Evaluating Hyperparameter Optimization Techniques for Image Classification in Astroinformatics

Andrew Zabelo | Caltech Ay 119

1) Introduction

Astroinformatics integrates astronomy with data science and machine learning to address the growing complexity and scale of modern astronomical data. As telescopes and simulations generate increasingly large and intricate datasets, researchers turn to machine learning models to automate the extraction of scientific insights. Convolutional Neural Networks are particularly well-suited for image-based classification tasks, playing a central role in analyzing astrophysical images. However, maximizing their effectiveness depends on the careful selection of hyperparameters, a process that poses significant computational and methodological challenges. This project investigates how different optimization strategies influence model performance, with the goal of identifying efficient approaches to tuning in a data-rich astronomical setting. All of our code is made available at https://github.com/azabelo/ay119 .

2) Background

Astroinformatics has emerged as a vital interdisciplinary field at the intersection of astronomy, data science, and machine learning, driven by the rapid growth of data from large-scale sky surveys, space missions, and high-resolution simulations. As astronomical datasets reach petabyte scales, from instruments like the Vera C. Rubin Observatory, Gaia, the James Webb Space Telescope, or the Zwicky Transient Facility, traditional analytic methods alone are no longer sufficient to manage, process, or extract meaningful insights. Machine learning classification models, in particular, have become indispensable tools in this landscape, enabling automated identification of galaxies, variable stars, exoplanets, and transient phenomena across complex, high-dimensional feature spaces. These models not only accelerate discovery by reducing manual effort but also help uncover subtle patterns and correlations that might elude conventional techniques. As a result, astroinformatics is playing a transformative role in modern astrophysics, supporting both real-time analysis and deeper theoretical understanding through data-driven approaches.

Supervised learning for image classification involves training a model to recognize patterns in labeled image data, where each image is associated with a known class or category. Convolutional Neural Networks (CNNs) are the most widely used architecture for this task due to their ability to efficiently capture spatial hierarchies in visual data. A CNN typically consists of a series of convolutional layers that apply learnable filters to local regions of the input image, extracting features such as edges, textures, and shapes. These are followed by pooling layers that reduce spatial dimensions and preserve the most salient information, and finally by fully connected layers that interpret the extracted features to produce a probability distribution over class labels. During training, the model adjusts its weights to minimize a loss function, such as cross-entropy, by comparing predicted labels to true labels using backpropagation and gradient descent. Once trained, the CNN can classify new, unseen images by applying the same learned filters, making it a powerful tool for astronomical applications such as galaxy morphology classification, star–galaxy separation, and transient event detection.

Selecting optimal hyperparameters for machine learning models, especially deep neural networks like CNNs, is a challenging and often time-consuming process. Hyperparameters such as learning rate, batch size, number of layers, and regularization strength can significantly affect model performance, yet there is no universally optimal setting that works across all datasets or tasks. The search space is often high-dimensional and non-convex, making exhaustive grid search computationally expensive and inefficient. Moreover, interactions between hyperparameters can lead to unpredictable behavior, where small changes in one parameter may drastically impact performance only in combination with others. As a result, effective hyperparameter optimization requires careful strategy, often involving techniques like random search, Bayesian optimization, or gradient-based tuning to efficiently explore the space and improve generalization on unseen data.

The purpose of this project is to explore the effectiveness of various hyperparameter optimization strategies in the context of an astrophysical image classification task. By restricting our focus to a relatively simple family of deep neural network models, we aim to isolate the impact of the optimization strategy itself rather than confounding results with complex architectural variations. This controlled approach allows us to assess how different tuning methods, such as grid search, random search, Bayesian optimization, and others, perform in practice when applied to real astrophysical data. Just as machine learning classification models accelerate discovery by automating tedious and repetitive classification tasks for astrophysicists, hyperparameter optimization algorithms streamline the model development process for ML practitioners and domain scientists alike, reducing the manual effort required to design and fine-tune effective models.

3) Hyperparameter Optimization Techniques

3.1) Grid Search

Hyperparameter optimization is a crucial step in machine learning model development, aiming to identify the combination of parameters that yields the best performance on validation data. Several strategies exist to tackle this problem, each with trade-offs in efficiency, interpretability, and performance. At one end of the spectrum is grid search, a brute-force method that systematically evaluates all possible combinations of hyperparameter values from a predefined set. While simple to implement and easy to parallelize, grid search becomes computationally prohibitive as the number of parameters increases, especially if some parameters turn out to be irrelevant or interact nonlinearly. Despite its limitations, grid search remains a popular baseline, especially when the search space is small or when the model is relatively inexpensive to train.

3.2) Random Search

Random search improves on grid search by sampling hyperparameter combinations randomly from specified distributions rather than evaluating all fixed combinations. This approach has been shown empirically to be more efficient, particularly in high-dimensional spaces, because it explores more configurations and avoids wasting resources on irrelevant parameters. For example, if only a few hyperparameters strongly influence performance, random search is more likely to discover a good setting in fewer iterations than grid search. It is also simple to implement and can be extended to continuous or mixed discrete-continuous parameter spaces, making it well-suited to early experiments or baseline tuning.

