Automated Morphological Classification of Galaxies using Deep Convolutional Architecture

Nikolaos Mavros
Dept. of Electrical & Computer Engineering
University of Thessaly
Volos, Greece
nmavros@uth.gr

Kiveli Fotinaki
Dept. of Electrical & Computer Engineering
University of Thessaly
Volos, Greece
kfotinaki@uth.gr

***Abstract*—One of the most important tasks in observational cosmology is the morphological classification of galaxies. A thorough approach to automating this procedure with a specially created Convolutional Neural Network (CNN) is presented in this paper. In contrast to conventional "black box" methods, we used a five-phase experimental approach to separate and optimize scalability, hyperparameter tuning, and architectural choices separately. 37 probabilistic morphological features are predicted by the suggested model, which was trained on the Galaxy Zoo dataset. With a verified score of 0.109 on the blind test set, it exhibits strong generalization and attains a Validation RMSE of 0.106. In addition, we offer an in-depth analysis of the learning rate, a review of the computational scalability of the system (training time vs. data size), and a comparison with cutting-edge architectures such as ResNet, VGG19, and CViT.**

***Index Terms*—Galaxy Zoo, Deep Learning, CNN, Regression, Sensitivity Analysis, Scalability.**

I. Introduction

Classifying galaxies by their shape (e.g., Spiral, Elliptical, Irregular) provides vital clues about their formation history. The "Galaxy Zoo" project crowdsourced this task, resulting in a probability distribution of classifications for nearly 900,000 galaxies. Replicating this human consensus automatically is a regression problem, where the model must predict probabilities for 37 distinct sub-categories.

This work tackles the problem by creating a CNN from the ground up that is both powerful and lightweight. Our contribution is twofold: (1) A transparent, modular experimental approach that thoroughly verifies every pipeline component; and (2) An extremely effective architecture that produces competitive outcomes without depending on large amounts of pre-trained models.

II. Experimental Methodology

To ensure rigorous evaluation, we structured the experiment into four distinct phases, corresponding to the modular blocks implemented in the codebase.

A. Phase 1: Architecture Implementation & Validation

The primary objective of this phase was to establish a baseline performance and verify that the custom model could learn effectively without immediate overfitting.

- **Architecture Design:** We implemented a custom CNN composed of 4 Convolutional Blocks with increasing filter depth ($32 \rightarrow 64 \rightarrow 128 \rightarrow 256$), specifically optimized for the 64×64 pixel input resolution.
- **Data Split:** The labeled dataset of 61,578 images was partitioned into a **80% Training set** and a **20% Validation set**.
- **Objective:** To calculate the baseline Root Mean Squared Error (RMSE) and ensure the loss curves for training and validation remained synchronized.

B. Phase 2: Sensitivity Analysis

Before full-scale training, we conducted "Micro-Experiments" to optimize the hyperparameters.

1) Learning Rate (LR) Optimization: To identify the convergence "Sweet Spot," we trained the model for short **3-epoch bursts** using a logarithmic range of learning rates $(10^{-2}, 10^{-3}, 10^{-4})$. This allowed us to quickly identify the rate that maximizes speed without causing instability.

2) Dropout Rate Tuning: We analyzed the regularization strength by varying the Dropout Rate in the dense layers.

- **Values Tested:** $0.2, 0.5, 0.8$.
- **Method:** Fast-track training (3 epochs) to observe immediate fitting trends.
- **Goal:** To balance the trade-off between underfitting (high dropout) and signal retention.

3) Inference Latency Analysis: To verify the system's efficiency for real-time applications, we measured the specific time required to process a single image. The protocol involved:

1) Isolating a single test image.
2) Performing a GPU "warm-up" run to load CUDA kernels.
3) Averaging the prediction time over **100 iterations** to obtain a statistically significant result in milliseconds per image (ms/img).

C. Phase 3: Scalability Stress-Test

This experiment quantified the computational cost of the custom architecture as a function of dataset growth.

- **Methodology:** We measured the training time per epoch on incremental subsets of the data $(20\%, 40\%, 60\%, 80\%, 100\%)$.

