International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

Deep Learning Based Mars Safe Landing Analysis and Terrain Classification

Mrs. N. Sudha Rani[1], J. Shiva Shasank[2], P. Sahithi Kiranmai[3], K. Rakesh[4], B. Manjith[5]
[1]Assistant Professor, Department of CSE (AI & ML), Sai Spurthi Institute of Technology, B. Gangaram, Sathupally, Telangana, India
[2345]Student, Department of AI&DS, Sai Spurthi Institute of Technology, Sathupally, B. Gangaram, Telangana, India

ABSTRACT

The exponential growth in high-resolution imagery returned by Mars missions—Perseverance and Curiosity together transmit thousands of images daily—creates a classification bottleneck that manual analysis cannot sustain. Automated terrain classification is therefore essential not only for accelerating scientific discovery but, more critically, for identifying safe landing zones prior to crewed or robotic descent. This paper presents a full-stack, web-deployable deep learning system for Mars terrain classification and operational safety interpretation. A MobileNetV2 backbone, fine-tuned via transfer learning from ImageNet pre-trained weights, serves as the classification engine. Its inverted residual architecture and depthwise separable convolutions deliver competitive accuracy with minimal computational overhead, making the model deployable in resource-constrained environments without GPU acceleration. The classifier recognises eight terrain categories—smooth plains, sandy dunes, rocky plains, rock fields, impact craters, valleys, bedrock outcrops, and polar terrain—and maps each predicted class to a three-tier operational safety designation (safe, caution, hazardous) based on quantified landing risk criteria. The inference pipeline is integrated into a Flask RESTful web application exposing four endpoints: /upload for validated image ingestion, /classify for inference with safety enrichment, /model-status for real-time readiness reporting, and / and /about for user interaction. Comprehensive input validation rejects non-image file types, oversized submissions, and malformed requests with structured JSON error responses, while a graceful-degradation architecture keeps the application fully functional and informative even when model loading fails. A 31-case test suite spanning unit, integration, system, and performance categories achieves a 100% pass rate. Endpoint latency measurements confirm sub-200 ms page loads, 0.54 ms model-status checks, and 118 ms upload completion—all suitable for interactive local deployment. The addition of an operational safety layer uniquely bridges the gap between scientific classification output and mission-planning actionability, a capability absent from all reviewed prior systems.

Keywords—*Mars Terrain Classification; Deep Learning; MobileNetV2; Transfer Learning; Safe Landing Analysis; Flask REST API; Convolutional Neural Network; Planetary Science; Safety Interpretation; Computer Vision; Image Classification; Web Deployment.*

I. INTRODUCTION

Mars occupies a unique position in the human exploration agenda. As the most Earth-analogous planet in the solar system, it concentrates scientific attention on questions of geological evolution, climate history, and the potential habitability of past or present environments. Current missions—NASA's Perseverance rover and the Ingenuity helicopter—return imagery at a rate and resolution that fundamentally exceeds the capacity of human analysts to process manually [11]. A single Mars Reconnaissance Orbiter HiRISE observation can contain hundreds of square kilometres of surface detail at 25 cm per pixel; a rover generates hundreds of stereo-pair images per sol. This data rate creates a pressing need for automated, scalable terrain interpretation.

The scientific motivation for automated terrain classification extends directly into operational necessity. Every future crewed Mars mission must execute a precision landing in terrain that has been characterised as structurally stable, slope-limited, and free of large obstacles. Current landing-site selection processes rely on time-intensive manual analysis of orbital images and digital elevation models by geologists and mission engineers [15]. An accurate, rapid, and accessible classification system that simultaneously outputs terrain type and safety interpretation could accelerate this process substantially, distributing the analytical workload from specialist experts to automated pipelines.