3.3) Ablation-Style Search

Another intuitive strategy is ablation-style search, which involves systematically varying one hyperparameter at a time while keeping others fixed to assess its individual effect on performance. This technique is often used to understand model sensitivity or to justify design choices in research papers. While it does not guarantee finding the best configuration in the limit of infinite trials, ablation style search is explicitly designed to actively find improvements in contrast to random or grid search. Ablation studies are also valuable for interpretability, helping researchers and practitioners identify which components contribute meaningfully to performance and which can be simplified or removed.

3.4) Genetic Algorithm Optimization

More sophisticated strategies include genetic algorithms, which are inspired by the principles of natural selection and evolutionary biology. In this framework, each set of hyperparameters is treated as an individual in a population, encoded as a “genome.” The population evolves over generations through processes such as selection, crossover, and mutation. Individuals with better fitness, typically measured by validation accuracy, are more likely to pass their traits to the next generation. Genetic algorithms are well-suited for exploring large and complex search spaces, including discrete or non-differentiable hyperparameters, but they can be slow to converge and sensitive to tuning of their own meta-parameters (e.g., mutation rate, population size). Nevertheless, they offer a flexible and parallelizable approach, especially useful when gradient-based methods are infeasible.

3.5) Bayesian Optimization

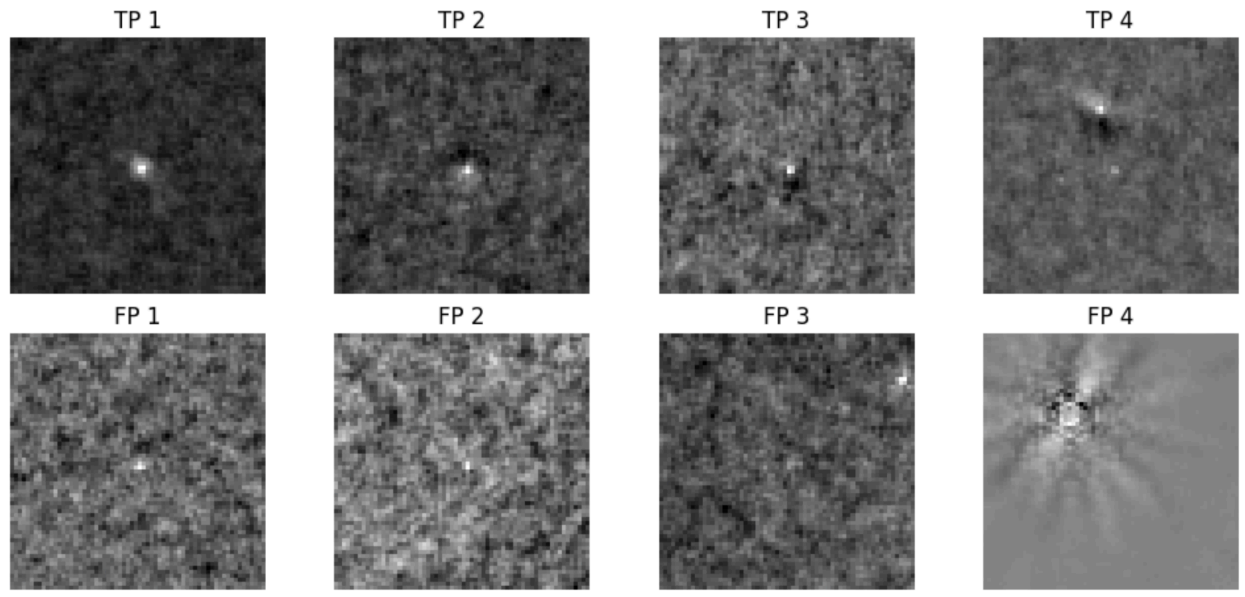
The most principled modern method is Bayesian optimization, which models the performance of hyperparameter configurations as a probabilistic function and selects new points to evaluate in a way that balances exploration and exploitation. At the core of Bayesian optimization is a surrogate model, commonly a Gaussian Process (GP), which estimates the objective function (e.g., validation loss) and provides both a mean prediction and uncertainty estimate for any point in the hyperparameter space. Using this model, an acquisition function, such as Expected

Improvement (EI), Probability of Improvement (PI), or Lower Confidence Bound (LCB), is optimized to determine the next best point to evaluate. This function encodes a trade-off between sampling uncertain regions (exploration) and sampling regions expected to improve the current best result (exploitation). Bayesian optimization is sample-efficient and particularly effective when model training is expensive, making it ideal for deep learning or astrophysical inference tasks where each function evaluation (i.e., model training) may take hours or days. However, the method becomes computationally challenging in very high-dimensional spaces and relies on assumptions such as smoothness of the objective function, which may not always hold.

4) Implementation Details

4.1) Dataset

For this project, we utilize an astrophysical image dataset obtained from Ashish Mahabal's *ExRomana* GitHub repository. The dataset consists of a training split containing 3,273 grayscale images, each with dimensions of 64×64 pixels. These images are labeled into two classes: True Positive and False Positive. Although detailed documentation about the dataset is limited, the class labels suggest that the data may originate from a post-processing step of a prior model, perhaps for transient detection or object classification, with the aim of learning to distinguish between genuine detections and erroneous ones. Notably, the dataset is imbalanced, with approximately a 2:1 ratio between the majority (True Positive) and minority (False Positive) classes, which poses additional challenges for supervised learning and may necessitate careful handling of class weights or sampling strategies. Below, we present a few example images from the dataset to illustrate the variation and structure of the data.