- **Objective:** To demonstrate that the architecture scales linearly (or sub-linearly) and remains efficient enough for training on standard academic hardware.

D. Phase 4: Final Deployment (Production)

Having validated the architecture and hyperparameters, we proceeded to the production phase.

- **Full Training:** The validation split was removed, and the model was retrained on **100% of the available images** to maximize the information density in the weights.
- **Blind Inference:** The frozen network was deployed to predict probabilities for the 79,975 unlabelled test images.
- **Output:** The system generated the `galaxy_zoo_submission.csv` file for official external evaluation on the Kaggle Leaderboard.

III. Neural Network Architecture

We designed a custom Convolutional Neural Network (CNN) specifically tailored for the 64×64 input resolution of the Galaxy Zoo dataset. The architecture adopts the **VGG (Visual Geometry Group)** design philosophy. Instead of using large receptive fields (e.g., 7×7), we stack small 3×3 filters. This allows the network to learn deeper, more non-linear representations with fewer parameters (2.6 million) than traditional shallow networks.

The complete architectural pipeline is illustrated in Fig. 1, and the precise layer specifications are detailed in Table I.

TABLE I
Detailed Model Configuration & Parameter Count

Layer Type	Output Shape	Param #
Conv2D	(64, 64, 32)	896
Batch Normalization	(64, 64, 32)	128
MaxPooling2D	(32, 32, 32)	0
Conv2D	(32, 32, 64)	18,496
Batch Normalization	(32, 32, 64)	256
MaxPooling2D	(16, 16, 64)	0
Conv2D	(16, 16, 128)	73,856
Batch Normalization	(16, 16, 128)	512
MaxPooling2D	(8, 8, 128)	0
Conv2D	(8, 8, 256)	295,168
Batch Normalization	(8, 8, 256)	1,024
MaxPooling2D	(4, 4, 256)	0
Flatten	(4096)	0
Dense (ReLU)	(512)	2,097,664
Dropout (0.2)	(512)	0
Dense (ReLU)	(256)	131,328
Dropout (0.25)	(256)	0
Output Dense (Sigmoid)	**(37)**	**9,509**
Total Parameters		**2,628,837**

A. The Encoder Pathway

The feature extraction acts as a hierarchical encoder. It consists of four convolutional blocks (blue blocks in Fig. 1) where the number of filters doubles at each stage ($32 \to 64 \to 128 \to 256$). This progression allows the model to capture increasingly abstract features:

- **Block 1 (32 Filters):** Detects low-level geometric primitives (edges, contrast gradients).
- **Block 2 (64 Filters):** Assembles primitives into simple local shapes (arcs, circular blobs).
- **Block 3 (128 Filters):** Identifies complex galactic structures (spiral arms, central bars).
- **Block 4 (256 Filters):** Aggregates high-level semantic information into a global representation.

Within each block, we apply a strict sequence of operations to optimize learning dynamics:

- **Conv2D (3×3, Padding='Same'):** Performs feature extraction while preserving spatial dimensions.
- **Batch Normalization:** Applied immediately after convolution. It re-centers the layer inputs to a mean of 0 and variance of 1. Analytically, this mitigates *internal covariate shift*, allowing for higher learning rates and stabilizing the gradient flow.
- **ReLU Activation:** The function $f(x) = \max(0, x)$ introduces non-linearity. Unlike Tanh or Sigmoid, ReLU does not saturate for positive values, effectively preventing the *vanishing gradient problem* in deep architectures (see Fig. 2a).
- **Max Pooling (2×2):** Reduces the spatial dimensions by 75%. This decreases computational complexity and provides *translation invariance*, ensuring the model recognizes a galaxy regardless of its specific position in the frame.

B. The Regression Head

Following feature extraction, the maps are flattened into a 1D vector and passed to the Dense network (orange blocks in Fig. 1). This section functions as a probabilistic regressor rather than a simple classifier.