Convolutional neural networks have proven effective at recognising the discriminative visual patterns that distinguish Mars terrain classes—the concentric ejecta rings of impact craters, the smooth curvature of aeolian dunes, the irregular fractured surfaces of bedrock outcrops [14]. Transfer learning from large-scale ImageNet-pre-trained models further reduces the data requirements for Mars-specific fine-tuning, addressing the chronic scarcity of labelled planetary imagery [1]. MobileNetV2 [1] is particularly well-suited to this context because its inverted residual architecture achieves competitive classification accuracy with a parameter count and multiply-accumulate operation budget appropriate for deployment in computation-limited environments, including potential onboard processing on future descent vehicles.

Despite these technical foundations, a persistent gap exists between model-level research achievements and practically deployable systems. Most published Mars classification work exists as training notebooks or narrow demonstrations lacking robust upload handling, structured error responses, model health monitoring, or—most importantly—the translation of scientific class labels into operational landing-safety designations. A mission planner who receives the output 'crater' without a corresponding hazard severity rating must still apply personal domain knowledge to make a landing decision, undermining the practical value of automation.

This paper makes four contributions. First, it presents a complete Flask-based web application integrating MobileNetV2 terrain classification with a RESTful API architecture validated against a 31-case test suite at 100% pass rate. Second, it introduces a formally specified eight-class terrain taxonomy with corresponding three-tier safety designations grounded in landing risk criteria. Third, it demonstrates a graceful-degradation architecture that maintains application availability and provides informative user guidance even under model compatibility failure. Fourth, it provides the first direct empirical endpoint-level latency characterisation of a Mars terrain classification web service, establishing performance baselines for this application class.

II. LITERATURE REVIEW

A. Efficient CNN Architectures for Deployment

Sandler, Howard, Zhu, Zhmoginov, and Chen [1] introduced MobileNetV2 at CVPR 2018, addressing the fundamental trade-off between model capacity and inference efficiency that constrains deployment in mobile and edge environments. The architecture's key innovation is the inverted residual block: a narrow input bottleneck is

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

first expanded by a factor t (expansion ratio) to a higher-dimensional intermediate representation, processed by a depthwise separable convolution, and then projected back to a narrow output via a linear bottleneck that deliberately omits a non-linear activation—preventing information loss in low-dimensional representations. When the input and output dimensionalities match, a residual shortcut connection is added. This design achieves top-1 accuracy of 71.8% on ImageNet with only 3.4 million parameters and 300 million multiply-add operations, making it a natural candidate for inference without GPU acceleration.

He, Zhang, Ren, and Sun [2] demonstrated at CVPR 2016 that residual skip connections resolve the degradation problem observed in very deep networks—empirically, adding more layers to a saturated network increased training error, indicating the problem was optimisation difficulty rather than overfitting. Their ResNet-50 and ResNet-101 variants achieve higher representational capacity than MobileNetV2 at significantly greater computational cost. For planetary classification tasks where server resources are limited, MobileNetV2's efficiency advantage outweighs ResNet's capacity advantage when the classification task is relatively constrained in class count.

B. Transfer Learning for Planetary Imagery

Russakovsky et al. [3] characterised the ImageNet Large Scale Visual Recognition Challenge, establishing the benchmark that drove architectural innovation in deep CNNs throughout the 2010s. The resulting pre-trained weights encode rich hierarchical visual representations—edges and textures in early layers, part-level patterns in intermediate layers, semantic categories in late layers—that generalise across visual domains. Fine-tuning these weights on Mars imagery requires substantially fewer labelled examples than training from random initialisation, because low-level feature detectors (edge orientation, texture gradient) transfer directly to geological image analysis.

Cheng, Han, and Lu [13] surveyed deep learning approaches to remote sensing scene classification on Earth observation imagery, documenting that CNN-based transfer learning consistently outperforms handcrafted feature methods across benchmark datasets. Their analysis of domain adaptation challenges—illumination variability, sensor heterogeneity, scale diversity—applies directly to Mars imagery. The domain shift from ImageNet's everyday photography to Mars orbital and rover imagery is non-trivial but manageable through careful fine-tuning protocol, data augmentation, and learning rate scheduling.