4.2) Classification Model:

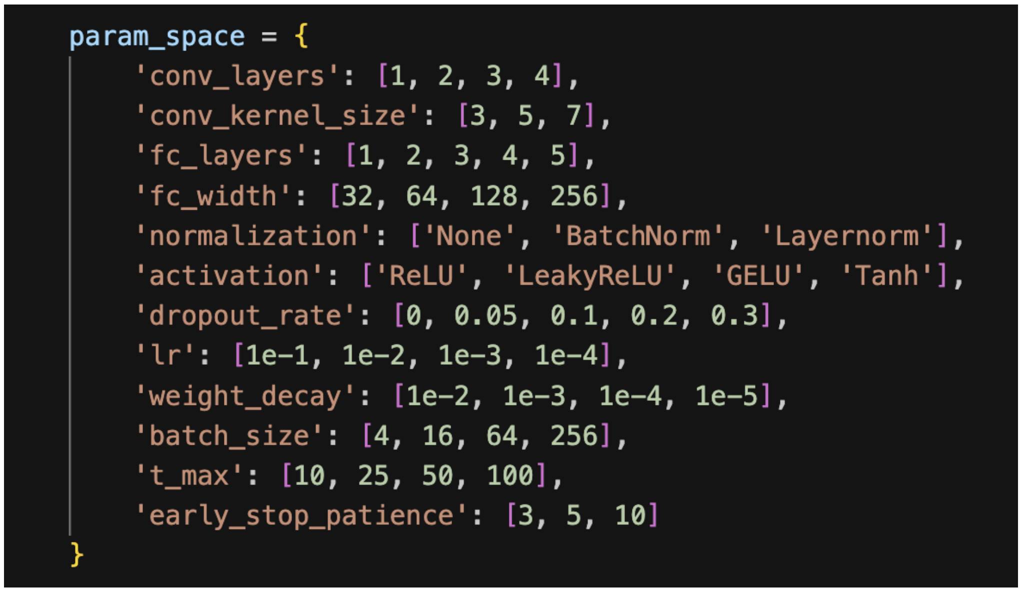
To facilitate a fair and consistent comparison between different hyperparameter optimization strategies, we adopt a fully automated and modular deep learning architecture, which is constructed dynamically based on a set of predefined hyperparameters. This design allows us to explore a wide range of model configurations while maintaining control over the complexity of the learning problem. The neural network architecture is implemented in PyTorch and comprises two main components: a convolutional feature extraction module and a fully connected classification head. Together, these modules transform 64×64 grayscale astrophysical images into scalar logits suitable for binary classification between True Positive and False Positive examples.

The convolutional component, controlled by the conv_layers hyperparameter, consists of a variable number of convolutional blocks. Each block applies a 2D convolution with kernel size defined by conv_kernel_size (chosen from {3, 5, 7}), followed by an activation function and a 2×2 max-pooling layer that reduces the spatial dimensions. The number of output channels begins at 16 and doubles after each convolutional layer, increasing the expressive power of deeper networks. This structure is well-motivated in image processing: convolutional layers exploit spatial locality and weight sharing, enabling the model to learn position-invariant features like edges, corners, and textures with far fewer parameters than a fully connected architecture would require. Pooling layers add translational robustness and progressively reduce the computational burden by decreasing resolution.

After the final convolutional output is flattened, it is passed through a series of fully connected (FC) layers, whose depth and width are governed by the hyperparameters fc_layers (1–5 layers) and fc_width (32–256 units per layer), respectively. Each intermediate FC layer includes a linear transformation, a normalization layer selected via normalization (None, BatchNorm, or LayerNorm), an activation function chosen from activation (ReLU, LeakyReLU, GELU, or Tanh), and a dropout layer with dropout_rate (ranging from 0 to 0.3). Normalization layers help stabilize training by reducing internal covariate shift, allowing for higher learning rates and improved convergence. Activation functions introduce nonlinearity, enabling the model to approximate complex functions; different choices affect the smoothness and saturation behavior of the network's response. Dropout serves as a regularizer that stochastically removes connections during training, reducing co-adaptation and helping the model generalize better on imbalanced and noisy datasets.

The final FC layer is a single-unit linear layer with no activation, producing a raw logit. During training, we use the binary cross-entropy loss (BCEWithLogitsLoss) and optimize the model using the AdamW optimizer, because AdamW handles weight decay in a more principled way than Adam. The hyperparameters controlling optimization include lr (learning rate), weight_decay (for L2 regularization), and batch_size, which together influence the speed, stability, and generalization of training. We also employ a cosine annealing learning rate schedule, with the decay horizon controlled by t_max. This schedule allows the learning rate to decrease gradually and cyclically, improving final performance by fine-tuning the model weights in later epochs. To avoid overfitting and reduce computation, we implement early stopping, controlled by the early_stop_patience hyperparameter, which halts training if validation loss does not improve for a set number of epochs.

