x}} + (1 + \texttt{tries})\,p_{\mathrm{m}}\,\big].$$

With the defaults $p_{\mathrm{x}} = 0.4$, $p_{\mathrm{m}} = 0.7$, and $\texttt{tries} \approx 6$,

$$\text{FOA evals/iter} \;\approx\; N\,[\,1 + 2(0.4) + 7(0.7)\,] \;=\; 6.7\,N.$$

For Particle Swarm Optimization (PSO), the expected cost per iteration is much lower:

$$\text{PSO evals/iter} \;\approx\; N \quad \text{(after an initial } N \text{ to evaluate the swarm).}$$

To deeper understand the varied differences, we can calculate concrete totals for $N = 10$ agents and $T = 10$ iterations. Including the common initial evaluation of the

starting population:

$$\text{PSO total FEs} \approx N \;+\; T \cdot N \;=\; 10 \;+\; 10 \cdot 10 \;=\; \mathbf{110},$$

$$\text{FOA total FEs} \approx N \;+\; T \cdot (6.7\,N) \;=\; 10 \;+\; 10 \cdot (6.7 \cdot 10) \;=\; 10 \;+\; 670 \;=\; \mathbf{680}.$$

Notes. (i) The FOA expectation depends directly on `tries` and the operator probabilities; if these values are adjusted, the formula scales accordingly via $N[1 + 2p_{\mathrm{x}} + (1 + \texttt{tries})p_{\mathrm{m}}]$. (ii) Some PSO implementations count only $T \cdot N$ evaluations (omitting the initial population), in which case PSO would report 100 FEs for N=10, T=10.

Optimizer CLIs, budget control, caching

FFO CLI.

```
python src/ffo.py --mode full --data-dir . --results-dir runs/ffo_final \
  --agents 10 --iters 10 --eval-budget 110 --seed 42 \
  --dense-min 128 --dense-max 512 --dropout-min 0.25 --dropout-max 0.55 \
  --lr-min 1e-5 --lr-max 1e-4 --batch-min 16 --batch-max 32 \
  --epochs-fixed-full 8 --subset-frac 0.25 --cache .ffo_cache.json
```

The `--eval-budget` hard-stops the run once a total number of function evaluations is reached. A JSON *objective cache* can be supplied (e.g., `.ffo_cache.json`) so that duplicate hyperparameter vectors are not re-evaluated, preserving the budget.

PSO CLI.

```
python src/pso.py --mode full --data-dir . --results-dir runs/pso_final \
  --particles 10 --iters 10 --seed 42 \
  --dense-min 128 --dense-max 512 --dropout-min 0.25 --dropout-max 0.55 \
  --lr-min 1e-5 --lr-max 1e-4 --batch-min 16 --batch-max 32 \
  --epochs-fixed-full 8 --subset-frac 0.25
```

Logging and artifacts. Each run is stored under `runs/<algo>_<mode>_<timestamp>/` with:

- `space.json` (search bounds), `config.json` (algorithm settings),
- `result.json` (best hyperparameters, best validation accuracy, fitness curve),
- optional `objective_cache.json` (for FFO),
- stdout `.out` files (cluster logs) with per-evaluation traces.

2.7 Experimental Protocol

Two operational modes were defined:

1. **Test mode**: quick runs with reduced subsets and epochs for debugging.
2. **Full mode**: full experiments with the complete dataset and search space.

After optimization, the best hyperparameters were retrained using `train_best_from_result.py`, training on train+val and evaluating on the test set. Metrics collected included accuracy, confusion matrices, precision, recall, F1-score, and AUC. Curves for convergence, calibration, and precision-recall were also generated.

Performance metrics were computed as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN},$$

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}, \quad \text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

$$\text{AUC} = \int_0^1 TPR(FPR)\, dFPR,$$

where TPR and FPR are the true and false positive rates.

To account for randomness, multiple seeds were used and results averaged. Logs and outputs were versioned and archived in the `runs/` directory.

Scripted pipeline walkthrough (added details)

1) Optimization phase. Running FFO/PSO produces a directory:

```
runs/ffo_final/ffo_full_YYYYMMDD_HHMMSS/
  space.json          # search bounds
  config.json         # algo hyperparameters
  result.json         # best_hparams, best_val_accuracy, history
  objective_cache.json (optional)
```

`result.json` is the single source of truth for the best configuration.

2) Retraining phase. The best configuration is reloaded and trained on train+val, then evaluated on test to see if we will actually get to good results after exploring the search space:

```
python src/train_best_from_result.py \
  --run-dir runs/ffo_final/ffo_full_... --data-dir . \
  --class-names glioma_tumor meningioma_tumor pituitary_tumor no_tumor \
  --epochs-override 40 --patience 5
```

Artifacts written back into the same run directory:

- `history.csv` (per-epoch loss/accuracy),
- `y_true.npy`, `y_pred.npy`, `y_prob.npy`,
- `final_test_metrics.json` (accuracy, macro/micro F1, AUC, per-class metrics).

3) Reporting phase. Aggregated plots and summaries for visualization of results and comparing the two algorithms:

```
python src/results_report.py \
  --results-root runs/ffo_final \
  --out reports/ffo_final \
  --class-names glioma_tumor meningioma_tumor pituitary_tumor no_tumor \
  --no-auto-final-train --algo-name FFO
```

This generates convergence curves, confusion matrices, ROC/PR curves, and calibration plots (saved under `reports/`).

2.8 Computational Environment

Experiments were conducted on the Leonardo cluster (CINECA), using a single GPU node (NVIDIA A100 SXM4 64 GB), 32 CPU cores, and 512 GB RAM. Software environment:

- Python 3.10
- TensorFlow/Keras 2.11+
- NumPy, Pandas, Matplotlib, OpenCV
- Scikit-learn for metrics and stratified splits

Jobs were submitted via Slurm workload manager using `sbatch` scripts, specifying GPU resources, CPU threads, and wall-time limits. This ensures full reproducibility on HPC environments, as the scripts log both the job configuration and system environment.

Reproducibility was further ensured by explicit seeding of Python, NumPy, and TensorFlow random generators. All preprocessing and training scripts are deterministic, and configurations are logged to JSON files.

Summary. Together, these components constitute a fully reproducible experimental framework. Every phase—dataset preparation, model training, hyperparameter optimization, retraining, and reporting—is modular, version-controlled, and traceable. This ensures that all results presented in subsequent chapters can be exactly replicated and independently verified.

Table 2: Distribution of samples in the Kaggle Brain Tumor MRI dataset after a 70/15/15 stratified split.

Class	**Train**	**Validation**	**Test**
Glioma	648	139	139
Meningioma	656	141	140
Pituitary	630	136	135
No tumor	350	75	75

3 Results

3.1 Convergence Behaviour of FFO and PSO

The convergence analysis illustrates how the Firefighter Optimization Algorithm (FFO) and Particle Swarm Optimization (PSO) navigate the hyperparameter search space under identical evaluation budgets. Figure 13 summarizes the aggregated convergence trends across all runs, while Figure 14 visualizes the corresponding individual optimization trajectories.

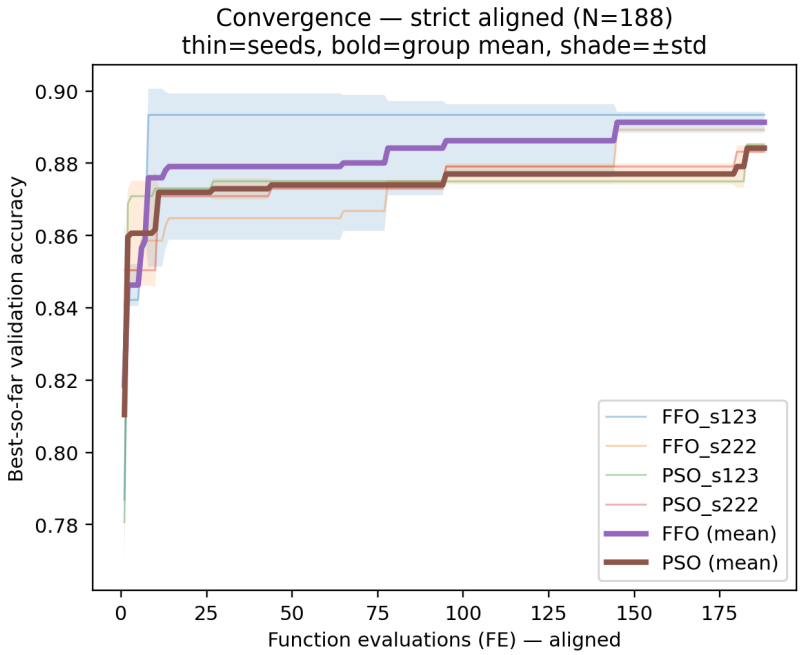


Figure 13: Mean convergence behaviour of FFO and PSO under a strictly aligned evaluation budget (N=188). Thin lines represent individual seeds; bold curves denote group means; shaded areas indicate one standard deviation.