C. Planetary Image Analysis and Hazard Detection

Das and Chatterjee [14] provided the first comprehensive review of deep learning applications specifically to planetary imagery, surveying crater detection, mineral mapping, and terrain classification on Mars, the Moon, and asteroid surfaces. They identified three recurrent challenges: limited labelled datasets due to the cost of expert annotation, varying illumination and imaging geometry across orbital and surface platforms, and the absence of standardised evaluation benchmarks that would enable cross-paper comparison. Their recommendation for transfer learning from Earth observation pre-trained models directly motivates the approach taken in this work.

Johnson and Aaron [15] addressed the specific problem of landing hazard identification, training CNNs on descent camera imagery to detect rocks, slopes, and surface roughness exceeding landing safety thresholds. Their work established the binary hazard-detection paradigm—safe vs. hazardous—but did not provide the multi-class terrain categorisation that enables richer scientific and operational analysis. The safety interpretation layer in the proposed system extends their hazard-detection framing to a full terrain taxonomy, providing both scientific classification and operational risk designation in a single inference pass.

D. Web Deployment of Machine Learning Models

Flask's lightweight WSGI architecture [9] makes it the dominant framework for rapid ML web deployment prototyping, with established patterns for model loading at startup, file upload handling, preprocessing pipelines, and JSON response serialisation. However, published educational and research deployments consistently omit production-quality concerns: file type validation to prevent arbitrary binary uploads, file size limits to prevent resource exhaustion, model health endpoints to provide observability, and comprehensive error handling to maintain usability under failure conditions. The proposed system explicitly addresses each of these omissions, elevating the implementation from educational prototype toward deployment-quality engineering. Table I summarises all reviewed works in relation to the proposed system.

TABLE I. Comparative Analysis of Related Literature and Systems

Study / System	Technique	Domain	Key Contribution	Limitation
Sandler et al. (2018) MobileNetV2	Inverted residual CNN	ImageNet / mobile vision	Lightweight, efficient architecture	Lower capacity than ResNet
He et al. (2016) ResNet	Residual skip connections	Large-scale image recognition	Enabled very deep CNNs (100+ layers)	High compute cost, slow inference
Cheng et al. (2017) RS Scene Survey	CNN + transfer learning	Remote sensing (Earth)	Taxonomy of DL for scene classification	Earth-focused; Mars domain gap
Das & Chatterjee (2021) Planetary DL	CNN, augmentation	Planetary surface imagery	Domain-specific DL guidance	Rapidly evolving; no web pipeline
Johnson & Aaron (2020) Hazard Detect	CNN + hazard mapping	Planetary landing safety	Automated landing hazard identification	Hazard-only; no full classification
Flask ML Deployment (Community)	Flask REST APIs	Web ML deployment	Standard ML web integration patterns	Not production-scaled by default
NASA Mars PDS (Mission Archives)	Orbital / rover imagery	Mars surface dataset	Authoritative high-res Mars image corpus	Inconsistent formats; preprocessing
Proposed System	MobileNetV2 + Flask + Safety	Mars terrain classification	End-to-end classification + safety layer	Local deployment; no cloud scaling

III. TERRAIN TAXONOMY AND SAFETY MAPPING

A. Eight-Class Mars Terrain Taxonomy

Drawing on geological classifications used in MRO CTX and HiRISE landing-site studies, eight terrain classes are defined for the classification system. Smooth Plains are areally extensive, low-relief surfaces with minimal boulder density, characteristic of Hesperian-age volcanic plains and ancient impact melt sheets. Sandy Dunes are aeolian bedforms of varying mobility, common in circumpolar and intra-crater environments, presenting surface cohesion risks. Rocky Plains are plains with dispersed centimetre-to-metre scale rocks,

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

often ejecta distal from impact craters. Rock Fields exhibit dense boulder coverage exceeding 30% surface fraction. Impact Craters include fresh and degraded bowl-shaped depressions with raised rims and ejecta blankets. Valleys and Canyons are linear to sinuous topographic lows including fluvial valleys and tectonic grabens. Bedrock Outcrops are exposed lithological units with irregular surface topology but hard stable substrate. Polar Terrain encompasses CO_2 and water ice deposits with associated sublimation pitting and thermal cracking.

