Project: From Model to Production: Scalable Batch Processing for Sustainable Retail

DLBDSMTP01 Task 2: Batch Image Classification for Refund Department

Alejandro Moral Aranda

IU University

August 24, 2025

Live Web Version

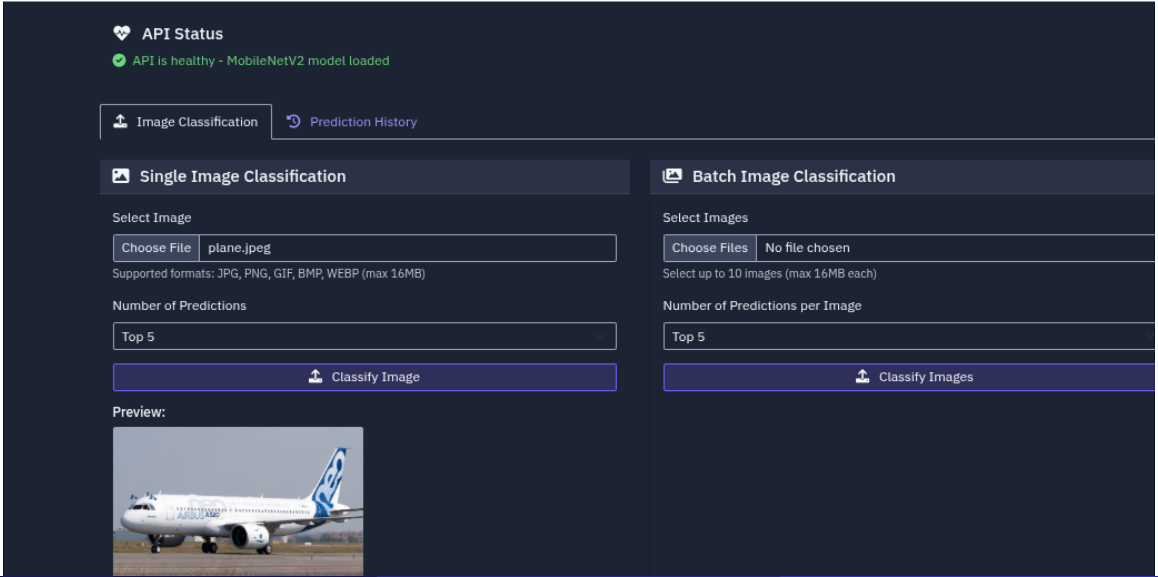
Try it online now: https://api-fenlei-production.up.railway.app/

Hosted on Railway (free plan). First request may be slow after inactivity.

Also runnable locally.

Source Code: github.com/alemxral/api-fenlei

Web Interface (Screenshot)



API Status

API is healthy - MobileNetV2 model loaded

Single Image Classification

Select Image

plane.jpeg

Supported formats: JPG, PNG, GIF, BMP, WEBP (max 16MB)

Number of Predictions

Top 5

Preview:

Batch Image Classification

Select Images

No file chosen

Select up to 10 images (max 16MB each)

Number of Predictions per Image

Top 5

Outline

1. Project Overview
2. Preparations & Planning
3. Architecture
4. Challenges & Requirements
5. Risks, Timeline & Resources
6. Results & Reflection
7. Summary & Conclusion

Project Overview

Goal: Automate refund item categorization from images to reduce manual effort and costs.

Solution: REST API for batch image classification using MobileNetV2.

Deployment: Hosted on Railway (public demo), also supports local runs.

Key Features:

- Batch and single image processing.
- Web interface for interactive/manual testing.
- Structured API responses and validation.

Objectives

Reduce manual sorting workload.

Enable nightly batch classification jobs.

Ensure robust data processing and validation.

Provide easy-to-use manual testing UI.

Preparations

Model: Pre-trained MobileNetV2 (ImageNet).

Environment: Python 3.11, TensorFlow/Keras, Flask.

Validation: File type, size, dimensions, integrity checks.

Prototyping: Verified local inference and batch processing.

Project Planning

Architecture: Defined ingestion, storage, processing, and API flows.

Code Structure: Modular organization (API, ML logic, utilities, UI).

