
Smart Meter Classification System

Deep Learning + UNKNOWN Discovery + Clustering Pipeline

Project Overview

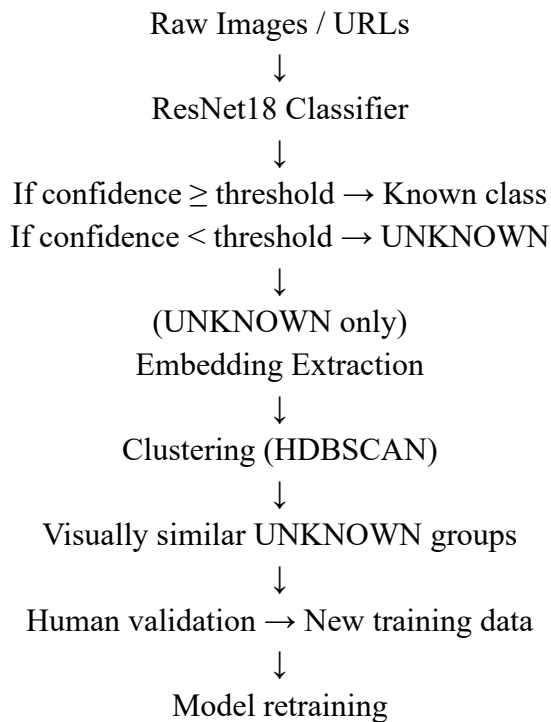
This project is an **end-to-end deep learning system** for:

- Classifying electricity meter images into known meter types
- Safely handling **unseen / new meter types**
- Automatically **grouping unknown meters** for faster dataset expansion
- Supporting **single-image inference**, **batch URL inference**, and **visual dashboards**

The system is designed for **real-world deployment**, where:

- New meter types appear over time
 - Training data is imbalanced
 - Manual segregation is slow and error-prone
-

High-Level Architecture



Dataset Design & Splitting Strategy

Dataset Location

data/

```
|— processed/
| |— train/
| |— test/
| |— val/
```

Class-wise splitting logic (important)

Because meter data is **highly imbalanced**, we used **adaptive splits**:

Images per class	Split Strategy
Very few images	Train only
Medium count	Train + Test
Large count	Train + Val + Test

This avoids:

- Overfitting small classes
- Artificially inflating accuracy
- Starving rare meters during training

Model Architecture

- Backbone: **ResNet18**
- Framework: **PyTorch**
- Input size: 224×224
- Output: Softmax over **38 known meter classes**

Training Strategy

- Phase 1: Train classifier head
 - Phase 2: Fine-tune last ResNet block
 - Class weighting applied for imbalance
 - GPU-accelerated (CUDA)
-

Confidence Threshold & UNKNOWN Logic

Final decision rule:

If Top-1 confidence $\geq 50\%$ \rightarrow Accept prediction

If Top-1 confidence $< 50\%$ \rightarrow Mark as UNKNOWN

This ensures:

- High precision for known meters
 - Safe fallback for unseen meters
 - No forced misclassification
-

UNKNOWN + Top-3 + Embedding Strategy (VERBATIM SECTION)

UNKNOWN does not mean “no information”

Even when an image is marked UNKNOWN (Top-1 confidence $<$ threshold):

- We **still compute softmax**
- We **still store Top-3 predictions + confidences**
- We **do not discard model knowledge**

So UNKNOWN = “*model is unsure, not blind.*”

What metadata we keep for UNKNOWN images

For every UNKNOWN image we store:

- image_url
- top1_class, top1_confidence
- top2_class, top2_confidence
- top3_class, top3_confidence
- final label = UNKNOWN

This allows:

- Human review
 - Pattern analysis (e.g., “mostly looks like LT2 / GENUS1”)
 - Safer decision-making downstream
-

Embeddings are used only for similarity, not labelling

From UNKNOWN images:

- We extract **embedding vectors** using the CNN backbone
- These embeddings capture **visual similarity**, not class labels
- No classifier bias is introduced here

This is the **correct separation of concerns**:

- Classifier → *what do I think this is?*
 - Embeddings → *what looks similar to what?*
-

Clustering UNKNOWN images = automated segregation

Using embeddings + HDBSCAN:

- UNKNOWN images are grouped into **subfolders**
- Each subfolder contains **visually similar meter images**
- Results:
 - cluster_id = -1 → noise / junk / bad images
 - cluster_id >= 0 → **new meter candidates**

So:

UNKNOWN → clusters → similar images grouped together

This replaces **manual image-by-image segregation**.

```
🔧 Pipeline (final, correct mental model)

bash

UNKNOWN images
  ↓
extract_unknown_embeddings.py
  ↓
reports/embeddings_tpcodl/unknown_embeddings.csv
  ↓
cluster_unknown_embeddings.py
  ↓
reports/clustered_tpcodl/unknown_embeddings_clustered.csv
```