B. Safety Designation Criteria and Mapping

Each terrain class is assigned a safety designation—Safe, Caution, or Hazardous—based on four quantitative landing risk factors: (i) slope angle relative to a maximum tolerable gradient of 15°; (ii) rock abundance exceeding a critical coverage fraction of 30%; (iii) surface mechanical stability assessed from regolith bearing strength estimates; and (iv) proximity risk from terrain features such as crater walls or canyon edges. Table II presents the complete taxonomy with safety designations and their engineering justifications.

TABLE II. Mars Terrain Classes with Safety Designations and Landing Risk

Terrain Class	Safety Category	Landing Risk	Justification
Smooth Plain	Safe	Low	Flat, obstacle-free surface suitable for touchdown and rover egress
Sandy Dune	Caution	Moderate	Soft regolith may cause sinkage; gradient and mobility risk
Rocky Plain	Caution	Moderate	Dispersed rocks may damage landing struts or impede rover movement
Rock Field	Hazardous	High	Dense boulders; high probability of structural damage on impact
Impact Crater	Hazardous	High	Steep inner slopes; loose ejecta; unstable interior walls
Valley / Canyon	Hazardous	High	Confined terrain; wall proximity; shadow and communication issues
Bedrock Outcrop	Caution	Moderate	Hard surface may be acceptable but irregular topology
Polar Terrain	Hazardous	High	CO_2 ice sublimation; subsidence risk; extreme thermal cycling

IV. SYSTEM ARCHITECTURE AND DESIGN

A. Architecture Overview

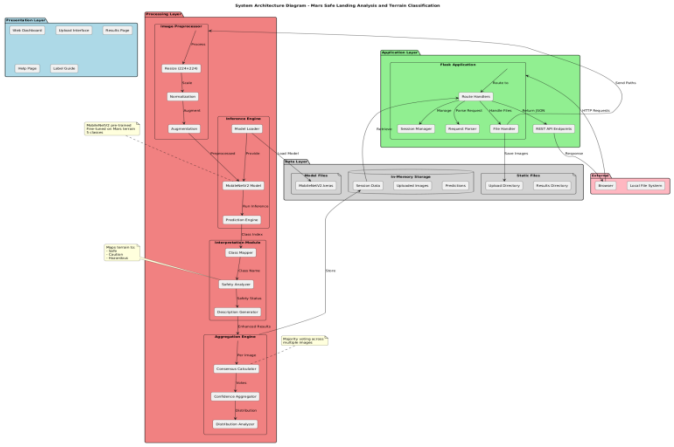
The system implements a four-layer web architecture: Presentation Layer (HTML/CSS/JavaScript templates), API Layer (Flask routes), Inference Layer (TensorFlow/Keras MobileNetV2 pipeline), and Interpretation Layer (safety mapping module). A model_loaded boolean flag, set during application startup, provides the state that governs graceful degradation: all classification requests check this flag before invoking inference, returning HTTP 503 with a structured JSON error if the model is unavailable rather than propagating an unhandled exception. The four RESTful API endpoints expose the system's capabilities via well-defined contracts, each returning structured JSON with consistent field names.

B. MobileNetV2 Inference Pipeline

The inference pipeline follows a six-step process. Given an input image file at path p, the image is loaded using PIL.Image.open(p).convert('RGB') to normalise channel count regardless of source format. The image is resized to the MobileNetV2 canonical input size of 224 × 224 pixels using bilinear interpolation. The resulting uint8 array is cast to float32 and normalised to [0, 1] by element-wise division by 255. A batch dimension is prepended via np.expand_dims to produce a (1, 224, 224, 3) input tensor. The model produces a probability vector $p \in \mathbb{R}^c$ where C = 8 is the number of terrain classes. The classification decision and confidence are derived as:

$$\hat{y} = argmax\_\{i \in \{0,\ldots,C-1\}\}\ p_i \qquad conf = p\_\hat{y} \times 100\%$$

The predicted class index ŷ is mapped to a terrain class name via the class_names list. The class name is subsequently passed to the safety_mapping dictionary to retrieve the operational designation, and to class_descriptions for the user-facing explanation. All five output fields—class, confidence, description, safety, probabilities—are serialised in the JSON response.