All of these components are parameterized by the following hyperparameter space:



```
param_space = {
    'conv_layers': [1, 2, 3, 4],
    'conv_kernel_size': [3, 5, 7],
    'fc_layers': [1, 2, 3, 4, 5],
    'fc_width': [32, 64, 128, 256],
    'normalization': ['None', 'BatchNorm', 'Layernorm'],
    'activation': ['ReLU', 'LeakyReLU', 'GELU', 'Tanh'],
    'dropout_rate': [0, 0.05, 0.1, 0.2, 0.3],
    'lr': [1e-1, 1e-2, 1e-3, 1e-4],
    'weight_decay': [1e-2, 1e-3, 1e-4, 1e-5],
    'batch_size': [4, 16, 64, 256],
    't_max': [10, 25, 50, 100],
    'early_stop_patience': [3, 5, 10]
}
```

4.3) Hyperparameter Optimization Experimental Environment

To rigorously compare different hyperparameter optimization strategies, we implement a unified experimental framework that evaluates each algorithm's ability to discover performant models within a large and complex hyperparameter space. Each hyperparameter in our space is carefully chosen for its relevance to model behavior, ranging from architectural choices (such as the number of layers or activation functions) to optimization settings (like learning rate and batch size) and regularization techniques (including dropout and weight decay). Because we are tackling a new classification task in astrophysics where strong priors on model design are not available, this comprehensive space is intentionally kept broad. However, even though we restrict ourselves to a modest 3–5 discrete values per hyperparameter, the total number of combinations exceeds 10 million, making exhaustive grid search infeasible for practical purposes.

Instead, we compare four diverse hyperparameter optimization strategies: **random search**, **ablation-style search**, **genetic algorithms**, and **Bayesian optimization**. Each algorithm explores the hyperparameter space in a distinct manner. Random search selects hyperparameter combinations uniformly at random, providing a simple yet surprisingly effective baseline. Ablation-style search varies one hyperparameter at a time while keeping others fixed to evaluate its marginal impact. Genetic algorithms simulate evolution through selection, crossover, and mutation, gradually evolving a population of models toward better performance. Bayesian optimization, on the other hand, constructs a Gaussian Process surrogate model and selects new hyperparameter configurations by maximizing an acquisition function that balances exploration and exploitation.

For the random, genetic, and bayesian hyperparameter optimization strategies, we perform 100 independent trials, where a new set of hyperparameters is selected, the corresponding model is initialized, and training is conducted on the same dataset. For the ablation style strategies, we iterate over all possible unilateral perturbations of the hyperparameters from the starting configuration, resulting in 47 runs. For each trial, we record a comprehensive set of metrics including training loss, training accuracy, validation loss, and validation accuracy at every epoch, as well as the final validation accuracy achieved and the total wall-clock time taken to train the model. This detailed logging allows us not only to compare the final outcomes but also to analyze convergence behavior, overfitting trends, and computational efficiency across strategies. By keeping the training and evaluation procedures consistent, we ensure that the performance differences we observe are attributable solely to the choice of hyperparameter optimization method.

5) Results

5.1) Improvement Time Series

Figure 1: Random Search Improvement Time Series (Validation Accuracy, Validation Loss, Running Best)

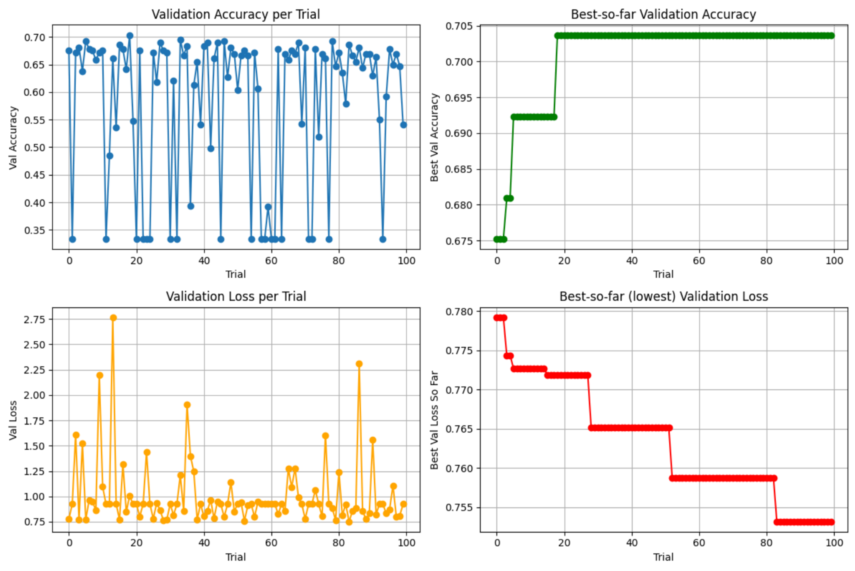


Figure 2: Ablation Search Improvement Time Series (Validation Accuracy, Validation Loss, Running Best)