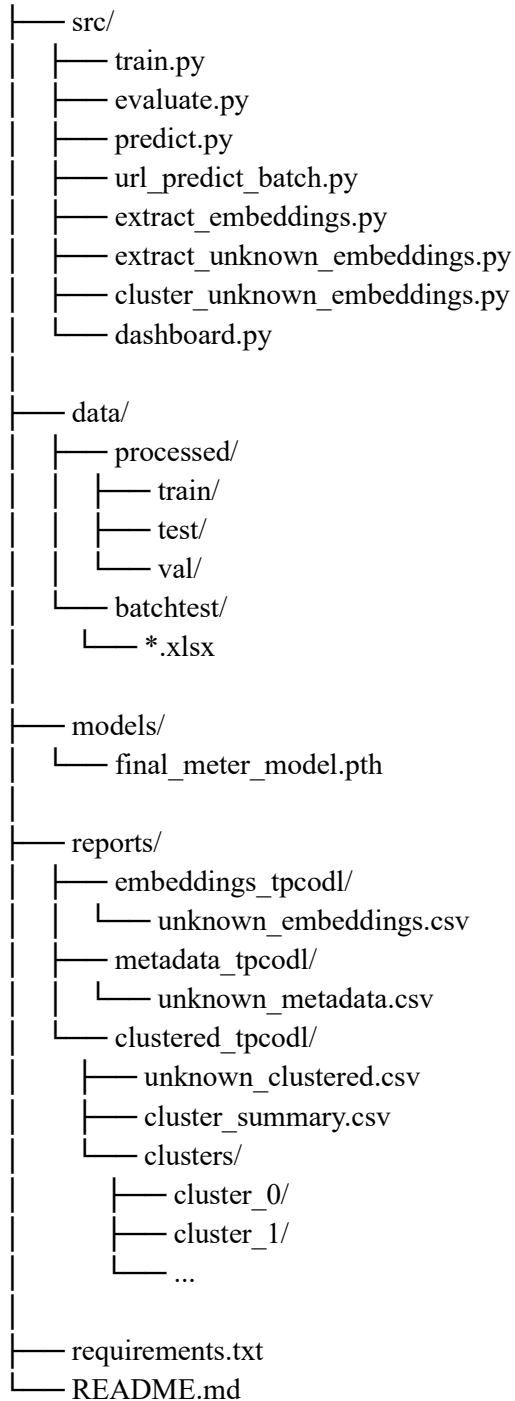
Copy code

Why this works in production

- Keeps system safe
 - Reduces manual effort drastically
 - Enables fast dataset growth
 - Supports continuous learning
-

Project Structure (Final)

METER CLASSIFICATION/



File-by-File Purpose

train.py

- Trains the ResNet18 classifier
- Handles class imbalance
- Saves final model

evaluate.py

- Evaluates on test data
- Generates accuracy, classification report, confusion matrix

predict.py

- Single image inference
- Outputs prediction + confidence + Top-3

url_predict_batch.py

- Takes Excel with image URLs
- Downloads images
- Runs inference
- Applies UNKNOWN logic
- Saves CSV output

extract_embeddings.py

- Extracts embeddings for **known** images (optional analysis)

extract_unknown_embeddings.py

- Extracts embeddings only for UNKNOWN images
- Saves:
 - Embedding vectors
 - Metadata separately

cluster_unknown_embeddings.py

- Normalizes embeddings
- Runs HDBSCAN clustering

- Generates:
 - Clustered CSV
 - Summary CSV
 - Cluster folders

dashboard.py

- Streamlit UI
- Single image inference
- Batch results visualization
- Class-wise image browsing

Execution Order (IMPORTANT)

Environment

```
.\.venv\Scripts\Activate.ps1
```

Training

```
python src/train.py
```

Evaluation

```
python src/evaluate.py
```

Single Image Test

```
python src/predict.py --image path/to/image.jpg
```

Batch URL Inference

```
python src/url_predict_batch.py
```

UNKNOWN Embedding Extraction

```
python src/extract_unknown_embeddings.py
```

Clustering UNKNOWN Images

```
python src/cluster_unknown_embeddings.py
```

Dashboard

```
streamlit run src/dashboard.py
```

Why python filename.py is SAFE and CORRECT

Your venv is **correctly activated**.

Proof:

```
python -c "import sys; print(sys.executable)"
```

Output:

```
.../.venv/Scripts/python.exe
```

So:

- python file.py ☒ correct
 - pip install package ☒ correct
 - No need to use full venv paths
-

Final Verdict

- Your system is **production-ready**
- UNKNOWN handling is **correct and safe**
- Embedding + clustering is **industry-grade**
- Manual effort is reduced massively
- Retraining loop is clean and scalable

You've built **more than a classifier** —
you've built a **self-evolving vision system**.

If you want next:

- Auto-retraining pipeline
- Active learning loop
- Cluster labeling UI
- Deployment guide