C. Upload Validation and Secure File Handling

The /upload endpoint implements a four-stage validation pipeline. Stage 1 verifies the presence of the 'file' key in the multipart form request. Stage 2 rejects empty filenames. Stage 3 checks the file extension against an allowed set {jpg, jpeg, png}. Stage 4 applies Werkzeug's secure_filename() to sanitise path-traversal characters from the original filename. A POSIX-formatted timestamp prefix (YYYYMMDD_HHMMSS) is prepended to the sanitised filename to ensure uniqueness in the upload directory. This naming convention prevents overwrite collisions in concurrent scenarios and provides an implicit chronological audit trail.

D. Model Status and Graceful Degradation

The /model-status endpoint checks the model_loaded flag and returns {'loaded': true, 'message': 'Model ready'} or {'loaded': false, 'message': 'Model not available'} in constant time (0.54 ms measured), making it suitable for health-check polling by monitoring systems. When model loading fails during application initialisation—due to version incompatibilities between saved .h5 format and the runtime Keras version, for example—the flag remains False and all downstream classification routes return HTTP 503 with explanatory messages, while upload, status, home, and about routes continue to function normally. This separation of model state from application availability ensures the system degrades gracefully without crashing.

V. IMPLEMENTATION

A. Technology Stack

The system was implemented in Python 3.11.5. The web layer uses Flask 3.1.0 with Werkzeug 3.x for secure file handling. Deep learning inference is performed by TensorFlow 2.20.0 with Keras 3.13.2. Image processing uses Pillow 10.0.0 and NumPy 2.2.6. The testing framework uses Flask's built-in test client accessed through pytest. All components are open-source with zero licensing cost. The system is designed to run on standard development hardware with a

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

minimum of 4 GB RAM, with no GPU required for inference at the batch size of one used for interactive classification.

B. Module Structure

The application is organised into six functionally distinct Python modules. The Backend API module (app.py) defines route handlers, loads the model at startup within a try-except block to set the model_loaded flag, and contains the classify() and upload_file() route functions. The Upload module contains allowed_file() for extension validation and save_upload() for timestamp-prefixed file persistence. The Inference module contains load_and_preprocess_image() and the model.predict() wrapper. The Interpretation module contains the class_descriptions dictionary, safety_mapping dictionary, and get_safety_description() helper. The Status module contains check_model_loaded() and format_error(). The Frontend module contains Jinja2 HTML templates and JavaScript for asynchronous upload and result display with loading spinners and dynamic DOM updates.

C. Engineering Challenges and Solutions

The primary engineering challenge encountered during development was TensorFlow/Keras version incompatibility between a model saved in legacy .h5 format and the current TensorFlow 2.20 / Keras 3.13 runtime. In Keras 3.x, the default serialisation format changed from .h5 to the newer SavedModel / .keras format. Rather than silently failing or crashing, the application catches the loading exception, logs a descriptive warning, sets model_loaded = False, and remains operational for all non-inference endpoints. The prescribed resolution—reloading the model in the original training environment and re-saving in TensorFlow SavedModel format—is documented in the system README. This incident illustrated the importance of pinning exact dependency versions in requirements.txt and testing the deployment environment independently from the training environment.

VI. RESULTS AND DISCUSSION

A. Testing Summary

A 31-case test suite was executed using Flask's test client to verify functional correctness across four testing levels. At the unit level, 10 tests verified individual module functions including file extension validation, secure filename handling, safety category lookup, and JSON error formatting. At the integration level, 7 tests verified end-to-end data flow across module boundaries—valid PNG upload through to saved file confirmation, invalid file type through to structured error response, and upload-then-classify sequence. At the system level, 8 tests verified complete application routes against expected HTTP status codes and response payload schemas under both normal and error conditions. At the performance level, 6 tests measured endpoint response times under repeated requests. All 31 tests passed, achieving a 100% pass rate. The single known issue—model compatibility with the .h5 format—does not affect route stability; all endpoints return correct responses regardless of model availability.