Deployment: Public demo on Railway, portable for local use.

Why This Approach?

Pre-trained Model: Fast, high-quality baseline.

MobileNetV2: Efficient CPU inference.

Batch Processing: Designed for nightly automation.

Flask: Simple, reliable RESTful backend.

Railway: Easy/cheap cloud demo deployment.

System Architecture

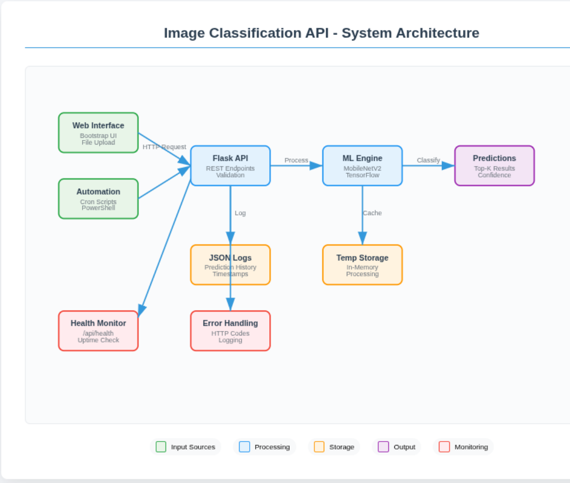
Frontend: HTML/CSS, Vanilla JS for UI and API interaction.

Backend: Flask REST API.

ML Model: MobileNetV2.

Batch: Scheduled jobs (cron, GitHub Actions).

Storage: JSON logs, optional DB backend.



System architecture diagram

API Endpoints

`POST /api/classify` Classify a single image.

`POST /api/classify/batch` Batch classification (up to 10 images).

`GET /api/health` API health check.

`GET /api/info` Build/config info.

Error Handling

Clear JSON errors for invalid format, size, or corrupted images.

Checks for image dimensions (324096px) and MIME/type.

Consistent and meaningful HTTP status codes.

Batch Processing

Nightly jobs for batch image classification.

Triggered by cron or GitHub Actions.

Logs written to temporary storage or database.

Integration Challenges

Speed Needs:

Batch: high throughput.

Interactive: low latency.

Scalability:

Handle more users/data via horizontal scaling.

Consistent Preprocessing:

Same data pipeline for training and serving.

Security & Privacy:

Encryption, access control.

Resource Limits:

Stay within budget/hardware constraints.

Model Versioning:

Track and roll back deployed models.

Data Requirements

Getting Images: Collect photos easily from warehouse devices.

Where to Store:

- Save images in object storage (like S3 or minIO).
- Keep results in JSON files or a database.
- Save extra info (metadata) in a database (like PostgreSQL).

Processing Steps:

- Check and label images for quality.
- Always use the same cleaning and prep steps.

How It Works: Special programs check, organize, and save every image and its info.

Monitoring for Reliability

Data Quality: Resolution, corruption, metadata checks.

Prediction Drift: Monitor class distribution vs. baseline.

Performance: Latency, throughput, error rates.

Model Quality: Periodic accuracy/F1 checks.

Alerting: Threshold-based notifications.

Audit Logs: Track prediction events and model versions.