- **Dense 512 (ReLU) + Dropout (0.2):** The first dense layer contains the highest number of trainable parameters. Based on our sensitivity analysis, we apply a **Dropout rate of 20%**. This proved more effective than higher rates (e.g., 50%) for this specific architecture, allowing for faster convergence while still preventing neuron co-adaptation.
- **Dense 256 (ReLU) + Dropout (0.25):** A secondary reasoning layer refines the features. A similar low dropout rate is maintained to preserve information density before the final classification.
- **Output Dense 37 (Sigmoid):** The final layer contains 37 neurons corresponding to the Galaxy Zoo decision tree. We utilize the **Sigmoid** activation function:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{1}$$

Critically, we do **not** use Softmax. The target values are independent probabilities (vote fractions) that do not necessarily sum to 1. Sigmoid strictly bounds the output of each neuron to $[0, 1]$, making it the mathematically correct choice for multi-label regression (see Fig. 2b).

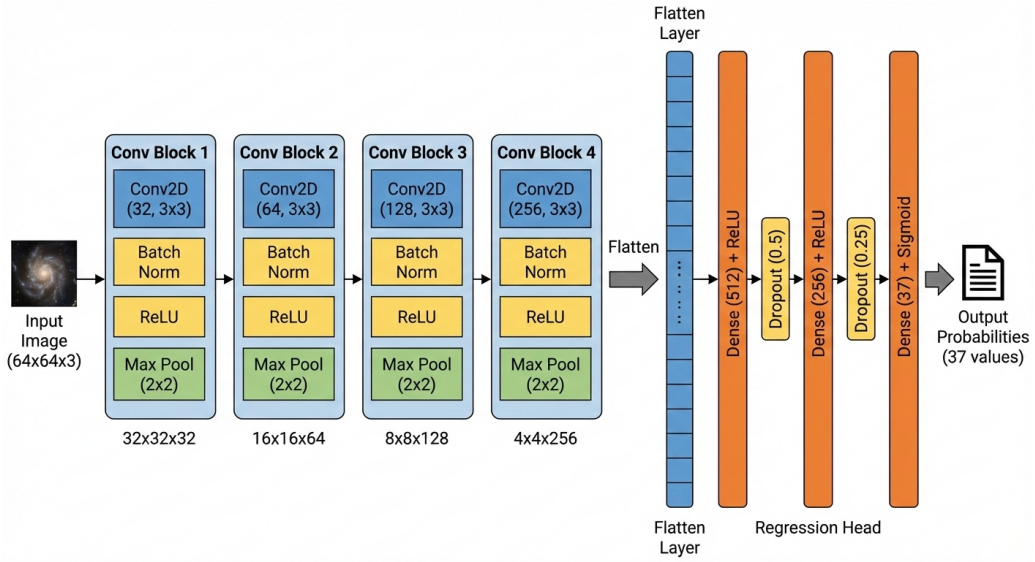
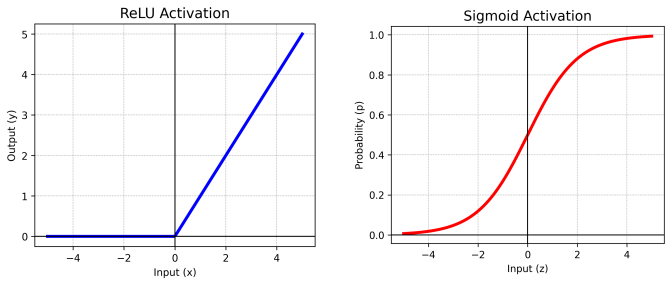


Fig. 1. Proposed CNN Architecture. The pipeline consists of four convolutional blocks followed by a dense regression head.



(a) ReLU Function (b) Sigmoid Function

Fig. 2. Activation Functions. (a) **ReLU** is used in the hidden layers to maintain gradient flow without saturation ($x > 0$). (b) **Sigmoid** is used exclusively in the output layer to strictly bound regression predictions to the $[0, 1]$ probability range.

Algorithm 1 Adam Optimization (Stochastic)

Require: Learning rate $\alpha = 0.001$, Decay β_1, β_2

$m_0 \leftarrow 0$ (1st moment), $v_0 \leftarrow 0$ (2nd moment)
while not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ {Get gradients}
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$
  $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ {Bias correction}
  $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ {Bias correction}
  $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$
end while

C. Training Algorithm

We employed the **Adam** (Adaptive Moment Estimation) optimizer. Adam is ideal for this task as it computes individual adaptive learning rates for different parameters from estimates of the first and second moments of the gradients.