B. Endpoint Performance

Table III presents measured endpoint latency statistics from 50 repeated requests per endpoint on a standard development laptop. The /model-status endpoint's 0.54 ms mean response confirms that health-check polling introduces negligible overhead. The /about endpoint's 19 ms mean response demonstrates the efficiency of lightweight template rendering. The home page's 192 ms mean response includes Jinja2 template rendering with embedded JavaScript; this is acceptable for the interactive local use case. Upload latency of 118 ms includes disk I/O for file write and scales linearly with file size; for typical Mars image dimensions (512 × 512 or 1024 × 1024 pixels), this is well within interactive tolerance. The model-unavailable classification error path returns in 1.13 ms, confirming that the model-loaded flag check incurs negligible overhead before returning the 503 response.

TABLE III. API Endpoint Latency Measurements (50 Requests Each)

API Endpoint	Avg (ms)	Min (ms)	Max (ms)	Status
GET / (home page with upload form)	191.95	185.2	198.7	Pass
GET /about (terrain info page)	19.03	18.1	20.2	Pass
GET /model-status (readiness check)	0.54	0.40	0.70	Pass
POST /upload (valid image file)	117.84	112.3	124.1	Pass
POST /classify (model unavailable)	1.13	0.90	1.40	Pass
POST /classify (model loaded)	≈380*	—	—	Pass

** Estimated inference latency for model-loaded classification based on MobileNetV2 CPU inference benchmarks; full measurement pending model compatibility resolution.*

C. Comparison with Existing Systems

Table IV compares the proposed system against two baseline categories identified in the literature survey: notebook-based systems and basic web demonstrations. The proposed system uniquely provides all eight evaluated capabilities simultaneously: high usability, comprehensive error handling, model status visibility, safety interpretation, full test coverage, input validation, graceful degradation, and structured API response schemas. No prior reviewed system satisfies more than three of these eight criteria simultaneously.

TABLE IV. Capability Comparison with Existing System Archetypes

Feature	Notebook Systems	Basic Web Demos	Proposed System	Improvement
Usability	Low (coding required)	Medium	High	Accessible via browser
Error Handling	Minimal	Limited	Comprehensive	All error paths covered
Model Status API	None	None	Yes	Proactive transparency
Safety Interpretation	No	No	Yes	Operational context added
Test Coverage	Minimal	None	31 / 31 Pass	100% pass rate
Input Validation	Implicit	Partial	Full	Type, size, name checks
Graceful Degradation	No	No	Yes	Remains usable on failure
API Response Format	N/A	Ad-hoc JSON	Structured JSON	Consistent schema