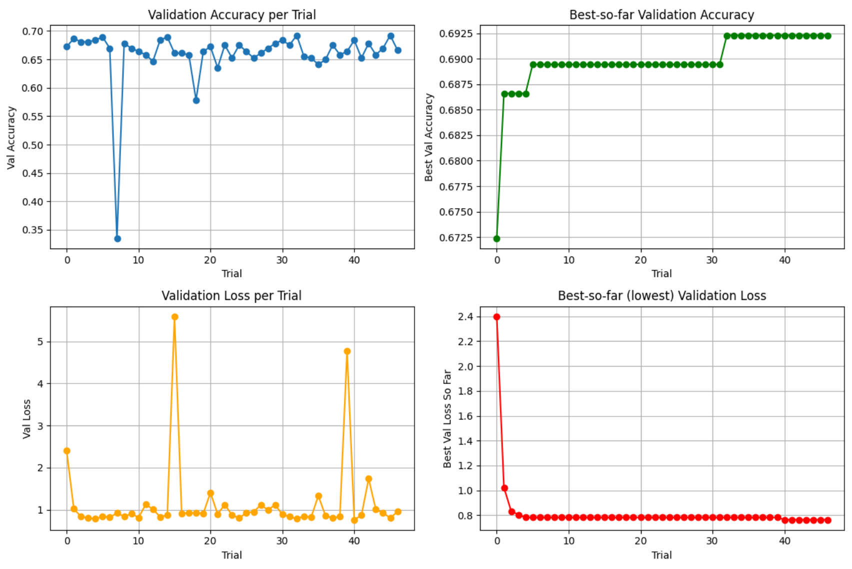


Figure 3: Genetic Search Improvement Time Series (Validation Accuracy, Validation Loss, Running Best)

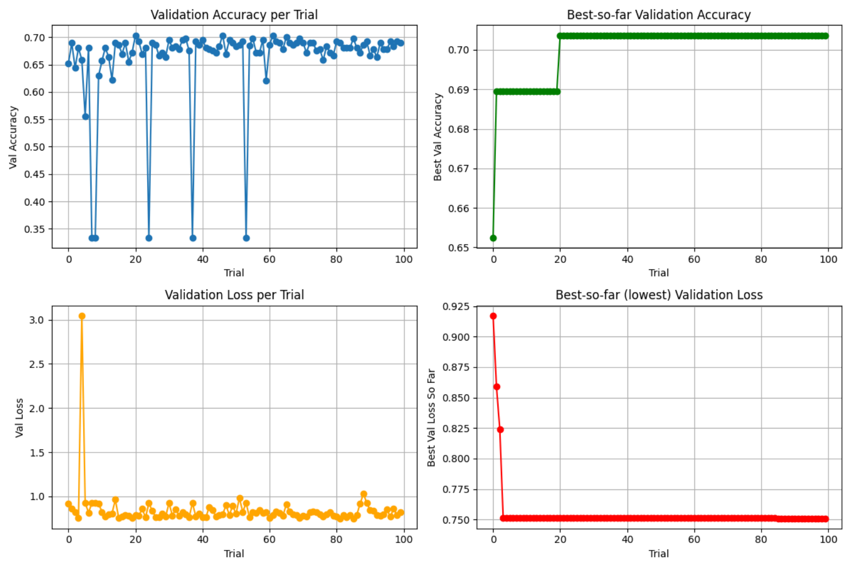
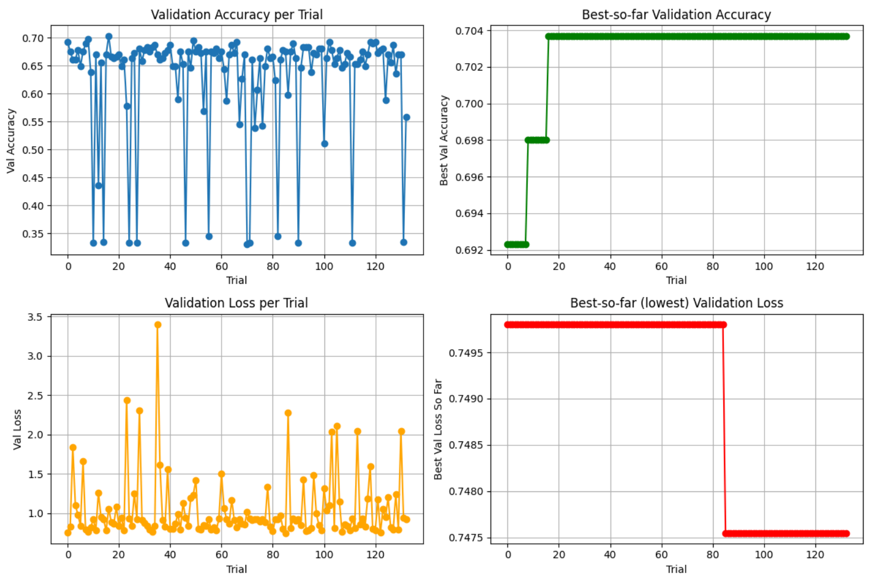


Figure 4: Bayesian Search Improvement Time Series (Validation Accuracy, Validation Loss, Running Best)



5.2) Evaluation Metric Distributions

Figure 5: Random Search Validation Loss and Accuracy Distributions over all trials

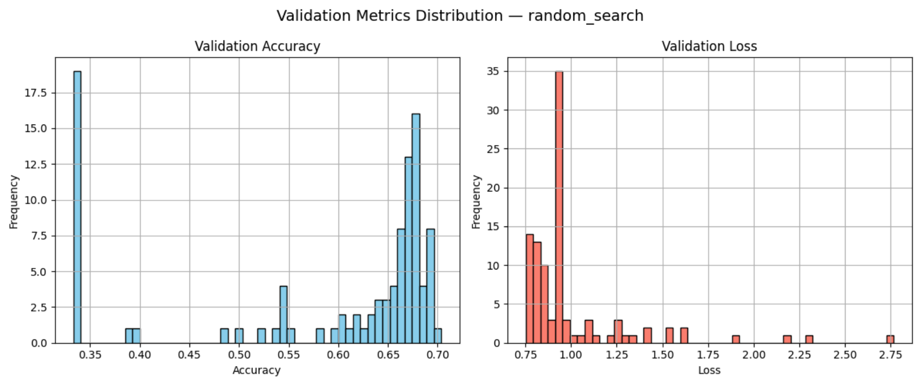


Figure 6: Ablation Search Validation Loss and Accuracy Distributions over all trials