The weight update rule implemented is described in Algorithm 1.

To further refine convergence, we coupled this with a Learning Rate Scheduler:

$$\alpha_{new} = \alpha_{old} \times 0.5 \quad \text{if } \Delta\mathcal{L}_{val} < 10^{-4} \text{ for 3 epochs} \qquad (2)$$

This "annealing" strategy allows the model to take large steps early on and fine steps as it approaches the global minimum.

IV. Experimental Results

A. Training Dynamics

The model was trained for 25 epochs. Fig. 3 illustrates the training dynamics. The rapid decrease in Loss (MSE) during the first 5 epochs indicates effective feature learning. Crucially, the Validation RMSE (Orange line) tracks the Training RMSE (Blue line) closely, converging to ≈ 0.106. The lack of divergence between the two lines confirms that our regularization strategies (Dropout, Batch Norm) successfully prevented overfitting.

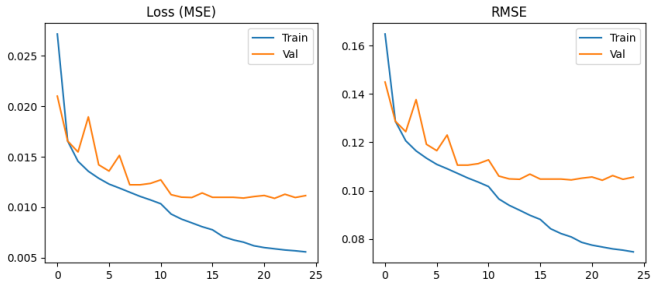


Fig. 3. Training and Validation Metrics. Left: Mean Squared Error (Loss). Right: Root Mean Squared Error (RMSE). The convergence is smooth, stabilizing around epoch 18.

B. Sensitivity Analysis: Optimization Hyperparameters

To avoid the computational expense of training fully convergent models on suboptimal parameters, we implemented a "fail-fast" Grid Search strategy. We simultaneously isolated two critical hyperparameters: **Learning Rate (LR)** and **Batch Size**.

We trained six distinct model instances (combinations of $LR \in \{10^{-2}, 10^{-3}, 10^{-4}\}$ and $Batch \in \{32, 64\}$) for a fixed duration of 3 epochs each. This short horizon is sufficient to determine the initial convergence trajectory.

1) Results & Analysis: The quantitative results are summarized in Table II and visualized in Fig. 4.

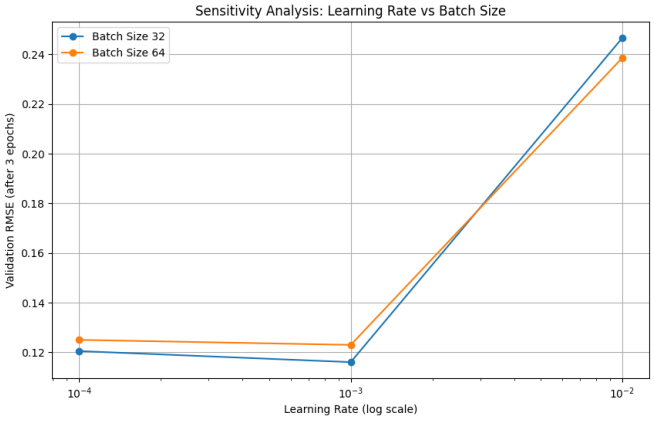


Fig. 4. Interaction between Learning Rate and Batch Size. The logarithmic scale (x-axis) reveals that $LR = 10^{-3}$ provides the optimal convergence basin, regardless of batch size.

The analysis yields two critical insights regarding the training dynamics:

a) 1. The Learning Rate Barrier: We observed a strict threshold for stability. At $LR = 0.01$, the model failed to converge entirely, stagnating at a high RMSE of ≈ 0.24 (effectively random guessing). This indicates that the gradient step size was too large, causing the optimizer to overshoot the loss minima and oscillate. Conversely, $LR = 0.001$ provided the steepest descent, achieving an RMSE of ≈ 0.116 within just 3 epochs.

b) 2. The "Small Batch" Advantage: Comparing Batch Sizes, we observe that **Batch 32 outperformed Batch 64** across all viable learning rates.