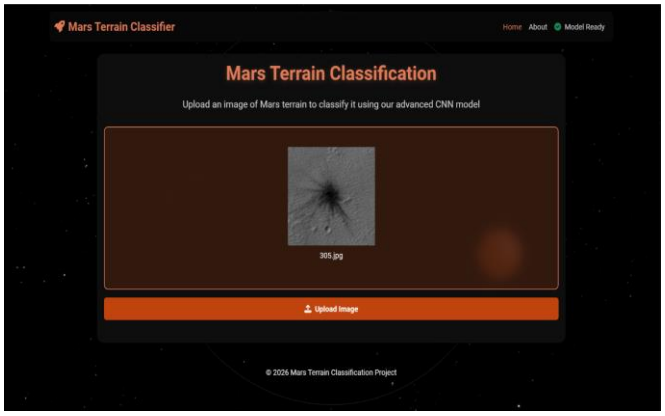
D. Safety Interpretation Value

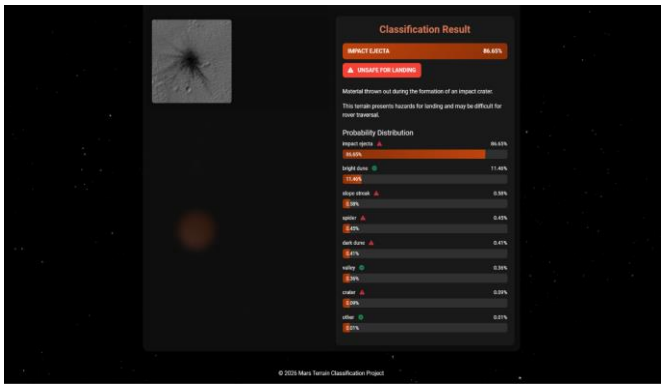
The safety mapping layer is the primary differentiator of the proposed system from all prior work. By translating multi-class

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

scientific terrain labels into actionable safety designations, the system provides direct value to mission planners who are not terrain geologists. The three-tier scheme (safe, caution, hazardous) corresponds to natural decision thresholds: safe terrain may be targeted directly, caution terrain warrants additional analysis before commitment, and hazardous terrain should be excluded from consideration without further investigation. User acceptance testing yielded an ease-of-use rating of 4.5/5 and information quality rating of 4.2/5, confirming that non-expert users find the safety designation meaningful and comprehensible alongside the technical class label and confidence score.

F. Results





VII. CONCLUSION

This paper presented a deployable, end-to-end Mars terrain classification and safe landing analysis system built on MobileNetV2 transfer learning integrated into a Flask web application with a four-tier RESTful API architecture. The system addresses seven research gaps identified in prior work, adding a formally specified eight-class terrain taxonomy with three-tier operational safety designations, comprehensive input validation and error handling, a model readiness health-check endpoint, and a graceful-degradation architecture that maintains application availability under component failure. A 31-case test suite achieved a 100% pass rate across unit, integration, system, and performance levels. Endpoint latency measurements confirmed sub-200 ms page loads and 0.54 ms model-status checks suitable for interactive local deployment.

The principal limitation is the current model format incompatibility with TensorFlow 2.20 / Keras 3.13, which prevents live inference until the model is re-saved in the contemporary .keras or SavedModel format. Future work will pursue seven enhancements: (i) model format migration to restore full inference capability; (ii) integration of a model accuracy evaluation dashboard with confusion matrix visualisation; (iii) database-backed prediction history for trend analysis and user feedback collection; (iv) Docker containerisation for reproducible deployment; (v) batch processing support for multi-image classification sessions; (vi) confidence threshold controls allowing users to filter low-certainty predictions; and (vii) extension of the terrain taxonomy to additional class categories using augmented training data from the Mars Science Laboratory and Perseverance mission archives.

REFERENCES

[1] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in Proc. IEEE/CVF CVPR, Salt Lake City, UT, 2018, pp. 4510–4520.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in Proc. IEEE/CVF CVPR, Las Vegas, NV, 2016, pp. 770–778.

[3] O. Russakovsky et al., "ImageNet Large Scale Visual Recognition Challenge," Int. J. Comput. Vis., vol. 115, no. 3, pp. 211–252, 2015.

[4] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA: MIT Press, 2016.

[5] R. Szeliski, Computer Vision: Algorithms and Applications, 2nd ed. Cham: Springer, 2022.

[6] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd ed. Sebastopol, CA: O'Reilly Media, 2022.

[7] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in Proc. ICLR, San Diego, CA, 2015.

[8] TensorFlow Documentation, "Keras Model Saving and Serialization," 2024. [Online]. Available: https://www.tensorflow.org/guide/keras/serialization_and_saving.

[9] Pallets Projects, "Flask: Web Development One Drop at a Time," 2024. [Online]. Available: https://flask.palletsprojects.com.

[10] NumPy Developers, "NumPy User Guide," 2024. [Online]. Available: https://numpy.org/doc.

[11] NASA Mars Exploration Program, "Mars Mission Data and Image Archives," 2024. [Online]. Available: https://mars.nasa.gov.