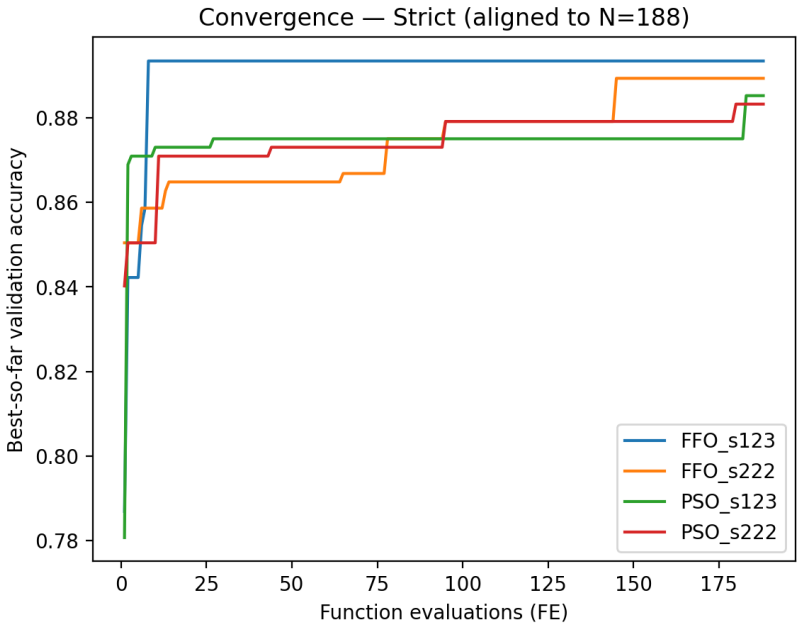


Figure 14: Individual convergence trajectories for FFO and PSO under the same evaluation budget (N=188). Each curve corresponds to a distinct random seed.

In the grouped mean plot (Figure 13), both optimizers exhibit a sharp initial rise in validation accuracy followed by a gradual stabilization phase, indicating that each algorithm can quickly identify promising regions of the search space. FFO achieves a steeper ascent in the early iterations and plateaus sooner, reflecting rapid exploitation of high-performing configurations once discovered rather than broad exploration. In contrast, PSO advances at a steadier pace, improving more gradually as swarm information sharing continues to refine solutions.

The narrow shaded variance bands indicate consistent convergence dynamics across different random seeds, suggesting both algorithms are stable and relatively insensitive to initialization. Minor differences in the slope of the curves, however, imply that FFO's search is more front-loaded—making significant progress in fewer function evaluations—while PSO distributes its progress more evenly across the evaluation budget.

The per-run overlays in Figure 14 reveal these differences more clearly. FFO runs typically reach high validation accuracies within the first third of the budget and then stabilize, showing limited deviation between trajectories. This pattern indicates that FFO tends to commit early to a favourable basin and focus its resources on local refinement. PSO trajectories, by contrast, rise more gradually but continue improving late into the budget, consistent with its collective and iterative exploitation of information shared across particles.

Overall, the convergence profiles highlight two complementary behaviours. FFO demonstrates greater sample-efficiency—achieving strong configurations with fewer evaluations—whereas PSO exhibits more sustained, steady-state improvement and

smoother convergence across seeds. In practice, this suggests FFO may be advantageous under strict computational constraints, while PSO is better suited to extended search horizons where long-term refinement can further close the performance gap. The following subsection examines how these dynamics translate into final validation performance and best-trial hyperparameter configurations.

3.2 Per-Run Analysis of Optimization Behaviour

To complement the aggregate convergence trends, representative single runs were analysed to observe how each optimizer navigated the hyperparameter space. Figures 15–20 display one-dimensional response curves and surrogate-based hyperparameter-importance rankings for the Firefighter Optimization Algorithm (FFO) and Particle Swarm Optimization (PSO), both using seed 123.

FFO_s123 — early commitment and localized refinement. In the FFO run, the learning-rate response (Figure 15) shows a dense cluster of high-accuracy trials within the range of approximately 2×10^{-4} to 3×10^{-4}, with few evaluations outside this band. The median curve forms a short, flat plateau, and the scatter remains narrow—evidence that the algorithm rapidly converged to a single promising region and then concentrated on refining it. This behaviour indicates *early exploitation* of a favourable basin rather than broad, continued exploration.

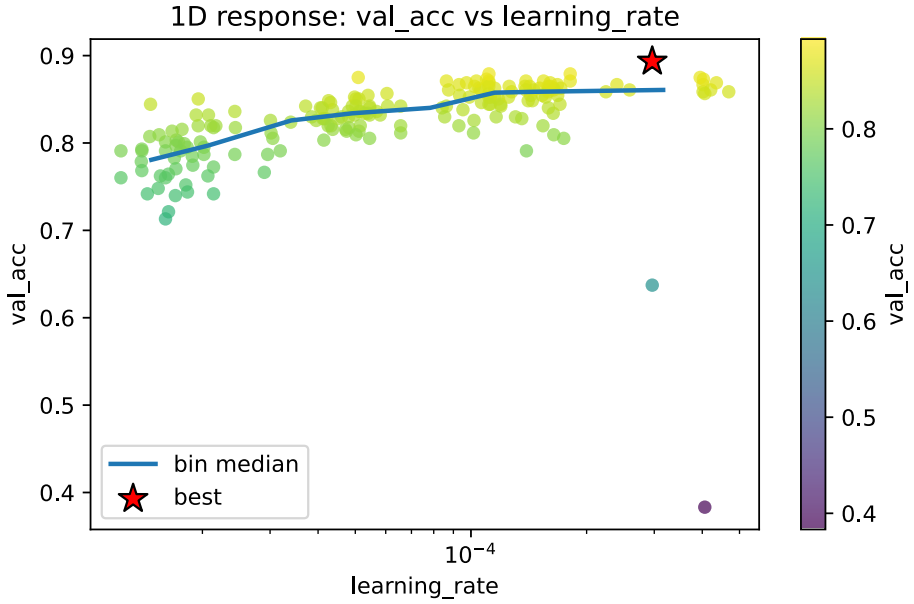


Figure 15: FFO (seed 123): one-dimensional response of validation accuracy with respect to learning rate.

For dropout (Figure 16), validation accuracy remains consistently high between

≈ 0.32 and 0.45, peaking around $0.36-0.38$. Only sparse evaluations occur beyond this corridor, and low outliers are limited. The pattern supports the interpretation that FFO's search became increasingly localized once stable hyperparameter combinations were identified.

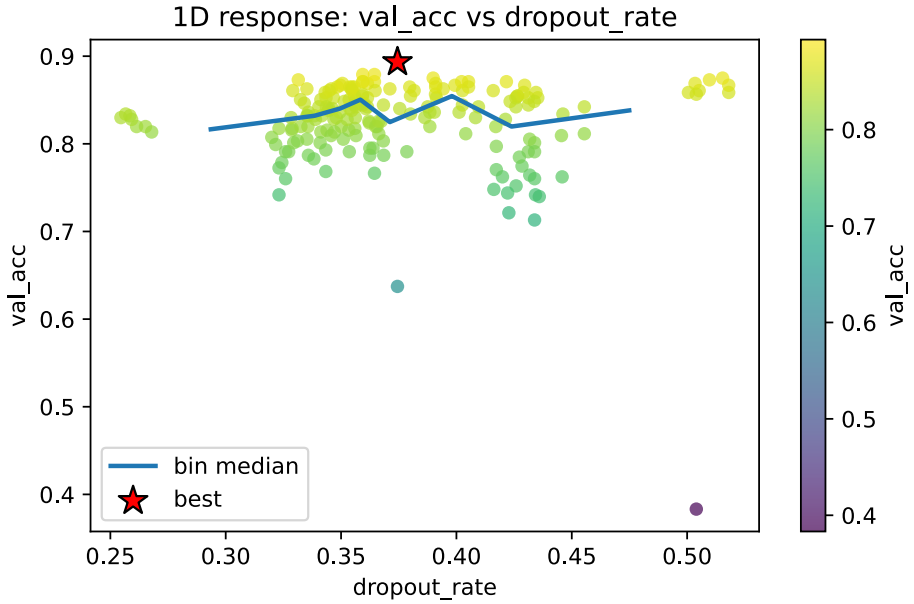


Figure 16: FFO (seed 123): one-dimensional response of validation accuracy with respect to dropout rate.

The importance ranking (Figure 17) assigns dominant weight to *learning rate*, followed by *unfrozen blocks* and *L2*, with minimal contribution from *batch size* and *dense units*. This hierarchy is consistent with FFO's front-loaded behaviour: once an optimal learning rate is located, secondary parameters are fine-tuned through small, local perturbations.