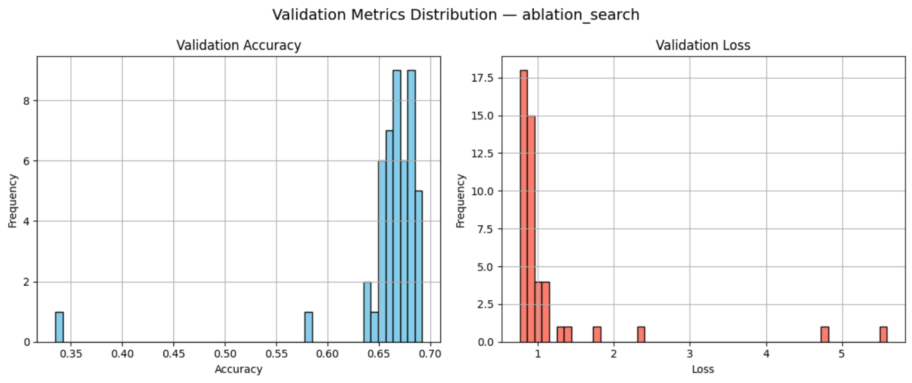


Figure 7: Genetic Search Validation Loss and Accuracy Distributions over all trials

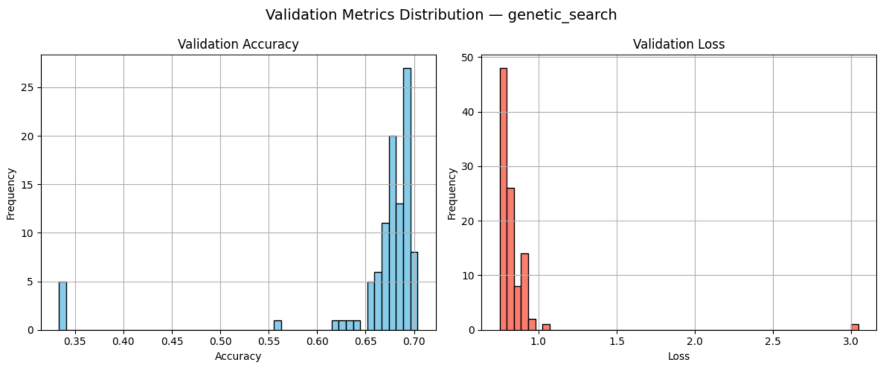
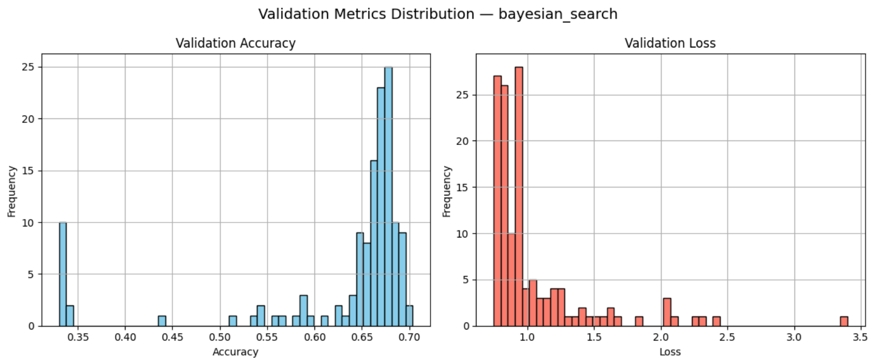


Figure 8: Bayesian Search Validation Loss and Accuracy Distributions over all trials



5.3) Time Series over all 100 trials

Figure 9: Random Search time series over all trials (train and val, loss and accuracy) (more blue indicates earlier run, more yellow indicates later run)

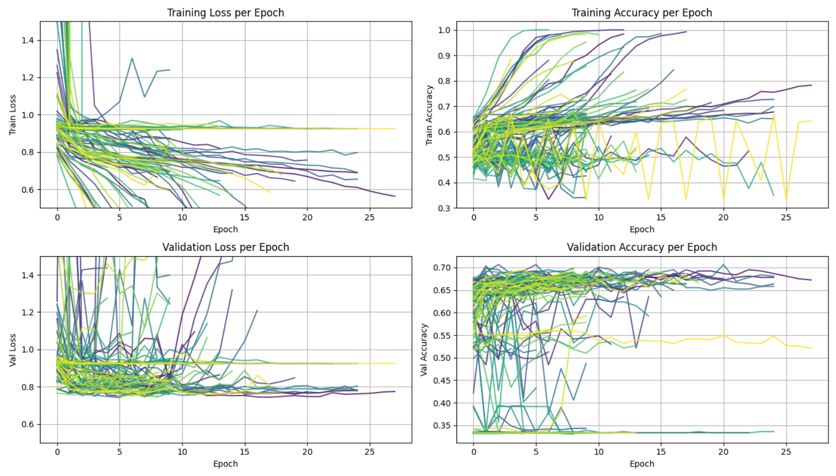


Figure 10: Ablation Search time series over all trials (train and val, loss and accuracy) (more blue indicates earlier run, more yellow indicates later run)