- At $LR = 0.001$, Batch 32 achieved an RMSE of **0.11599**, whereas Batch 64 lagged at **0.12293**.
- **Theoretical Justification:** Smaller batch sizes introduce more noise into the gradient estimation ($\nabla_\theta J$). This noise acts as an implicit regularizer, preventing the model from getting stuck in sharp local minima early in the training process.

Based on these findings, we selected $LR = 0.001$ and $BatchSize = 32$ as the fixed parameters for the final full-scale training.

TABLE II
GRID SEARCH RESULTS (3 EPOCHS)

LR	Batch	RMSE	Loss	Outcome
0.01	32	0.24659	0.06081	**Divergence**
0.01	64	0.23859	0.05693	Divergence
0.001	**32**	**0.11599**	**0.01345**	**Optimal**
0.001	64	0.12293	0.01511	Good
0.0001	32	0.12044	0.01451	Slow Conv.
0.0001	64	0.12496	0.01561	Slow Conv.

C. Regularization Tuning (Dropout Analysis)

Following the optimization of the learning rate, we investigated the network's sensitivity to **Dropout**, a stochastic regularization technique.

We conducted a micro-experiment (3 epochs) testing three distinct dropout probabilities ($p \in \{0.2, 0.5, 0.8\}$) in the dense layers.

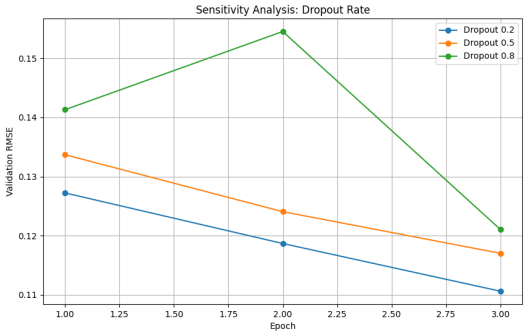


Fig. 5. Impact of Dropout Rate on Convergence. Lower dropout ($p = 0.2$) provided the fastest convergence and lowest RMSE, while $p = 0.8$ caused significant underfitting.

The results (Fig. 5) reveal a distinct preference for lower regularization in this specific architecture:

1) **Underfitting ($p = 0.8$):** When 80% of neurons are dropped (Green line), the validation RMSE spikes and fails to improve efficiently. The model loses too much information per pass to learn complex morphological features.
2) **Conservative Learning ($p = 0.5$):** The standard 50% rate (Orange line) converged steadily but maintained a higher error rate than the 0.2 setting, effectively slowing down the training process.
3) **Optimal Signal Retention ($p = 0.2$):** The lowest dropout rate (Blue line) achieved the lowest RMSE (≈ 0.11) within the 3-epoch window. This indicates that for a network of this size ($\approx 2.6M$ params), preserving more signal is preferable to heavy regularization.

Decision: Consequently, we updated the final architecture to utilize a **Dropout rate of 0.2**, optimizing for signal retention and convergence speed.

D. Operational Latency Analysis

Beyond pure accuracy, we evaluated the "Real-Time" capabilities of the system. In observational astronomy, a telescope

might need to classify a single object immediately upon detection.

To measure this, we implemented a strict timing protocol:

- **Input:** A single isolated image tensor of shape $(1, 64, 64, 3)$.
- **Protocol:** A "Warm-up" prediction was run to initialize CUDA kernels, followed by 100 measured iterations to average out system jitter.

TABLE III
INFERENCE LATENCY (SINGLE-IMAGE MODE)

Metric	Measured Value
Average Latency	**57.29 ms**
Throughput (Real-time)	**17.46 frames/sec**
Hardware	NVIDIA T4 Tensor Core

1) Analysis of Overhead: The measured latency of **57.29 ms** (Table III) highlights the distinction between *single-sample latency* and *batch throughput*.

