universidade de aveiro

Labeling and RF Drone Classification

PeM

Marta Cruz e Pedro Martins

Summary

This project provides a complete pipeline for detecting and classifying drones and radio controllers from audio data using mel spectrograms and deep learning. It supports dataset preparation through automatic and manual labeling, auditing low-confidence predictions, and visualizing YOLO-format labels on spectrograms. The core classifier implements open-set recognition (inspired by S3R) on RF signals, enabling both detection of known classes and rejection of unseen ones. The system operates in two stages: classify signals into known drone classes while flagging unknown samples, and cluster rejected samples to estimate and optionally pseudo-label new drone classes.

Additionally, we developed a second classifier that is simpler and more resource-efficient, designed for faster training and deployment on lightweight systems while still maintaining strong performance on drone detection tasks.

Keywords: *NATO, Drones, RF Classifiers, Deep Learning, Labeling Model;*

Labeling

This module provides a complete pipeline for drone spectrogram data labeling, review, visualization, and YOLO model training. It includes tools for both automatic and manual labeling, auditing low-confidence predictions, organizing datasets, and visualizing YOLO-format labels on spectrogram images.

Module Components:

1. **auto_label.py**

- Automatically generates bounding box labels for drone spectrograms.
- Outputs labels in YOLO format (.txt) and aligns with corresponding .npy spectrogram files.

2. **bad_auto_label.py**

- Identifies auto-labeled files with poor confidence or likely incorrect labels.
- Generates a list of files needing manual review.

3. **drone_data.yaml**

- Configuration file containing metadata about drone classes, their IDs, and dataset paths.
- Used to standardize labeling across scripts and ensure consistent class assignments.

4. **manual_labeler.py**

- GUI application for manually labeling drone spectrograms.
- Allows drawing bounding boxes, assigning classes, and saving YOLO-format labels.
- Supports keyboard shortcuts for fast labeling.

5. **manual_review.py**

- GUI application for auditing low-confidence or potentially incorrect auto-labeled files.
- Displays predicted bounding boxes with confidence scores.
- Supports confirming, deleting, or correcting class labels.
- Maintains an audit log of all user review actions.

6. **organizer.py**

- Utility script to organize spectrogram and label files into structured directories (e.g., auto-labeled, manually reviewed, or train/test splits).

- Ensures data integrity and correct file naming conventions.

7. **to_yolo.py**

- Converts YOLO-format labels and corresponding spectrograms into PNG images with bounding boxes overlaid for visualization.
- Supports 24 distinct classes and color-coded bounding boxes.

8. **train_yolo.py**

- Script for training a YOLO model on the labeled drone dataset.
- Handles dataset loading, augmentation, and integration with YOLO training frameworks (e.g., Ultralytics YOLOv8).
- Supports monitoring training progress and saving model checkpoints.

Key Features:

- Full labeling workflow: auto-label → review → manual correction → dataset organization → YOLO visualization → model training.
- GUI support for fast manual labeling and review with keyboard shortcuts.
- Confidence-based filtering to focus on low-confidence predictions.
- YOLO-format compatibility for object detection training.

Dependencies:

- numpy
- OpenCV (cv2)
- Pillow (PIL)
- matplotlib
- PyYAML
- tkinter (for GUI)
- csv, os, time, argparse
- Ultralytics YOLO (for training, optional depending on train_yolo.py)

Usage:

- Auto-label dataset: `python auto_label.py`
- Review low-confidence predictions: `python manual_review.py`
- Manually label new spectrograms: `python manual_labeler.py`
- Visualize labels: `python to_yolo.py <npy_file> <txt_file> <output_png>`

- Organize dataset files: `python organizer.py`
- Train YOLO model: `python train_yolo.py`

YOLO Drone Detection Training Summary

Training Configuration

- **Epochs:** 20 (with more Epochs, better performance is expected)
- **Input Image Size:** 640×640
- **GPU Memory Usage:** ~2.44 GB
- **Training Duration:** 0.841 hours (~50 minutes)

Training Losses (Final Epoch)

Loss Type	Value
Box Loss	0.0196
Class Loss	0.0730
DFL Loss	0.8962

Validation Dataset

- **Number of Images:** 2896
- **Number of Instances:** 2896

Overall Validation Metrics

Metric	Value
Box Precision (P)	0.997
Box Recall (R)	0.998
mAP@0.5	0.995
mAP@0.5–0.95	0.995