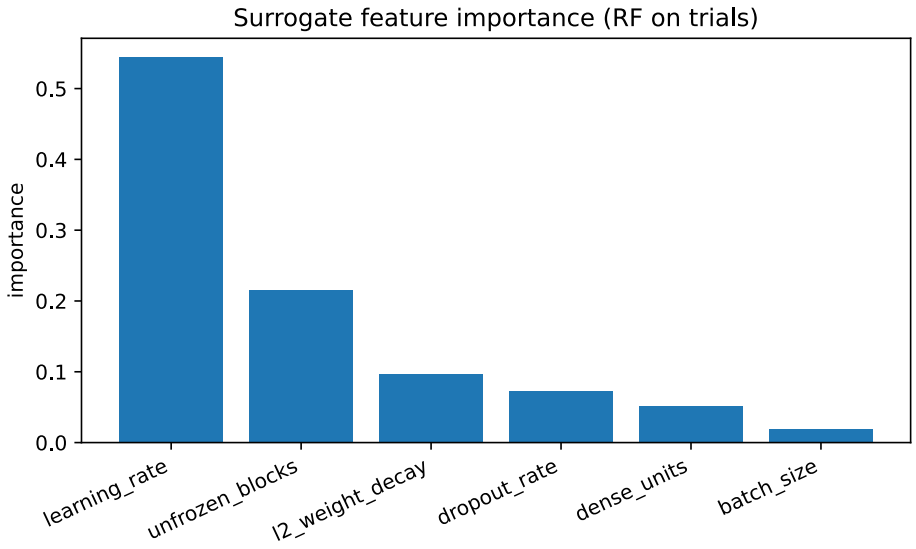


Figure 17: FFO (seed 123): relative hyperparameter importance derived from model evaluation statistics.

PSO_s123 — broader probing before consolidation. In contrast, PSO's learning-rate curve (Figure 18) spans a wider interval, with trials distributed across both low and high values. Validation accuracy peaks near $\sim 3{\times}10^{-4}$ but declines sharply beyond that, and the visible scatter at extreme rates confirms that PSO actively sampled less-favourable regions before narrowing its focus. This indicates more balanced exploration in the early stage, followed by progressive convergence toward the high-accuracy zone.

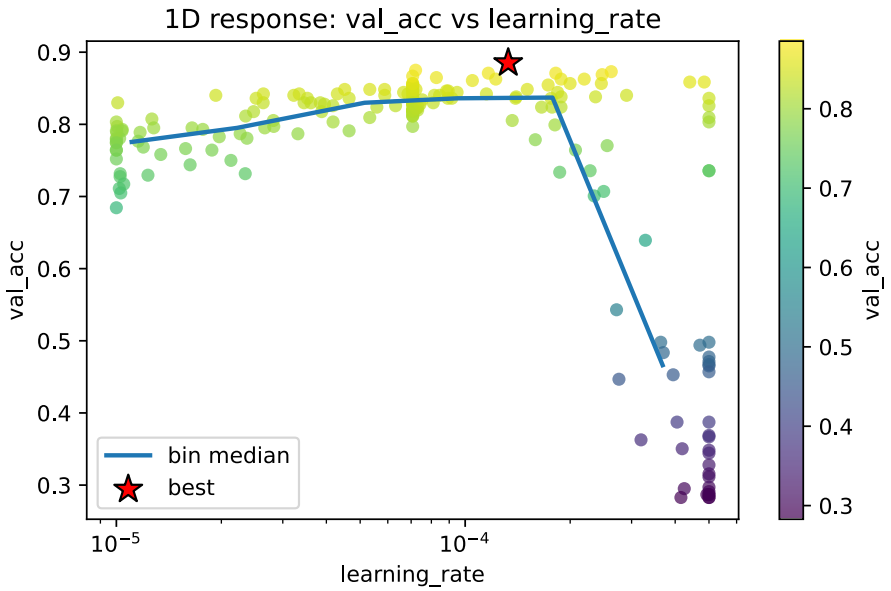


Figure 18: PSO (seed 123): one-dimensional response of validation accuracy with respect to learning rate.

For dropout (Figure 19, bottom), accuracy rises toward ~ 0.36 and then levels off; it is also obvious that PSO systematically tested boundary values as part of its exploratory phase. Compared with FFO, the spread of points is broader, particularly near range edges, consistent with PSO's collective mechanism of velocity-based adjustment and information sharing across particles.

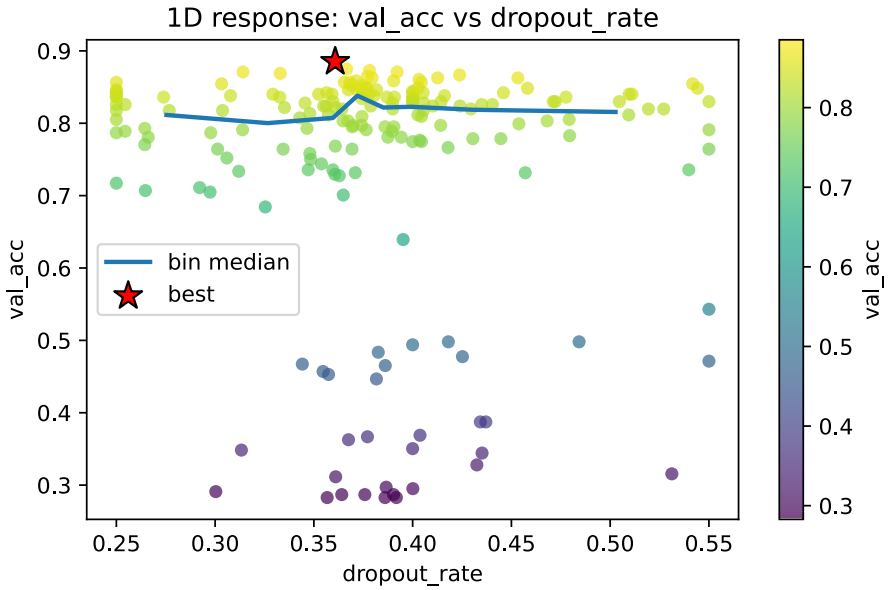


Figure 19: PSO (seed 123): one-dimensional response of validation accuracy with respect to dropout rate.

The corresponding importance ranking (Figure 20) again highlights *learning rate* as dominant in addition to *unfrozen blocks*. FFO assigned higher relative influence to *L2* and *dropout rate* than in PSO. This suggests that FFO maintained moderate attention to regularization effects while gradually consolidating around a stable configuration.

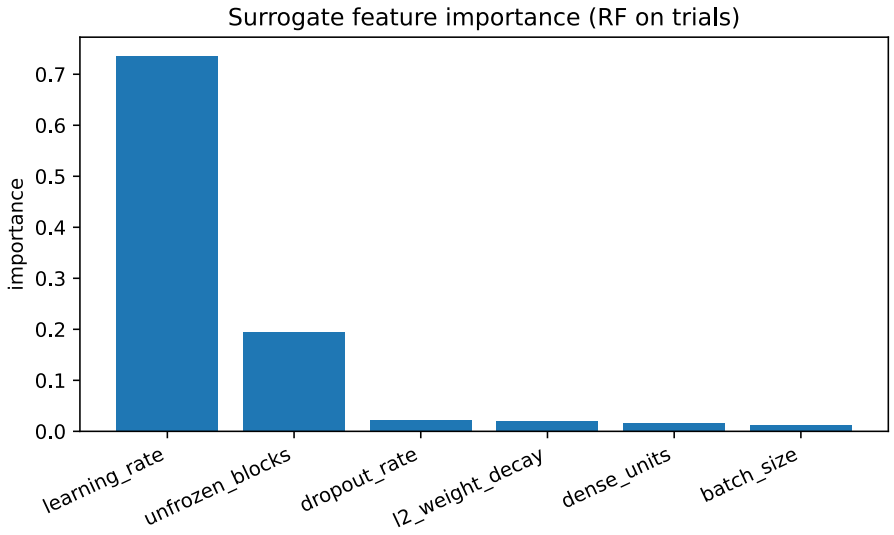


Figure 20: PSO (seed 123): relative hyperparameter importance showing influence on validation accuracy.

Comparative interpretation. Overall, the per-run analyses show that FFO and PSO exhibit distinct yet complementary search dynamics. FFO's trajectories reveal a *basin-focused*, sample-efficient pattern—rapidly exploiting a promising region but potentially overlooking alternative optima—whereas PSO displays a more *distributed exploration*, sampling a wider portion of the space before gradual consolidation. Both optimizers converge on the same critical drivers, namely *learning rate* and *dropout*, but PSO's broader sensitivity to *batch size* and *L2* reflects its more adaptive fine-tuning once a viable region is identified.

These qualitative trends align with the convergence results of Section 3.1: FFO achieves early, high accuracy through intensive local refinement, while PSO continues to improve later in the search via collective, steady adjustments. Quantitatively, both reach comparable peak validation accuracies ($\approx 0.95-0.98$), differing primarily in the smoothness and diversity of their trajectories.