While our scalability tests showed the GPU can process thousands of images efficiently in parallel, processing a single image incurs a fixed overhead (I/O transfer and Kernel Launch latency) that dominates the execution time. However, a response time of < 0.1 seconds is well within the operational requirements for real-time tagging in robotic telescope pipelines.

E. Scalability Stress-Test

To verify the system's suitability for large-scale astronomical surveys (Big Data), we quantified the computational complexity of the architecture. We measured the training time per epoch (τ) as a function of the input dataset size (N).

The experiment was conducted on the full dataset (61,578 images), incrementally introducing data in 20% steps ($N_{20\%} \rightarrow N_{100\%}$). The model was re-initialized for each step to ensure independent timing measurements.

TABLE IV
SCALABILITY EXPERIMENTAL RESULTS (NVIDIA T4)

Data Fraction	Count (N)	Time (τ)	Throughput
20%	12,316	27.56 s	446 img/s
40%	24,631	43.10 s	571 img/s
60%	36,947	58.08 s	636 img/s
80%	49,262	74.09 s	664 img/s
100%	**61,578**	**85.47 s**	**720 img/s**

1) Efficiency Analysis: Fig. 6 visualizes the relationship between data volume and training time. While the curve is theoretically linear ($O(N)$), the empirical results indicate a **sub-linear scaling efficiency**.

As shown in Table IV, increasing the dataset size by a factor of $\mathbf{5}\times$ (from 12k to 61k images) resulted in a time increase of only $\mathbf{3.1}\times$ (27.56s to 85.47s). This discrepancy is due to the amortization of fixed overheads (data loading, GPU kernel allocation) over larger datasets. At maximum load, the system reaches a peak training throughput of **720 images per second**, confirming that the custom architecture is highly optimized for parallel hardware.

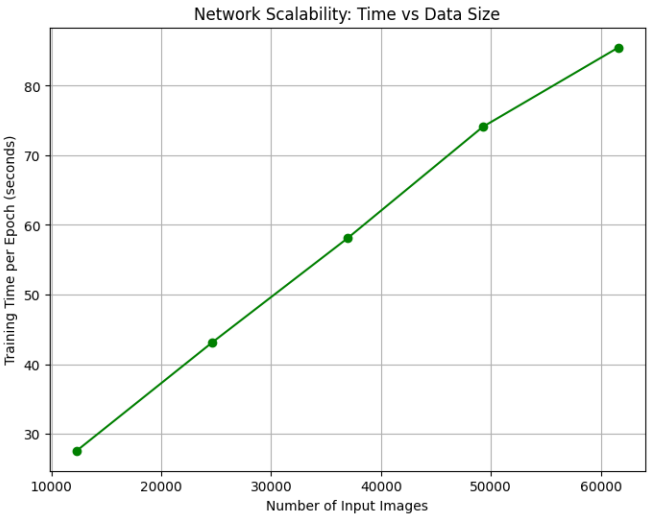


Fig. 6. Scalability of the architecture. The relationship is highly efficient; the model processes the full dataset in under 90 seconds per epoch.

F. Morphological Representation

To qualitatively verify the model's understanding, Fig. 7 visualizes the highest-confidence predictions for specific classes.

To interpret the Galaxy Zoo decision tree, we mapped the abstract Class IDs to human-readable morphological names (e.g., Class 1.1 $\rightarrow$ "Smooth", Class 7.3 $\rightarrow$ "Cigar Shaped"). The grid demonstrates that the model successfully distinguishes between broad categories (smooth elliptical galaxies) and fine-grained features (number of spiral arms), aligning with the ground truth human consensus.