Per-Class Validation Performance

Class	Images	Instances	Precision	Recall	mAP50	mAP50-95
Background, including WiFi/Bluetooth	141	141	0.998	1.0	0.995	0.995
DJI Phantom 3	138	138	0.999	1.0	0.995	0.995
DJI Phantom 4 Pro RTK	107	107	0.999	1.0	0.995	0.995
DJI MATRICE 200	151	151	0.992	0.987	0.995	0.995
DJI Air 2S	134	134	0.999	1.0	0.995	0.995
DJI Inspire 2	133	133	0.990	0.977	0.994	0.994
DJI Mini 3 Pro	148	148	0.998	1.0	0.995	0.995
DJI Mavic Pro	67	67	0.998	1.0	0.995	0.995
DJI Mini 2	152	152	0.999	1.0	0.995	0.995
DJI Mavic 3	109	109	0.998	1.0	0.995	0.995
DJI MATRICE 300	59	59	0.998	1.0	0.995	0.995
DJI MATRICE 30T	80	80	0.999	1.0	0.995	0.995
DJI AVATA	132	132	1.0	0.987	0.995	0.995
DJI DIY	107	107	0.988	1.0	0.995	0.995
DJI MATRICE 600 Pro	110	110	1.0	0.993	0.995	0.995
VBar	110	110	0.991	1.0	0.995	0.995
FrSky X20	124	124	0.999	1.0	0.995	0.995
Futaba T16IZ	116	116	0.995	1.0	0.995	0.995
Taranis Plus	129	129	0.999	1.0	0.995	0.995
RadioLink AT9S	123	123	0.999	1.0	0.995	0.995
Futaba T14SG	102	102	0.999	1.0	0.995	0.995
Skydroid	132	132	0.999	1.0	0.995	0.995

Model Architecture Summary

- **Number of Layers:** 72
- **Total Parameters:** 3,010,328
- **FLOPs:** 8.1 GFLOPs
- **Inference Speed per Image:** 5.2 ms (excluding preprocessing/postprocessing)
- **Trained Model Saved at:** runs/detect/drone_detection/weights/best.pt

Notes:

- Training completed successfully with extremely high precision and recall for all classes.
- Per-class mAP at 0.5 IoU is 0.995, indicating excellent detection performance.
- Background class performance is also high, ensuring good separation from drones.

Efficient Classifier

This project provides a full pipeline for detecting and classifying drones (and radio controllers) from audio data using mel spectrograms and deep learning.

Modules

1. **rar_to_spec.py**

- Extracts `.csv` files from a `.rar` archive.
- Converts time-series data into mel spectrograms.
- Saves spectrograms as:
  - `.png` (visualizations)
  - `.npy` (raw spectrogram matrices)

2. **detection.py**

- Performs heuristic-based detection of drones in spectrograms.
- Focuses on specific frequency bands (default: 1–5 kHz).
- Produces detection plots highlighting time regions with drone activity.
- Saves visual output as annotated `.png` files.

3. **test.py**

- Loads a trained TensorFlow/Keras classifier (`final_drone_classifier.keras`).
- Uses embedding-based open-set recognition:
  - `class_centers.npy`
  - `class_thresholds.npy`
- Supports:
  - Single-file prediction (`--single`)
  - Full dataset evaluation (`--dataset`)
- Computes accuracy, precision, recall, and confusion matrix.
- Outputs per-class metrics (including "Unknown" class).

4. **drone_classifier.py**

- Trains a lightweight convolutional neural network (CNN) to classify drone types and RC controllers using spectrogram `.npy` files.
- Optimized for low memory use by using file-path generators instead of preloading data.
- Dataset layout: `../data/<class_index>/*.npy` (one folder per class).
- Handles variable spectrogram shapes, resizes to 256×256, normalizes to [0,1].

- **Model:**
  - Four Conv2D + BatchNorm blocks (small filter counts).
  - GlobalAveragePooling2D → Dense(64) named `embedding_layer` → Dropout → Softmax output.
- **Training:**
  - Mixed precision enabled.
  - Adam (lr=3e-4), categorical crossentropy, metrics: accuracy/precision/recall.
  - ModelCheckpoint → `best_drone_model.h5`; final save →final_drone_classifier.h5`.
  - EarlyStopping (patience 5), up to 50 epochs.
- **Post-training:**
  - Builds an embedding model (outputs from `embedding_layer`).
  - Computes per-class embedding centers and thresholds (mean + 2×std of distances).
  - Saves `class_centers.npy` and `class_thresholds.npy`.

