Intel Image Classification

End-to-End Deep Learning Pipeline

Ali Traore

12. Januar 2026

Agenda

Projektübersicht

Ziel

End-to-End Bildklassifikationssystem für 6 Landschaftskategorien mit Production-ready API und Dashboard

Kategorien:

- Buildings
- Forest
- Glacier
- Mountain
- Sea
- Street

Tech Stack:

- TensorFlow/Keras
- FastAPI + Uvicorn
- Streamlit + Plotly
- Docker Compose

Projektkomponenten

Vollständige ML Pipeline:

1. **Datenverarbeitung:** Data loading, augmentation, preprocessing
2. **Modelltraining:** 3 CNN-Architekturen mit verschiedenen Strategien
3. **Evaluation:** Metriken, Visualisierungen, Grad-CAM
4. **API Backend:** FastAPI mit automatischer Dokumentation
5. **Frontend:** Multi-page Streamlit Dashboard
6. **Deployment:** Docker Compose für einfache Bereitstellung

Dataset

Intel Image Classification:

- $\sim$25.000 Bilder
- 150$\times$150 RGB
- 6 Klassen (balanciert)

Split:

- Train: $\sim$14.000
- Validation: $\sim$3.000
- Test: $\sim$7.000

Preprocessing:

- Normalisierung [0,1]
- Data Augmentation:
  - Rotation ($\pm 20^\circ$)
  - Horizontal Flip
  - Zoom (10%)
  - Shift (10%)
- Batch Size: 32

Modellübersicht

Modell	Parameter	Val Accuracy	Strategie
Baseline CNN	$\sim$2M	73%	Simple CNN
Regularized CNN	$\sim$2M	79%	+ L2 + Dropout
MobileNetV2	$\sim$3.5M	85-90%	Transfer Learning

Implementierung:

- `src/models/`: baseline.py, regularized.py, transfer_learning.py
- `src/training/`: Unified trainer + callbacks
- `scripts/train_pipeline.py`: Komplette Pipeline

Modelle im Detail

Baseline CNN:

- $3\times$ Conv2D + MaxPooling
- Flatten + Dense
- Dropout (0.5)
- Problem: Overfitting (73% val)

Regularized CNN:

- + Batch Normalization
- + L2 Regularization
- + Dropout (0.6)
- Besser: 79% val

MobileNetV2:

- Pre-trained (ImageNet)
- Custom Head:
  - GlobalAvgPooling
  - Dense(256)
  - Dropout(0.5)
  - Dense(6)
- Beste: 85-90% val

Training & Evaluation

Training Features:

- EarlyStopping (patience=10)
- ModelCheckpoint
- ReduceLROnPlateau
- TensorBoard Logging
- CSV Metrics Export

Implementierung:

- `src/training/trainer.py`
- `src/training/callbacks.py`
- `scripts/train_pipeline.py`

Evaluation:

- Accuracy, Precision, Recall, F1
- Confusion Matrix
- Training/Validation Curves
- Grad-CAM Visualisierung

Outputs:

- `outputs/models/*.keras`
- `outputs/figures/`
- `outputs/logs/`

FastAPI Backend

RESTful API (`api/main.py`):

Endpoints:

- `GET /` - Health Check
- `GET /models` - List Models
- `POST /predict/{model}` - Classify
- `POST /feedback` - User Feedback

Features:

- Lazy Model Loading
- Model Caching
- CORS Support
- Auto Documentation (Swagger)

Modelle:

- baseline
- regularized
- transfer_learning

Deployment:

- Uvicorn ASGI Server
- Port 8000
- Docker Container
- Health Checks

Streamlit Dashboard

Multi-Page Dashboard (`streamlit/`):

Seiten:

- `_Home.py`: Upload & Predict
- `_Model_Comparison.py`: Vergleich
- `_Data_Analysis.py`: Dataset Stats
- `_Feedback_Dashboard.py`: Analytics

Components:

- `status_panel.py`
- `visualization.py`

Features:

- Drag & Drop Upload
- Model Selection
- Confidence Scores
- Grad-CAM Heatmaps
- Confusion Matrices
- Plotly Charts
- Feedback System

Port: 8501

Docker Deployment

Docker Compose Setup:

Services:

- `api`: FastAPI (Port 8000)
- `streamlit`: Dashboard (Port 8501)

Features:

- Health Checks
- Volume Mounts
- Auto Restart
- Service Dependencies

Usage:

- `docker compose up --build`
- `docker compose up -d`
- `docker compose logs -f`
- `docker compose down`

Vorteile:

- Einfaches Deployment
- Reproduzierbar
- Portabel

Ergebnisse

Modell	Train Acc	Val Acc	F1-Score
Baseline	82%	73%	0.71
Regularized	84%	79%	0.78
MobileNetV2	91%	88%	0.88

Beste Performance: MobileNetV2 Transfer Learning

- Validation Accuracy: 88.5%
- Alle Klassen: 85-92% Accuracy
- Herausforderung: Glacier vs. Mountain Verwechslung

Lessons Learned

Technische Erkenntnisse:

- **Transfer Learning:** MobileNetV2 deutlich besser als Custom CNNs
- **Regularisierung:** Batch Norm + L2 + Dropout reduziert Overfitting
- **Data Augmentation:** Essentiell für Generalisierung
- **FastAPI:** Schnelle API-Entwicklung mit Auto-Docs
- **Streamlit:** Einfache Dashboard-Erstellung
- **Docker Compose:** Vereinfacht Multi-Service Deployment

Ausblick

Mögliche Erweiterungen:

Features:

- Ensemble Models
- Batch Prediction
- Video Classification
- Real-time Webcam

Production:

- Cloud Deployment
- CI/CD Pipeline
- Monitoring (Prometheus)
- Model Versioning (MLflow)

Zusammenfassung

Projektergebnisse:

- 3 CNN-Modelle: Baseline, Regularized, Transfer Learning
- Beste Accuracy: 88.5% (MobileNetV2)
- FastAPI Backend mit 4 Endpoints
- Multi-Page Streamlit Dashboard
- Docker Compose Deployment
- Grad-CAM Interpretierbarkeit
- Komplette Dokumentation (README, RUN.md)

Fazit

End-to-End ML Pipeline: Von Daten bis Production-Deployment

Projektstruktur

Code Organization:

Core ML:

- `src/models/`
- `src/training/`
- `src/evaluation/`
- `src/utils/`
- `scripts/train_pipeline.py`

Deployment:

- `api/main.py`
- `streamlit/app.py`
- `streamlit/pages/`
- `Dockerfile`
- `docker-compose.yml`

Vielen Dank!

Fragen & Diskussion

Ali Traore

Demo: `docker compose up --build`
API: http://localhost:8000/docs
Dashboard: http://localhost:8501