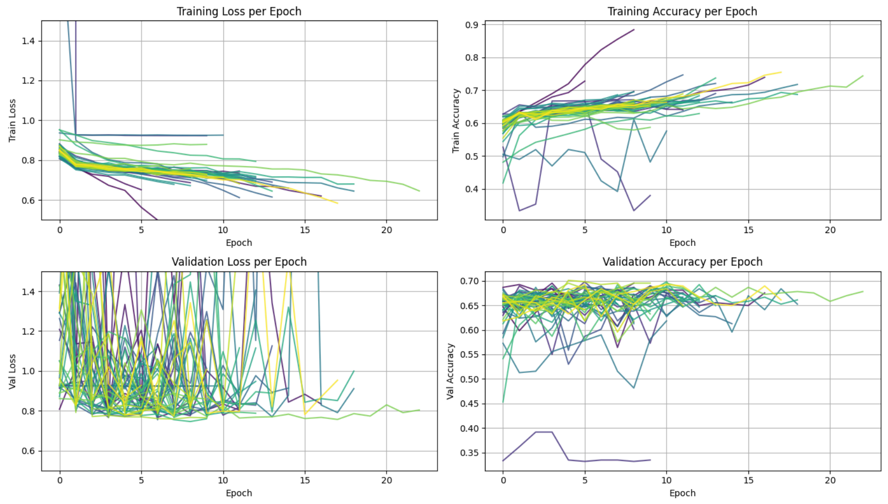


Figure 11: Genetic Search time series over all trials (train and val, loss and accuracy) (more blue indicates earlier run, more yellow indicates later run)

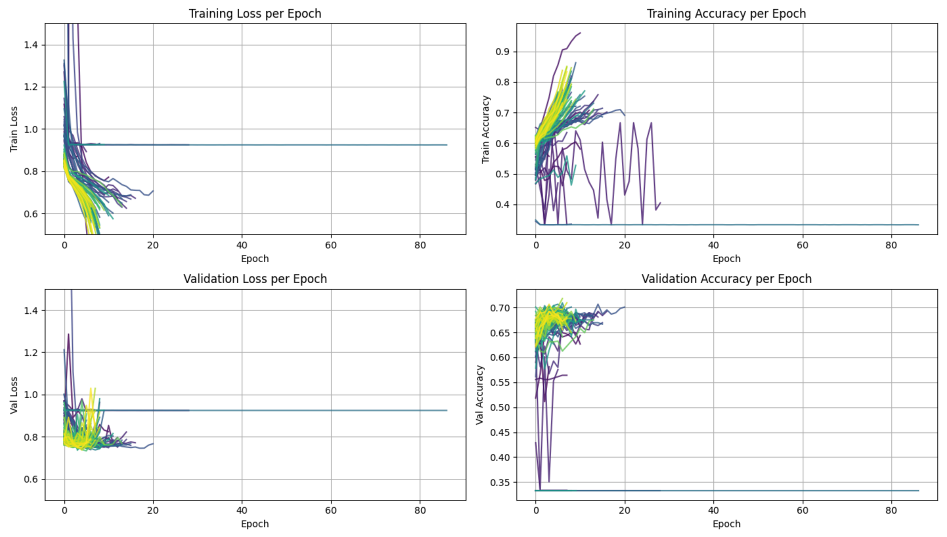
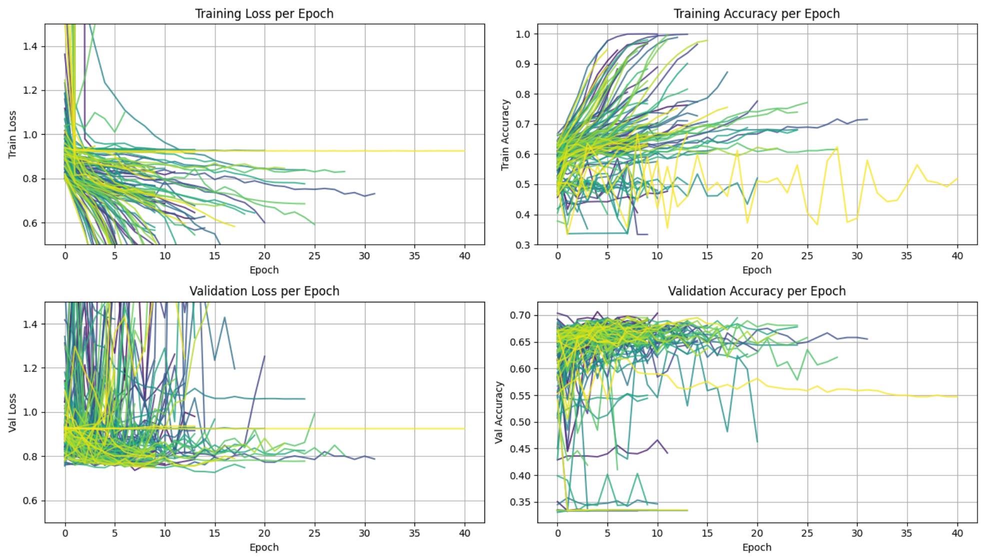


Figure 12: Bayesian Search time series over all trials (train and val, loss and accuracy) (more blue indicates earlier run, more yellow indicates later run)



5.4) Final Results Table (using best hyperparameter configuration found)