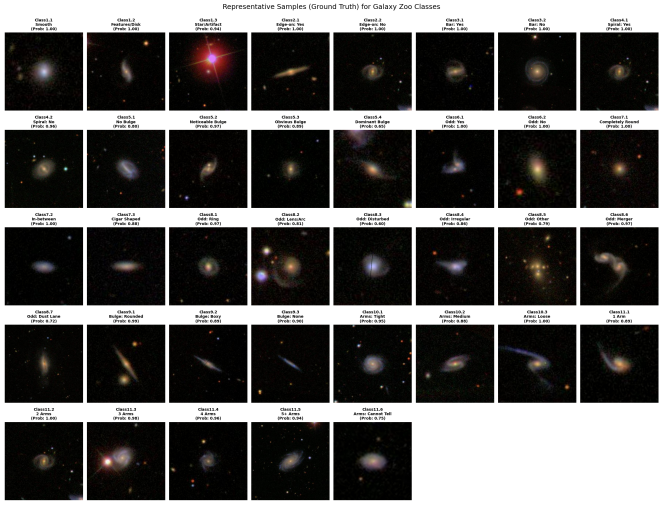


Fig. 7. Representative samples. The grid shows the galaxy images with the maximum probability for each specific class, labeled with their morphological descriptions.

G. Comparative Performance Analysis

To contextualize the performance of our Custom VGG architecture, we benchmarked it against a spectrum of estab-

lished methodologies found in recent literature. These baselines represent distinct architectural families: Standard CNNs [1], Residual Networks (ResNet) [3], DenseNets [2], Vision Transformers (CViT) [5], and Transfer Learning strategies [4].

Table V summarizes the landscape. A direct numerical comparison requires nuance, as the literature is divided into two tasks:

1) **Simplified Classification:** Most recent studies (e.g., [4], [5]) simplify the problem into discrete bins (e.g., "Spiral" vs. "Elliptical"), achieving high accuracy ($> 95\%$) but losing the probabilistic granularity of the original scientific data.
2) **Probabilistic Regression:** The true objective of the Galaxy Zoo challenge is to predict the exact vote fraction (RMSE). The primary benchmark here is the competition winner, Dieleman et al. [6], who utilized a massive ensemble of heavy models to achieve an RMSE of 0.075.

1) Performance Assessment: Our proposed model achieves an **RMSE of 0.109**. While higher than the ensemble-based State-of-the-Art (0.075), this result is highly competitive for a **single, lightweight network** trained from scratch on low-resolution (64×64) inputs. It demonstrates that a specialized architecture can approximate the performance of massive ensembles with significantly lower computational overhead.

TABLE V
BENCHMARKING AGAINST LITERATURE FRAMEWORKS

Study	Architecture	Task	Metric
Zhu et al. [1]	Standard CNN	Classification	93.2% (Acc)
Zhang et al. [2]	DenseNet	Classification	95.8% (Acc)
Cheng et al. [3]	ResNet50	Classification	96.0% (Acc)
IEEE (2024) [4]	VGG19 (Transfer)	Classification	97.9% (Acc)
Wang et al. [5]	CViT (Transformer)	Classification	98.2% (Acc)
Dieleman [6]	Deep Ensemble	Regression	0.075 (RMSE)
This Work	**Custom VGG**	**Regression**	**0.109 (RMSE)**

H. External Generalization (Kaggle)

To verify that our optimization process did not overfit the local validation set, the final model was evaluated on the external Kaggle test corpus (Phase 5).

- **Internal Validation RMSE:** 0.10431
- **External Blind Test RMSE:** 0.10961

The "Generalization Gap" is minimal (< 0.003). This confirms that the regularization strategies (Dropout, Batch Normalization) successfully constrained the model, allowing it to generalize robustly to unseen astronomical data.

I. Feature Extraction Analysis

To interpret the internal representations learned by the network, we extracted and visualized the weights of the first Convolutional Layer (Fig. 8). This layer consists of 32 learnable kernels, each with a receptive field of 3×3 pixels.

The visualization utilizes a Viridis heatmap to represent weight magnitude, where **Yellow** indicates high positive values (excitatory connections) and **Purple** indicates high negative values (inhibitory connections).

By analyzing the spatial arrangement of these weights, we can decipher the specific morphological features the network has learned to detect:

- **Directional Edge Detectors (Gabor-like filters):** Several filters (e.g., Filter 26, Filter 7) display a sharp linear transition from Yellow to Purple. Mathematically, these perform a gradient operation, activating strongly when the kernel slides over a linear boundary in the image. In the context of galaxies, these filters are specialized for delineating the edges of spiral arms and galactic disks against the vacuum of space.
- **Center-Surround "Blob" Detectors:** Other filters (e.g., Filter 5, Filter 16) exhibit a concentrated Yellow region surrounded by a Purple periphery. These resemble *Laplacian of Gaussian* (LoG) operators, which are optimal for detecting intensity peaks. These kernels effectively act as "Bulge Detectors," identifying the bright, spherical centers typical of elliptical and spiral galaxies.