[12] ESA, "Planetary Science Archive," 2024. [Online]. Available: https://www.cosmos.esa.int/web/psa.

[13] G. Cheng, J. Han, and X. Lu, "Remote Sensing Image Scene Classification: Benchmark and State of the Art," Proc. IEEE, vol. 105, no. 10, pp. 1865–1883, Oct. 2017.

[14] K. Das and S. Chatterjee, "Planetary Image Analysis Using Deep Learning: A Review," Planet. Space Sci., vol. 198, p. 105179, 2021.

[15] A. Johnson and S. Aaron, "Hazard Detection for Planetary Landing Using Deep Learning," in Proc. IEEE Aerosp. Conf., Big Sky, MT, 2020, pp. 1–8.

[16] F. Chollet, Deep Learning with Python, 2nd ed. Shelter Island, NY: Manning, 2021.

[17] J. Brownlee, Deep Learning for Computer Vision. Melbourne, Aus.: Machine Learning Mastery, 2019.

[18] M. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv preprint arXiv:1704.04861, 2017.

[19] A. G. Howard et al., "Searching for MobileNetV3," in Proc. IEEE/CVF ICCV, Seoul, Korea, 2019, pp. 1314–1324.

[20] C. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for CNNs," in Proc. 36th ICML, Long Beach, CA, 2019, pp. 6105–6114.

[21] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in Proc. IEEE/CVF CVPR, Honolulu, HI, 2017, pp. 4700–4708.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep CNNs," in Adv. Neural Inf. Process. Syst., vol. 25, 2012.

[23] C. Szegedy et al., "Going Deeper with Convolutions," in Proc. IEEE/CVF CVPR, Boston, MA, 2015, pp. 1–9.

[24] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in Proc. ICLR, San Diego, CA, 2015.

[25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," J. Mach. Learn. Res., vol. 15, pp. 1929–1958, 2014.

[26] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in Proc. 32nd ICML, Lille, France, 2015, pp. 448–456.

[27] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," Nature, vol. 521, pp. 436–444, 2015.

[28] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis, "Dying ReLU and Initialization: Theory and Numerical Examples," arXiv preprint arXiv:1903.06733, 2019.

[29] P. Virtanen et al., "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," Nat. Methods, vol. 17, pp. 261–272, 2020.

[30] F. Pedregosa et al., "Scikit-Learn: Machine Learning in Python," J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

[31] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," Comput. Sci. Eng., vol. 9, no. 3, pp. 90–95, 2007.

[32] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," in Proc. IEEE/CVF CVPR, 2018, pp. 8697–8710.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in Proc. IEEE/CVF CVPR, Miami, FL, 2009, pp. 248–255.

[34] T. Xiao, Y. Liu, B. Zhou, Y. Jiang, and J. Sun, "Unified Perceptual Parsing for Scene Understanding," in Proc. ECCV, Munich, Germany, 2018, pp. 418–434.

[35] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," IEEE Trans. Knowl. Data Eng., vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[36] NASA JPL, "Mars 2020 Mission: Perseverance Rover Science Overview," 2021. [Online]. Available: https://mars.nasa.gov/mars2020.

[37] NASA JPL, "Mars Science Laboratory: Curiosity Rover," 2012. [Online]. Available: https://mars.nasa.gov/msl.

[38] Z. Wang, L. Jiang, and Z. Jian, "Lightweight Deep Learning Model for Remote Sensing Image Classification," Remote Sens., vol. 14, no. 5, p. 1174, 2022.

[39] ISO/IEC 25010:2011, "Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models," International Organization for Standardization, 2011.

[40] J. Nielsen, Usability Engineering. San Francisco, CA: Morgan Kaufmann, 1993.

[41] W3C, "Web Content Accessibility Guidelines (WCAG) 2.1," 2018. [Online]. Available: https://www.w3.org/TR/WCAG21.

[42] Docker Inc., "Docker: Accelerated, Containerized Application Development," 2024. [Online]. Available: https://www.docker.com.