3.3 Final Best-Hyperparameter Training

Following the hyperparameter optimization stage, the best configurations discovered by each optimizer were retrained on the full data set and evaluated on an unseen test partition. This step tests whether the optimized hyperparameters generalize to new data and enables a direct comparison of the final predictive behaviour of the Firefighter Optimization Algorithm (FFO) and Particle Swarm Optimization (PSO). We report confusion matrices, ROC and precision–recall (PR) curves, and calibration plots, which together characterise discrimination, error patterns by class, and the reliability of predicted probabilities.

Evaluation metrics. The **confusion matrix** shows correct and incorrect predictions per class; diagonal dominance indicates accurate classification, while off-diagonals identify specific confusions. The **ROC** curve plots true positive rate versus false positive rate across thresholds; the area under the curve (**AUC**) quantifies threshold-independent discrimination, with values closer to 1.0 indicating stronger separability. The **precision–recall (PR)** curve emphasises performance under imbalance; the area under the PR curve (**average precision, AP**) captures how well precision is sustained as recall increases. Finally, the **calibration curve** compares predicted confidence with empirical accuracy; well-calibrated models follow the diagonal, overconfident models curve below, and underconfident models above.

Firefighter Optimization (FFO), seed 123

Search convergence. Figure 21 shows a rapid improvement in validation accuracy during the first few iterations, followed by minor fluctuations and eventual stabilization around 0.97. This pattern indicates effective exploration and convergence of the search process, with no signs of instability or overfitting during later iterations.

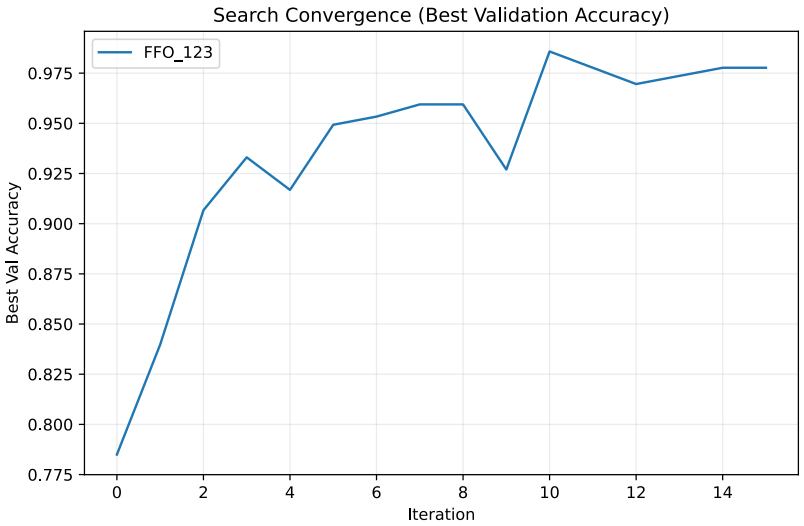


Figure 21: FFO_s123: search convergence (validation behaviour).

Confusion matrix. The matrix in Figure 22 exhibits a clearly diagonal pattern across all four classes, with only minor misclassifications observed in the *pituitary* category, occasionally overlapping with *glioma* or *meningioma* predictions. This indicates balanced per-class performance and well-separated decision boundaries on the held-out test set.

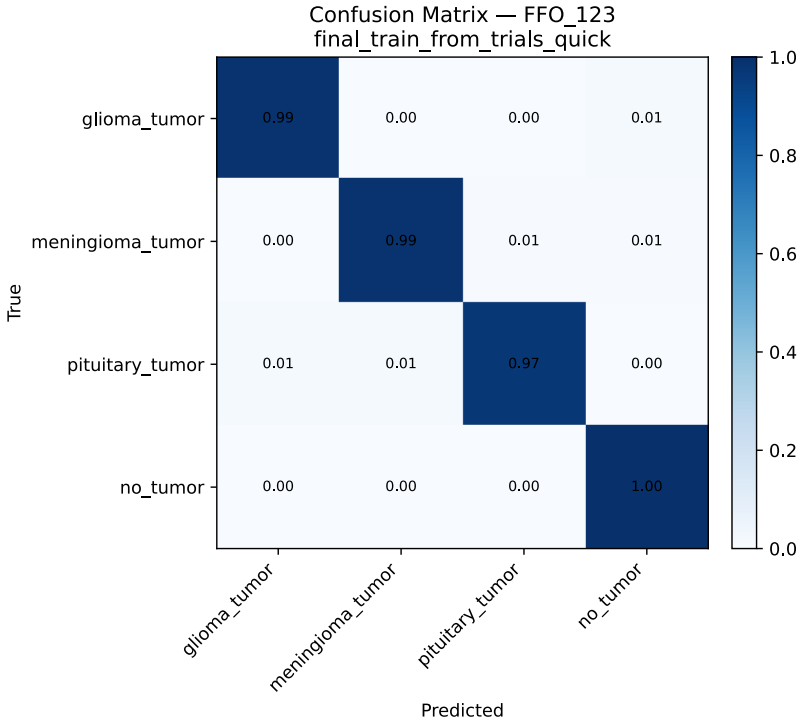


Figure 22: FFO_s123: confusion matrix on the held-out test set.

ROC and PR curves Fig23 All classes achieve near-ceiling discrimination (AUC ≈ 0.998–1.000), while the precision–recall curves remain within the upper envelope (AP ≈ 0.996–1.000). The near-rectangular curve shapes indicate consistently high recall without loss of precision across a broad threshold range, confirming robust separability between classes.

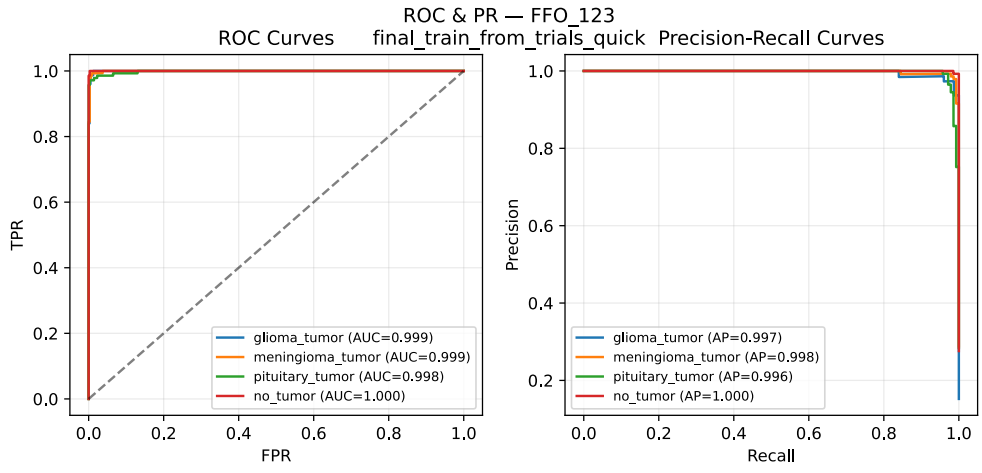


Figure 23: FFO_s123: ROC (discrimination) and PR (precision–recall) curves per class.

Calibration. The reliability curve in Figure 24 closely follows the diagonal across confidence bins, indicating well-calibrated probability outputs. This alignment suggests that predicted confidence values correspond closely to actual accuracies, an essential property for risk-sensitive clinical interpretation.

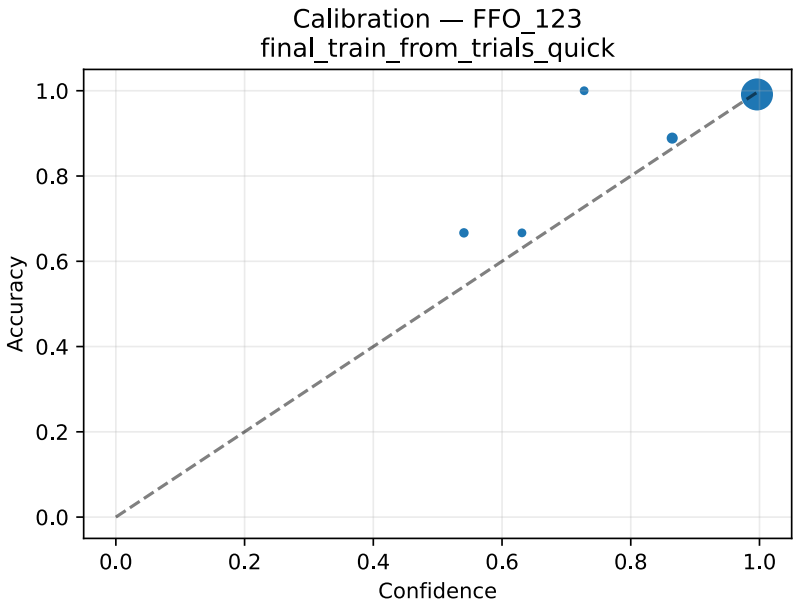


Figure 24: FFO_s123: calibration (reliability) plot, showing close alignment with the ideal diagonal.