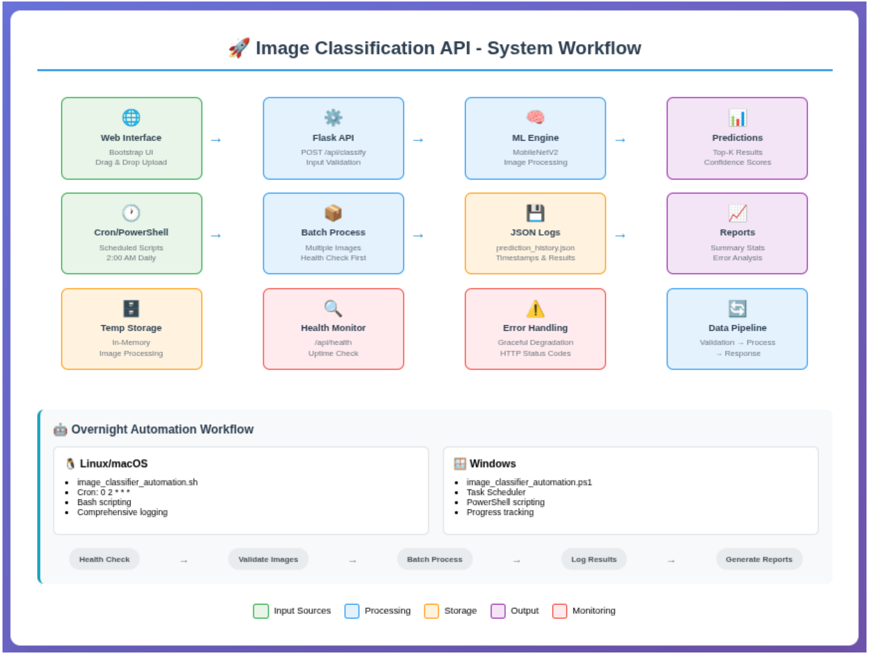


Image Classification API - System Workflow

Web Interface
Bootstrap UI
Drag & Drop Upload

→

Flask API
POST /api/classify
Input Validation

→

ML Engine
MobileNetV2
Image Processing

→

Predictions
Top-K Results
Confidence Scores

Cron/PowerShell
Scheduled Scripts
2:00 AM Daily

→

Batch Process
Multiple Images
Health Check First

→

JSON Logs
prediction_history.json
Timestamps & Results

→

Reports
Summary Stats
Error Analysis

Temp Storage
In-Memory
Image Processing

Health Monitor
/api/health
Uptime Check

Error Handling
Graceful Degradation
HTTP Status Codes

Data Pipeline
Validation → Process
→ Response

Overnight Automation Workflow

Linux/macOS

- image_classifier_automation.sh
- Cron: 0 2 * * *
- Bash scripting
- Comprehensive logging

Windows

- image_classifier_automation.ps1
- Task Scheduler
- PowerShell scripting
- Progress tracking

Health Check → Validate Images → Batch Process → Log Results → Generate Reports

🟩 Input Sources 🟦 Processing 🟧 Storage 🟪 Output 🟥 Monitoring

Risks & Mitigations

Infra: Cold starts, resource limits on free hosting.

Compatibility: TensorFlow/Pillow binary dependencies.

Security: File upload validation, size caps.

Performance: CPU-only inference latency.

Mitigations: Health checks, format validation, worker tuning.

Resources

Compute: Railway free tier, local CPU for development/testing.

Software: Open-source stack (Flask, TensorFlow/Keras, Pillow).

Results

Delivered REST API and batch processing for refund categorization.

Public demo: https://api-fenlei-production.up.railway.app/.

Met objectives for functionality and usability.

Reflection

Process: Iterative development for fast feedback.

Resources: Free-tier hosting is viable for demos; scalability remains to be tested.

Lessons: Early validation and health endpoints reduce downtime.

Summary & Conclusion

Developed a scalable API to automatically classify refund item images, reducing manual effort and errors.

Supports both real-time and batch processing, making it flexible for different needs.

Modular design allows easy updates, integration, and future improvements.

The project demonstrates how AI can streamline retail workflows and can be extended to new use cases.

API Usage: curl Examples

```
# Health check
curl https://api-fenlei-production.up.railway.app/api/health

# Single image classification (local)
curl -X POST -F "image=@path/to/image.jpg" -F "top_k=5" \
  http://localhost:5000/api/classify

# API information
curl http://localhost:5000/api/info
```

Bibliography

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen.
MobileNetV2: Inverted Residuals and Linear Bottlenecks.
In CVPR, 2018.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, Li Fei-Fei.
ImageNet: A Large-Scale Hierarchical Image Database.
In CVPR, 2009.

Martín Abadi et al.
TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015.
https://www.tensorflow.org/

François Chollet et al.
Keras, 2015.
https://keras.io/

Armin Ronacher.a
Flask (Python Web Framework).
https://flask.palletsprojects.com/

Thank You

Questions?

Live Demo: https://api-fenlei-production.up.railway.app/

Source Code: github.com/alemxral/api-fenlei