	Random	**Ablation**	**Genetic**	**Bayesian**
Final Train Loss	0.7208	0.7245	0.6819	0.7611
Final Val Loss	0.7709	0.7620	0.7679	0.7889
Final Test Loss	0.7428	0.7416	0.7358	0.7483
Final Train Accuracy	0.6331	0.6557	0.6993	0.6268

Final Val Accuracy	0.6638	0.6895	0.6866	0.6553
Final Test Accuracy	0.6809	0.7009	0.6980	0.6524

6) Conclusion

This project examined the efficacy of four distinct hyperparameter optimization strategies (random search, ablation-style search, genetic algorithms, and Bayesian optimization) for image classification in an astroinformatics setting. All four methods successfully identified reasonable hyperparameter configurations, demonstrating their ability to automate model tuning even in the absence of strong prior knowledge about the data or architecture. However, performance varied substantially when models selected by each strategy were evaluated on a held-out test set.

While ablation-style search, random search, and genetic algorithms each yielded strong test performance, Bayesian optimization notably underperformed. Despite achieving a sharp gain in validation accuracy during its search process, the corresponding model exhibited poor generalization, suggesting a form of overfitting or "p-hacking" induced by the optimization procedure. This was further supported by the time series analysis, which revealed that Bayesian optimization achieved its gains through a single large jump, rather than through incremental and robust improvements. In contrast, random search, despite its purely exploratory nature, demonstrated steady, albeit slow, performance gains across trials. Genetic and ablation-style search, both of which favor exploitation over exploration, showed rapid initial improvements followed by stagnation, yet still achieved strong final results, indicating that their conservative search behavior may have helped avoid overfitting.

The distribution of validation losses and accuracies across the 100 trials confirmed that most hyperparameter configurations led to reasonable training behavior, though a small number of outliers highlighted the occasional instability of model training. Interestingly, ablation-style search emerged as the best-performing strategy on the test set, despite conducting fewer trials than its counterparts. This finding suggests that, in high-dimensional yet structured search spaces, simple and interpretable optimization methods may outperform more complex strategies, especially when the dataset is noisy or limited in size.

Overall, these results highlight that advanced methods such as Bayesian optimization are not necessarily superior for all tasks, particularly in domains like astrophysical image classification where overfitting is a significant risk. In contrast, simpler methods like random search and ablation-style exploration not only remain competitive but may offer better generalization under certain conditions. Future work may extend this study by exploring continuous hyperparameter spaces, applying these optimization strategies to multi-class classification problems or regression tasks, and experimenting with more diverse or curated datasets. Additionally, given the limited performance gains observed from deep learning on this dataset, subsequent efforts might also consider simpler models with manually engineered features as a potentially more effective alternative.

7) References

Erden, C. Genetic algorithm-based hyperparameter optimization of deep learning models for PM2.5 time-series prediction. Int. J. Environ. Sci. Technol. 20, 2959–2982 (2023). https://doi.org/10.1007/s13762-023-04763-6

Frazier, P. I. (2018, July 10). A tutorial on Bayesian optimization. Retrieved from https://people.orie.cornell.edu/pfrazier/papers/bayesopt.pdf

Onorato, G. (2024). *Bayesian optimization for hyperparameters tuning in neural networks* (Undergraduate thesis, Sapienza University of Rome). Facoltà di Ingegneria dell'Informazione, Informatica e Statistica, Dipartimento di Ingegneria Informatica, Automatica e Gestionale.

O'Shea, K., & Nash, R. (n.d.). *An introduction to convolutional neural networks*. Department of Computer Science, Aberystwyth University; School of Computing and Communications, Lancaster University. Retrieved from https://arxiv.org/abs/1511.08458

Xiao, X., Basodi, S., Pan, Y., Ji, C., & Yan, M. (2020, June 25). *Efficient hyperparameter optimization in deep learning using a variable length genetic algorithm* [Preprint]. arXiv. https://arxiv.org/abs/2006.11659

White, C., Safari, M., Sukthanker, R., Ru, B., Elsken, T., Zela, A., Dey, D., & Hutter, F. (2023). *Neural architecture search: Insights from 1000 papers*. arXiv. https://arxiv.org/abs/2301.10910

Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research, 20*(55), 1–21. Retrieved from http://jmlr.org/papers/v20/18-598.html

Martinazzo, A., Espadoto, M., & Hirata, N. S. T. (n.d.). *Self-supervised learning for astronomical image classification*. Institute of Mathematics and Statistics, University of São Paulo. Retrieved from https://arxiv.org/abs/2302.01277

Martinazzo, A., Espadoto, M., & Hirata, N. S. T. (2023). *Self-supervised learning for astronomical image classification*. arXiv. https://arxiv.org/abs/2302.01277

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research, 13*, 281–305. Retrieved from http://www.jmlr.org/papers/v13/bergstra12a.html

Liashchynskyi, P., & Liashchynskyi, P. (2019). Grid search, random search, genetic algorithm: A big comparison for NAS. *arXiv*. https://arxiv.org/abs/1912.06059

Meyes, R., Lu, M., Waubert de Puiseau, C., & Meisen, T. (2019). *Ablation studies in artificial neural networks*. arXiv. https://arxiv.org/abs/1901.08644

Papastratis, I. (2021). *Ablation study of self-supervised learning for image classification*. arXiv. https://arxiv.org/abs/2106.03751