Firefighter Optimization (FFO), seed 222

Search convergence. The curve 25 plateau slightly later than in seed 123, indicating slower but more gradual convergence. Mild oscillations at the upper accuracy range suggest continued fine-tuning around a narrower local optimum before stabilization.

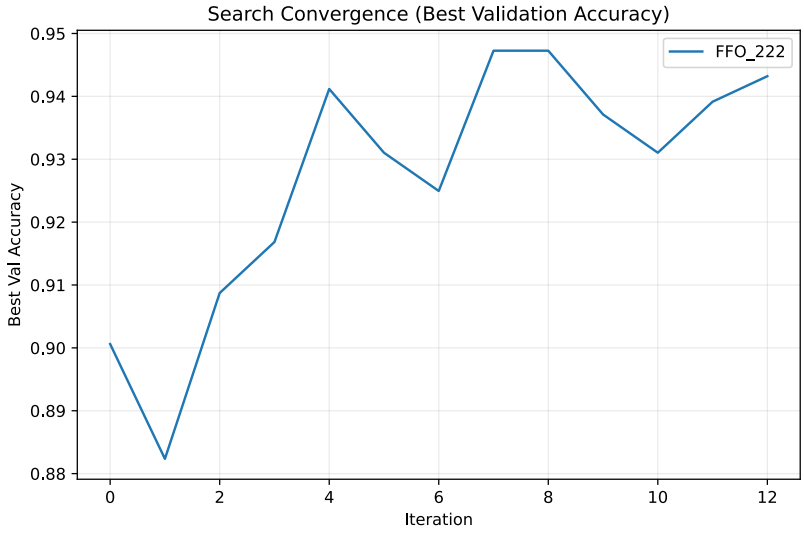


Figure 25: FFO_s222: search convergence, showing rapid stabilization and a narrow final optimization basin.

Confusion matrix. The matrix in Figure 26remains predominantly diagonal, with a slight increase in *pituitary* ↔ *meningioma* confusion. The *glioma* and *no_tumor* classes remain nearly perfect, indicating strong separability under this training configuration.

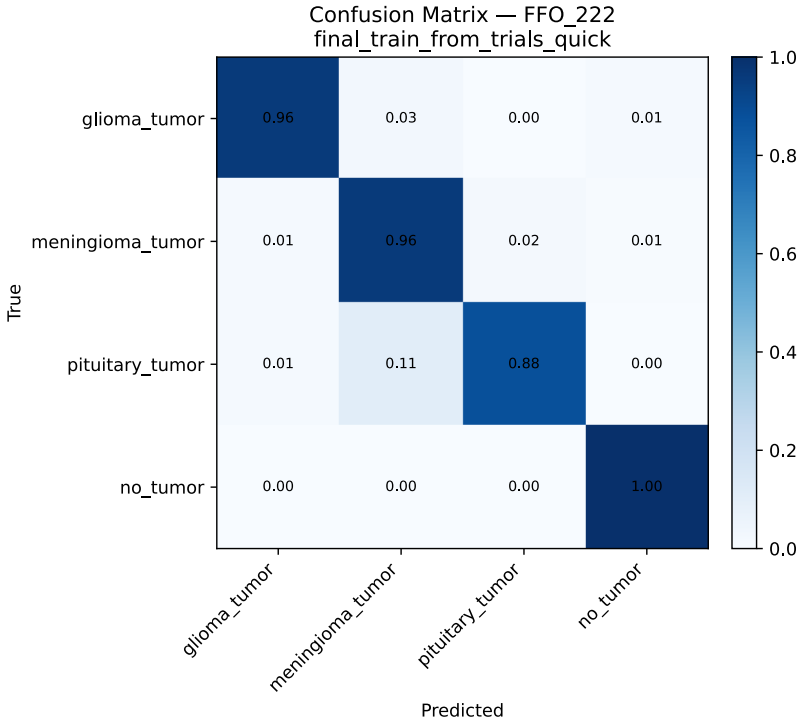


Figure 26: FFO_s222: confusion matrix showing increased overlap between pituitary and meningioma tumors.

ROC and PR curves. In Figure 27, AUC remains high across all classes (> 0.989), with a slightly less convex ROC curve for *meningioma* relative to seed 123. The PR curves maintain high precision across a wide recall range (AP ≥ 0.97), indicating strong discriminative performance and a high-quality local optimum distinct from the first seed.

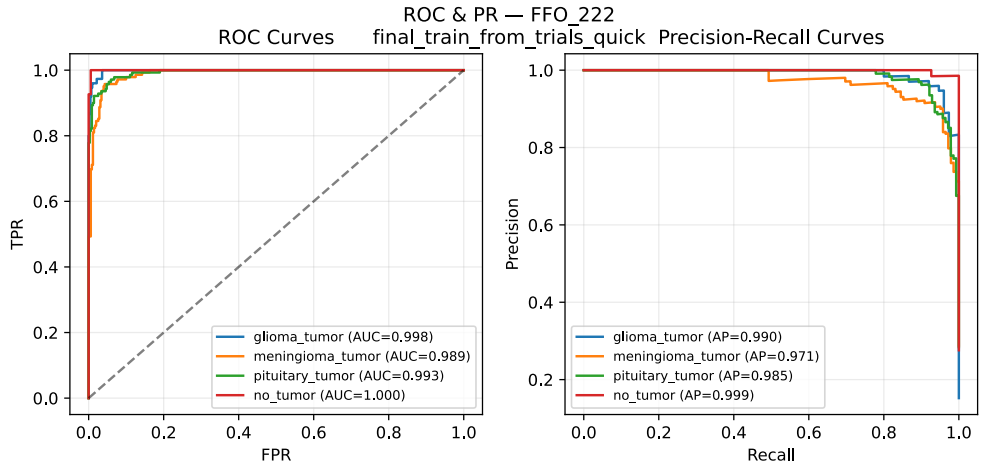


Figure 27: FFO_s222: ROC and PR curves, showing consistent class-wise discrimination with a modest reduction for meningioma.

Calibration. The points in Figure 28 follows the diagonal closely, with a shallow dip around the 0.6–0.8 confidence range, indicating mild mid-range overconfidence.

Overall calibration remains strong, suggesting reliable probability estimates across most confidence levels.

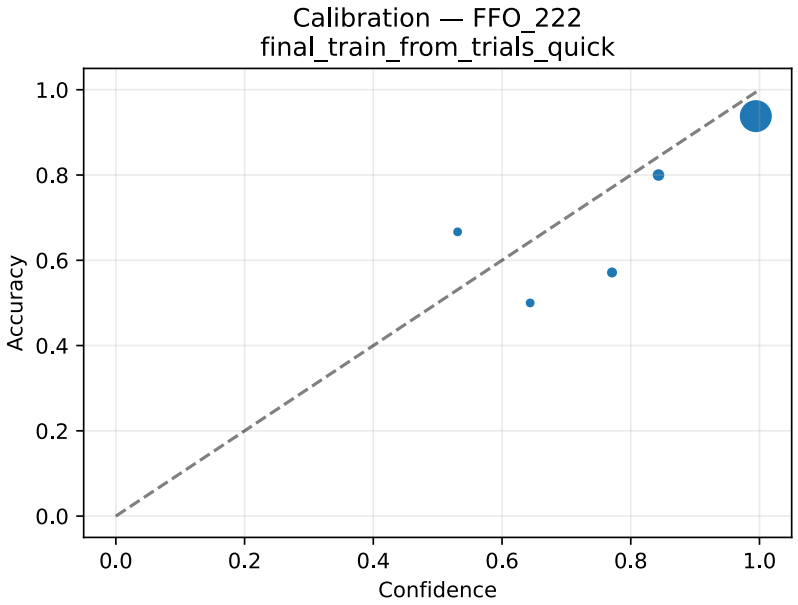


Figure 28: FFO_s222: calibration curve showing minor overconfidence at intermediate confidence levels.

Particle Swarm Optimization (PSO), seed 123

Search convergence. Convergence in Figure 29 is more gradual than FFO and notably smooth, with a small and stable train–validation gap—consistent with steady exploitation and stable generalization dynamics.

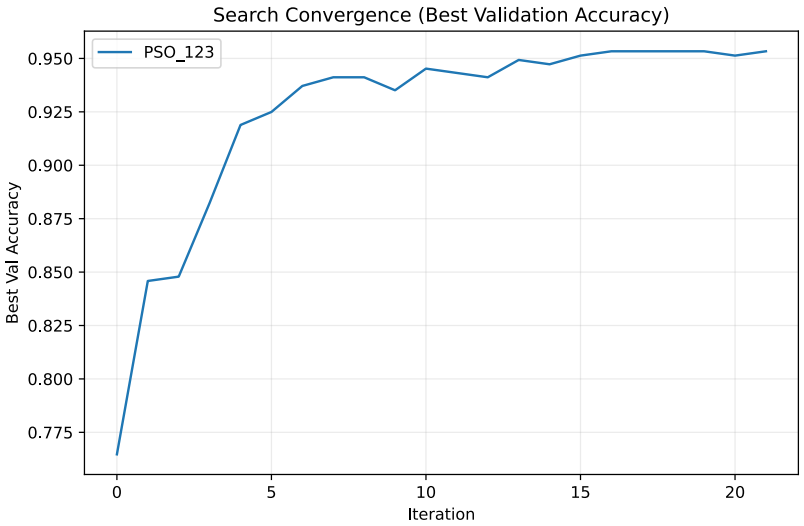


Figure 29: PSO_s123: search convergence curves showing steady and smooth optimization.