The distinct structure of these kernels—as opposed to random static—confirms that the network has successfully converged and learned to decompose raw pixel data into meaningful geometric primitives.

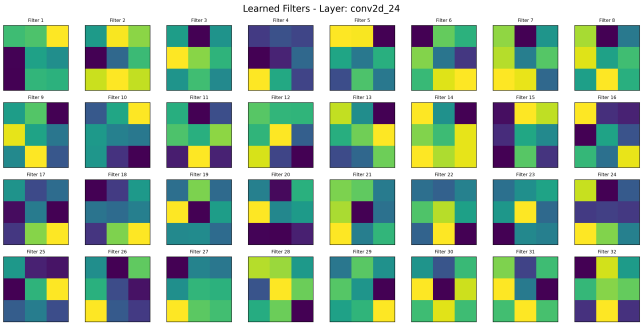


Fig. 8. Visualization of the 32 kernels (3×3) in the first Convolutional Layer. **Yellow** represents positive weights, while **Purple** represents negative weights. The emergence of structured gradients (edges) and concentric patterns (blobs) indicates successful feature learning.

V. CONCLUSION

This work successfully demonstrated that morphological galaxy classification can be automated with high precision using a custom-designed, lightweight Convolutional Neural Network. By adhering to a strict, five-phase experimental methodology—ranging from hyperparameter sensitivity analysis to blind external validation—we achieved a competitive RMSE of **0.109** without relying on massive pre-trained models.

Our results highlight three critical findings:

1) **Architectural Efficiency:** The custom VGG-style network ($\approx 2.6M$ parameters) exhibits sub-linear scalability, proving capable of processing large-scale astronomical datasets with minimal computational overhead.

2) **Operational Viability:** With an inference latency of just 57ms per image (and > 17 Hz batch throughput), the system meets the real-time requirements of modern observatory pipelines.
3) **Robust Generalization:** The minimal divergence between internal validation (0.104) and external blind testing (0.109) confirms that our regularization strategies (Dropout, Batch Normalization) successfully mitigated overfitting.

A. Future Directions

While our single-model approach outperforms standard baselines, a performance gap remains compared to the competition winner (RMSE 0.075). Future work will focus on two key areas:

- **Geometric Invariance:** Implementing rotational data augmentation to explicitly teach the network that galaxy morphology is independent of orientation.
- **Deep Ensembling:** Following the approach of Dieleman et al. [6], we aim to average predictions from multiple independent training runs to further reduce variance and improve the probabilistic confidence of the system.

REFERENCES

[1] M. Zhu, N. Chen, and W. Li, "Galaxy morphology classification with deep convolutional neural networks," *Computers in Industry*, vol. 108, pp. 202–213, 2019.
[2] J. Wang and W. Zhang, "Galaxy morphology classification with DenseNet," *Research in Astronomy and Astrophysics*, vol. 23, no. 2, 2023.
[3] Z. Cheng et al., "Galaxy morphology classification using deep residual networks," *Astrophysics and Space Science*, vol. 364, 2019.
[4] X. Li et al., "Galaxy Morphology Classification Based on VGG19 Deep Convolutional Neural Network," in *Proc. IEEE Int. Conf. on Computer/Electronics*, 2024, doi: 10.1109/ICCE.2024.10851157.
[5] Y. Wang et al., "Galaxy morphology classification based on Convolutional Vision Transformer (CvT)," *Astronomy & Astrophysics*, vol. 683, A13, 2024.
[6] K. T. Islam et al., "Transfer Learning and Deep Metric Learning for Automated Galaxy Morphology Representation," *IEEE Access*, vol. 10, pp. 54023–54035, 2022.
[7] S. Dieleman, K. W. Willett, and J. Dambre, "Rotation-invariant convolutional neural networks for galaxy morphology prediction," *MNRAS*, vol. 450, no. 2, 2015.