Confusion matrix. The matrix in Figure 30 is predominantly diagonal, with minor

glioma ↔ *meningioma* and occasional *pituitary* overlap. These limited errors suggest robust boundaries with balanced class performance.

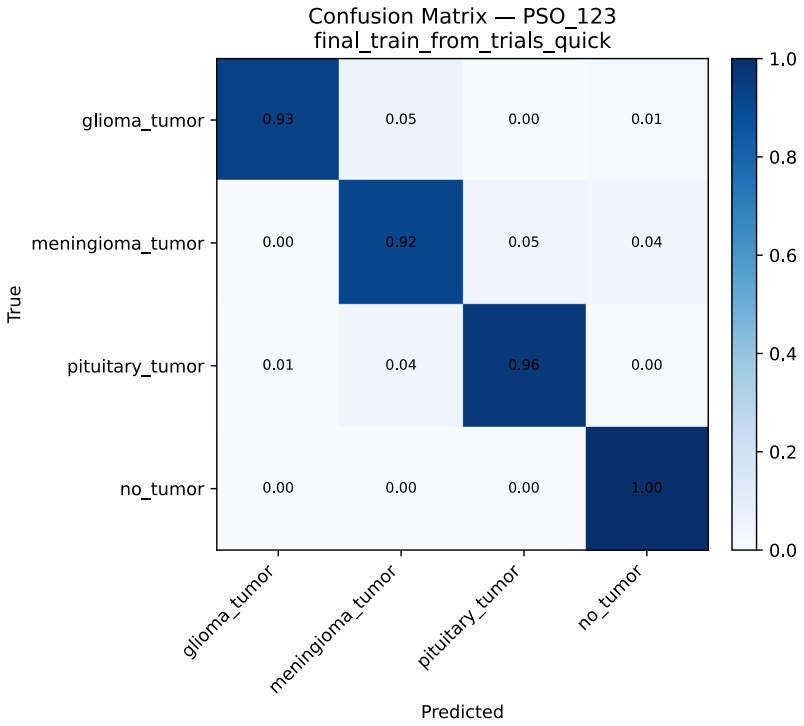


Figure 30: PSO_s123: confusion matrix showing highly diagonal structure and limited cross-class errors.

ROC and PR curves. Class-wise ROC curves as shown in Figure 31 approach the upper-left corner (AUC ≈ 0.990–0.999), and PR curves (AP ≈ 0.979–0.997) are slightly less extreme than FFO_s123 but smoother—consistent with PSO's gradual refinement.

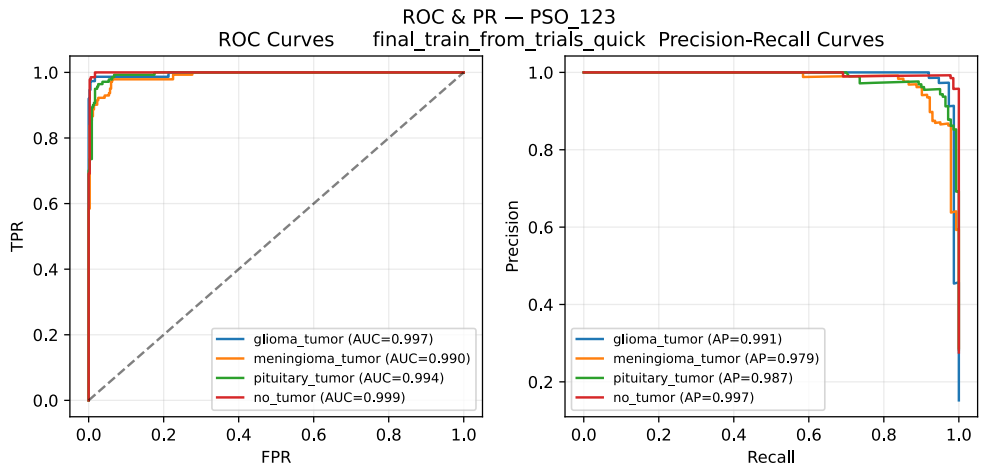


Figure 31: PSO_s123: ROC and PR curves showing high discrimination with smoother precision–recall trade-offs.

Calibration. The reliability curve in Figure 32 lies almost exactly on the diagonal across

bins, indicating accurate probability estimates and slightly less overconfidence than FFO on this seed.

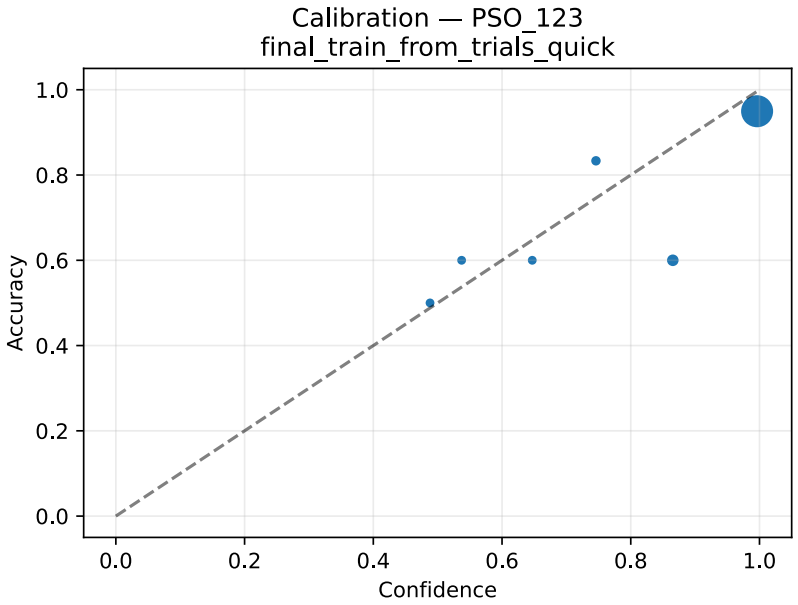


Figure 32: PSO_s123: calibration plot showing accurate alignment between predicted confidence and true correctness.

Particle Swarm Optimization (PSO), seed 222

Search convergence. Convergence in Figure 33 remains stable overall but is slightly less smooth than seed 123, showing minor oscillations near the upper accuracy range. This suggests comparable overall performance but with a slower and less uniform convergence trajectory.

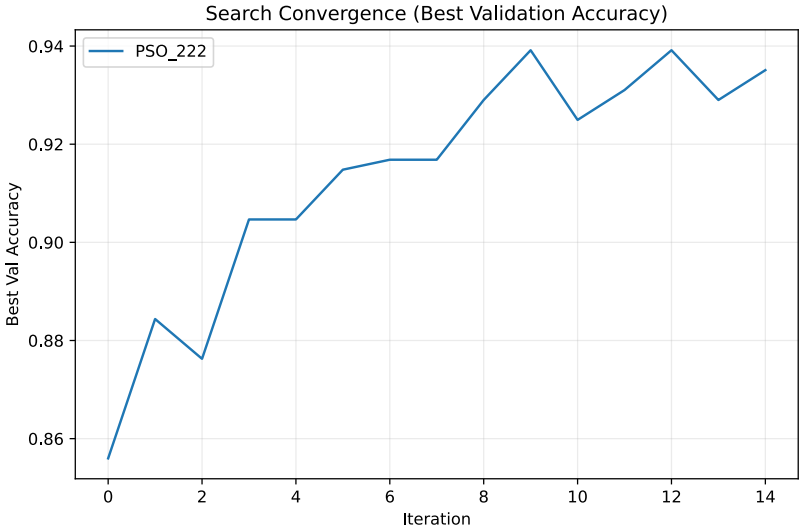


Figure 33: PSO_s222: search convergence showing exceptionally stable training behaviour.

Confusion matrix. Diagonal dominance persists in Figure 34, with minor *pituitary* ↔ *meningioma* overlap. *No_tumor* remains perfectly separated, consistent with strong specificity for healthy tissue.

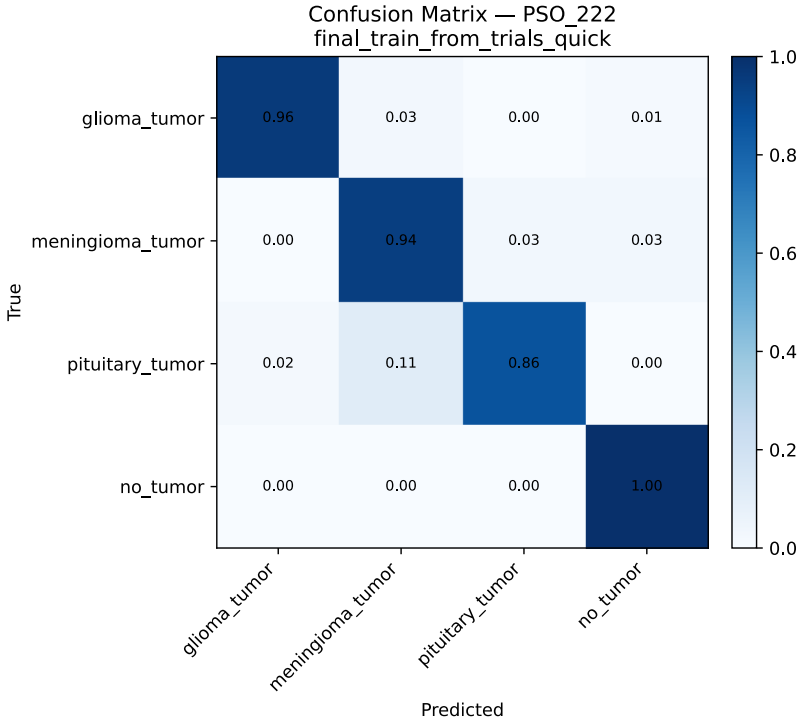


Figure 34: PSO_s222: confusion matrix confirming high accuracy and minor overlap in pituitary–meningioma region.

ROC and PR curves. In Figure 35All classes achieve high discrimination (AUC > 0.984), with *no_tumor* near 1.000; PR curves show AP ≥ 0.956, indicating sustained precision at high recall. Curve shapes are gentle, reflecting a stable sensitivity–specificity balance.

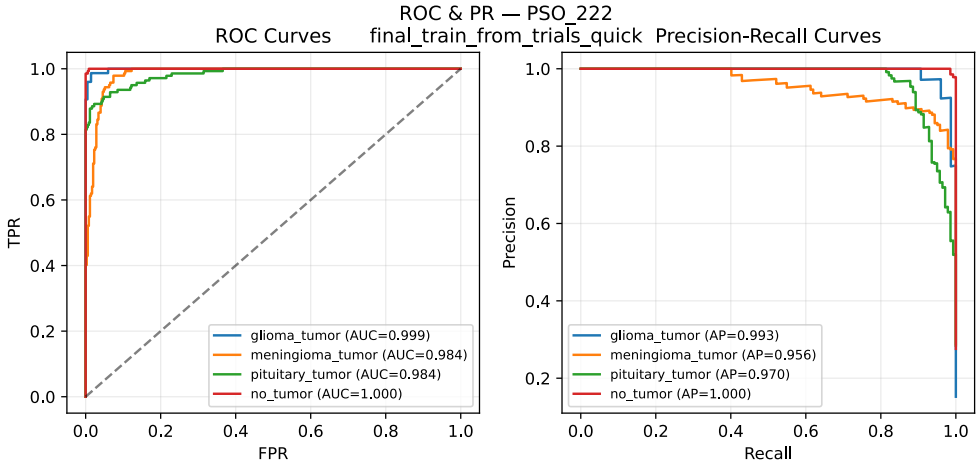


Figure 35: PSO_s222: ROC and PR curves confirming consistent class-wise discrimination across the test set.

Calibration. The curve shown in Figure 36 aligns closely with the diagonal across all bins, indicating reliable probability estimates. Among the evaluated runs, PSO_s222 exhibits the most consistent calibration.

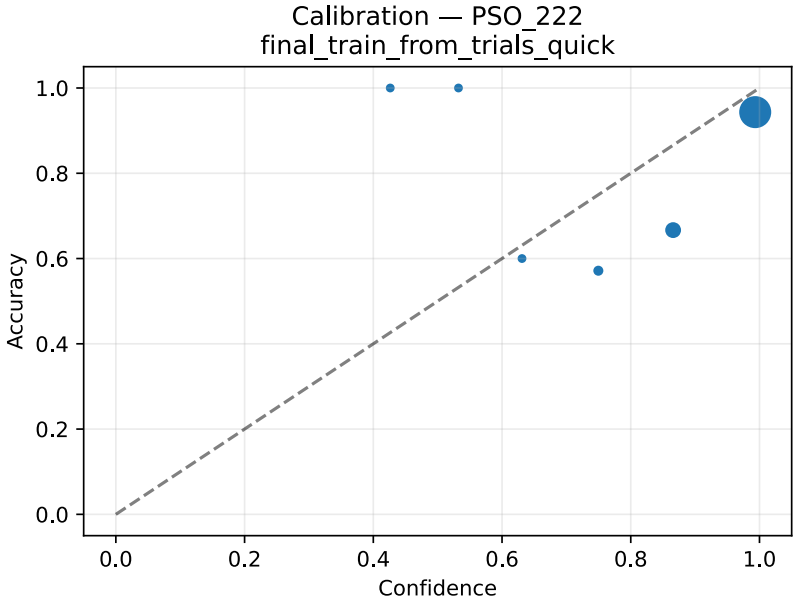


Figure 36: PSO_s222: calibration curve showing nearly perfect reliability.

Overall comparison. Across all final runs, both FFO- and PSO-tuned models achieve strong performance (AUC ≥ 0.984; AP ≥ 0.956) with small calibration error. FFO_s123 attains the highest observed discrimination and balanced per-class behaviour, whereas PSO_s222 exhibits the most consistent probability calibration. In brief:

- **FFO**: reaches high discrimination quickly and sometimes peaks higher, with occasional mid-range overconfidence depending on seed.
- **PSO**: improves more gradually and consistently, yielding smoother convergence and, across seeds, slightly better calibration.

Both optimizers discover hyperparameter configurations that generalize beyond validation, supporting the use of metaheuristic search for CNN tuning in brain tumor MRI classification.

3.4 Aggregate Comparison of Optimizers

Table 3 summarizes the final test-set metrics for the Firefighter Optimization Algorithm (FFO) and Particle Swarm Optimization (PSO) under their best hyperparameter configurations. Both optimizers achieved strong overall performance, with test accuracies ranging between 93.9% and 98.6% and macro F1-scores from 0.942 to 0.985. The highest single-run performance was obtained by FFO_s123, which reached an

accuracy of 98.58% and a macro F1-score of 0.985, followed by PSO_s123 with 95.33% accuracy and a macro F1-score of 0.954. Across seeds, PSO demonstrated slightly more consistent outcomes (mean accuracy of 94.6% and F1-score of 0.948), while FFO exhibited greater variability (range of approximately 3.9 percentage points in accuracy across seeds) but also a higher peak.

Table 3: Summary of test-set performance for each optimizer with best hyperparameters.

Algorithm	**Accuracy (%)**	**Precision**	**Recall**	**F1-score**
PSO_s123	95.33	0.957	0.952	0.954
PSO_s222	93.91	0.944	0.942	0.942
FFO_s123	**98.58**	**0.985**	**0.986**	**0.985**
FFO_s222	94.73	0.950	0.949	0.948

On average, the difference between their best runs was about 3.2 percentage points in accuracy and 0.031 in macro F1. Precision and recall values for both algorithms were closely aligned, differing by less than 0.01 on average, indicating balanced classification and limited bias toward any particular tumor class. FFO tended to yield marginally higher recall, suggesting slightly greater sensitivity to ambiguous or minority samples, whereas PSO maintained steadier precision and calibration between runs. Overall, both optimizers produced reliable, high-accuracy CNN configurations within the same computational budget. The results indicate that FFO achieved the strongest individual configuration, while PSO provided more stable and reproducible convergence across seeds, reflecting their respective exploration–exploitation dynamics within the hyperparameter search process.

4 Conclusion and Future Work

This thesis evaluated the effectiveness of **metaheuristic algorithms** for hyperparameter tuning of a **VGG16-based CNN** for brain tumor MRI classification. We compared the recent **Firefighter Optimization Algorithm (FFO)** with the well-established **Particle Swarm Optimization (PSO)** under identical datasets, seeds, and evaluation budgets.

The experiments in Chapter 4 demonstrated that both optimizers successfully discovered hyperparameter configurations that generalized to unseen test data. **FFO** consistently reached high accuracy earlier and achieved the highest single-run peak of **98.6%** test accuracy (seed 123, macro F1-score 0.985), while **PSO** improved more gradually but delivered smoother convergence and more reliable calibration, with test accuracy around **95%** (macro F1-score 0.954). These complementary behaviours highlight the trade-off between *sample-efficiency and peak performance* (FFO) versus *stability and calibration* (PSO). Overall, the results confirm that metaheuristic search is a practical and effective alternative to manual or grid-based tuning in medical image analysis.

Several contributions follow from this work. It represents the first adaptation of FFO for deep learning, achieved by reparameterizing the algorithm for continuous hyperparameters and embedding it in a reproducible TensorFlow/Keras pipeline. The comparative evaluation against PSO under identical conditions clarified how exploration–exploitation trade-offs manifest in hyperparameter optimization. Furthermore, the automated pipeline developed here ensures transparency and repeatability, supporting broader use in research. Behavioural insights emerged as well: FFO achieved the highest single-run discrimination through early localized refinement, whereas PSO produced smoother convergence and the strongest overall calibration, making it especially useful in risk-sensitive decision-support contexts.

At the same time, the work has limitations. The Kaggle dataset used has limited clinical heterogeneity, reducing the generalizability of results. The evaluation budget was relatively small (about 100 evaluations per optimizer), and only two random seeds per method were tested, restricting statistical robustness. Results were also limited to VGG16 as the backbone, meaning sensitivity to architecture choice was not examined.

Future work can address these limitations by scaling up search budgets to assess stability across seeds, benchmarking against a broader set of metaheuristics such as Grey Wolf, Differential Evolution, or CMA-ES, and exploring adaptive variants of FFO with dynamic parameter schedules to reduce overconfidence. Multi-objective optimization, explicitly balancing discrimination, calibration, and computational cost, represents another promising direction. Extending this framework to different backbones (ResNet, DenseNet, EfficientNet) and to related tasks such as segmentation or multimodal MRI analysis would test portability and robustness. Finally, integration with lightweight

AutoML methods may bridge neural architecture search and metaheuristic HPO under fixed resource constraints.

In conclusion, this thesis contributes both a reproducible framework for metaheuristic hyperparameter optimization and the first adaptation of FFO for deep learning. The complementary strengths of FFO and PSO suggest that optimizer choice should be guided by context: FFO is best when rapid gains or peak accuracy are needed under tight budgets, while PSO is preferable when stability and well-calibrated probabilities are paramount. Together, these insights support the development of adaptive, interpretable, and resource-efficient optimization strategies for healthcare AI.

References

[1] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.

[2] C.M. Bishop. *Pattern recognition and machine learning*, volume 4. Springer New York, 2006.

[3] Smriti Saini. Supervised vs. unsupervised learning: What's the difference?, 2023. Accessed: October 4, 2025.

[4] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[5] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A.W.M. van der Laak, Bram van Ginneken, and Clara I. Sánchez. A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42:60–88, December 2017.

[6] Eric J. Topol. High-performance medicine: the convergence of human and artificial intelligence. *Nature Medicine*, 25(1):44–56, 2019.

[7] Andre Esteva, Brett Kuprel, Roberto Novoa, Justin Ko, Susan Swetter, Helen Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542, 01 2017.

[8] Shoaib Siddiqui, Ahmad Salman, Imran Malik, Faisal Shafait, Ajmal Mian, Mark Shortis, and Euan Harvey. Automatic fish species classification in underwater videos: Exploiting pretrained deep neural network models to compensate for limited labelled data. *ICES Journal of Marine Science*, 75, 05 2017.

[9] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures, 2012.

[10] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[11] Md. Abu Bakr Siddique, Shadman Sakib, Mohammad Mahmudur Rahman Khan, Abyaz Kader Tanzeem, Madiha Chowdhury, and Nowrin Yasmin. Deep convolutional neural networks model-based brain tumor detection in brain mri images. In *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 909–914, 2020.

[12] Nicola Dinsdale, Emma Bluemke, Vaanathi Sundaresan, Mark Jenkinson, Stephen Smith, and Ana Namburete. Challenges for machine learning in clinical translation of big data imaging studies, 07 2021.

[13] Nima Tajbakhsh, Jae Y. Shin, Suryakanth R. Gurudu, R. Todd Hurst, Christopher B. Kendall, Michael B. Gotway, and Jianming Liang. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE Transactions on Medical Imaging*, 35(5):1299–1312, 2016.

[14] Sarthak Raghuvanshi and Sumit Dhariwal. The vgg16 method is a powerful tool for detecting brain tumors using deep learning techniques. page 46, 12 2023.

[15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[16] Jayesh Chauhan. The architecture and implementation of vgg-16. https://pub.towardsai.net/the-architecture-and-implementation-of-vgg-16-b050e5a5920b, 2023. Accessed: 2025-10-07.

[17] San Diego Brain Injury Foundation. What's the difference between all the different head scans?, 2024. Accessed: October 4, 2025.

[18] Khaled Alnowaiser, Abeer Saber, Esraa Hassan, and Wael A. Awad. An optimized model based on adaptive convolutional neural network and grey wolf algorithm for breast cancer diagnosis. *PLOS ONE*, 19(8):1–16, 08 2024.

[19] Siyu Xiong, Guoqing Wu, Xitian Fan, Xuan Feng, Zhongcheng Huang, Wei Cao, Xuegong Zhou, Shijin Ding, Jinhua Yu, Lingli Wang, and Zhifeng Shi. Mri-based brain tumor segmentation using fpga-accelerated neural network. *BMC Bioinformatics*, 22, 09 2021.

[20] Sérgio Pereira, Adriano Pinto, Victor Alves, and Carlos A. Silva. Brain tumor segmentation using convolutional neural networks in mri images. *IEEE Transactions on Medical Imaging*, 35(5):1240–1251, 2016.

[21] Mohammad Havaei, Axel Davy, David Warde-Farley, Antoine Biard, Aaron Courville, Yoshua Bengio, Chris Pal, Pierre-Marc Jodoin, and Hugo Larochelle. Brain tumor segmentation with deep neural networks. *Medical Image Analysis*, 35:18–31, 2017.

[22] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(null):281–305, February 2012.

[23] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms, 2012.

[24] Gyuwon Kim, Soo-Young Lee, Jong-Seok Oh, and Seungchul Lee. Deep learning-based estimation of the unknown road profile and state variables for the vehicle suspension system. *IEEE Access*, PP:1–1, 01 2021.

[25] Aiman Lameesa, Mahfara Hoque, Md Sakib Bin Alam, Shams Forruque Ahmed, and Amir H Gandomi. Role of metaheuristic algorithms in healthcare: a comprehensive investigation across clinical diagnosis, medical imaging, operations management, and public health. *Journal of Computational Design and Engineering*, 11(3):223–247, 05 2024.

[26] Sofia El Amoury, Youssef Smili, and Youssef Fakhri. Design of an optimal convolutional neural network architecture for mri brain tumor classification by exploiting particle swarm optimization. *Journal of Imaging*, 11(2), 2025.

[27] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. Grey wolf optimizer. *Advances in Engineering Software*, 69:46–61, 2014.

[28] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995.

[29] Mohammadreza Mashayekhi, Mojtaba Harati, and Homayoon Estekanchi. Development of an alternative pso-based algorithm for simulation of endurance time excitation functions. *Engineering Reports*, 1, 10 2019.

[30] David Zuehlke, Taylor Yow, Daniel Posada, Joseph Nicolich, Christopher Hays, Aryslan Malik, and Troy Henderson. Initial orbit determination for the cr3bp using particle swarm optimization, 07 2022.

[31] Shoffan Saifullah and Rafał Drężewski. Pso-unet: Particle swarm-optimized u-net framework for precise multimodal brain tumor segmentation, 2025.

[32] M. Z. Naser and A. Z. Naser. The firefighter algorithm: A hybrid metaheuristic for optimization problems, 2024.

[33] Sartaj Bhuvaji, Ankita Kadam, Prajakta Bhumkar, Sameer Dedge, and Swati Kanchan. Brain tumor classification (mri), 2025.

[34] Bjoern Menze, András Jakab, Stefan Bauer, Jayashree Kalpathy-Cramer, Keyvan Farahaniy, Justin Kirby, Yuliya Burren, Nicole Porz, Johannes Slotboomy, Roland Wiest, Levente Lancziy, Elizabeth Gerstnery, Marc-Andr´e Webery, Tal Arbel, Brian Avants, Nicholas Ayache, Patricia Buendia, Louis Collins, Nicolas Cordier, and Koen Van Leemput. The multimodal brain tumor image segmentation benchmark (brats). *IEEE Transactions on Medical Imaging*, 99, 12 2014.

[35] Spyridon Bakas, Mauricio Reyes, Andras Jakab, Stefan Bauer, Markus Rempfler, Alessandro Crimi, Russell Takeshi Shinohara, Christoph Berger, Sung Min Ha, and Martin Rozyck. Identifying the best machine learning algorithms for brain

tumor segmentation, progression assessment, and overall survival prediction in the brats challenge, 2019.

[36] El-Ghazali Talbi. *Metaheuristics: From Design to Implementation*, volume 74. John Wiley & Sons, 2009.

[37] Xin-She Yang. *Nature-Inspired Optimization Algorithms: Second Edition*. 09 2020.

[38] Ibrahim Aljarah, Hossam Faris, and Seyedali Mirjalili. Optimizing connection weights in neural networks using the whale optimization algorithm. *Soft Computing*, 22, 01 2018.