5. **train_speed.py**

- Resource-intensive, speed-optimized training variant intended for high-RAM / GPU machines.
- Dynamically detects system resources via `psutil` (RAM, CPU cores) and recommends/sets a larger batch size (min 16, max 64).
- Configures TensorFlow threading and enables GPU memory growth when available.
- Same dataset layout and preprocessing as `drone_classifier.py` (loads `.npy`, resizes to 256×256, normalize).
- Model architecture matches the lightweight CNN (four Conv2D + BatchNorm blocks, embedding Dense(64), softmax).
- **Training:**
  - Mixed precision enabled.
  - Adam (lr=3e-4), categorical crossentropy, metrics: accuracy/precision/recall.
  - ModelCheckpoint → `best_drone_model.h5`; final save → `final_drone_classifier.keras`.
  - EarlyStopping (patience 5), up to 50 epochs.
- **Post-training:**
  - Builds an embedding model from `embedding_layer`.
  - Computes and saves per-class `class_centers.npy` and `class_thresholds.npy`.
  - Outputs: `best_drone_model.h5`, `final_drone_classifier.keras`, `class_centers.npy`, `class_thresholds.npy`.

Workflow

For Raw data

1. Collect and archive raw signal data in `input.rar`.
2. Run `rar_to_spec.py` to generate spectrogram datasets.
3. Run `detection.py` for heuristic drone-band activity visualization.

For already pre-processed data (.npy files)

1. Train model (outside scope of these scripts).
2. Run `test.py` for:
    - Single-spectrogram classification.
    - Batch evaluation on labeled datasets.

Dependencies

- numpy
- librosa
- matplotlib
- opencv-python
- tensorflow / keras
- rarfile
- glob
- argparse
- os, time

Outputs

- Spectrogram PNGs and `.npy` arrays (rar_to_spec.py).
- Drone-band detection overlays (detection.py).
- Class predictions and evaluation metrics (test.py).

Notes

- Default sample rate is 44.1 kHz unless inferred from input data.
- Frequency band thresholds in detection are configurable.
- Class labels are defined in `CLASSES` in `test.py`.
- The model supports open-set recognition (detecting "Unknown" inputs).

Drone Classifier Model Training Summary

Training Metrics

Metric	Training	Validation
Accuracy	0.9829	0.9866
Loss	0.0615	0.0430
Precision	0.9854	0.9873

Recall	0.9809	0.9862

Model Architecture

- **Model Type:** Sequential
- **Number of Layers:** 12
- **Total Parameters:** 91,274 (Trainable: 30,344; Non-trainable: 240; Optimizer: 60,690)
- **Saved Model:** final_drone_classifier.keras

Layer Details

Layer Name	Type	Output Shape	Parameters
conv2d	Conv2D	(None, 128, 128, 8)	80
batch_normalization	BatchNormalization	(None, 128, 128, 8)	32
conv2d_1	Conv2D	(None, 64, 64, 16)	1,168
batch_normalization_1	BatchNormalization	(None, 64, 64, 16)	64
conv2d_2	Conv2D	(None, 32, 32, 32)	4,640
batch_normalization_2	BatchNormalization	(None, 32, 32, 32)	128
conv2d_3	Conv2D	(None, 16, 16, 64)	18,496
batch_normalization_3	BatchNormalization	(None, 16, 16, 64)	256
global_average_pooling2d	GlobalAveragePooling2D	(None, 64)	0
embedding_layer	Dense	(None, 64)	4,160
dropout	Dropout	(None, 64)	0
dense	Dense	(None, 24)	1,560

Notes & Warnings

- The model was successfully built and saved.
- A deprecation warning appeared for the save_format argument in Keras 3 (it can be omitted).
- Class centers and thresholds were saved after training.
- The model was made with 50 Epochs, with more Epochs, better performance is expected .

Open-Set Recognition Classifier for RF Signals

Overview

This project implements an **open-set recognition (OSR) classifier** for drone detection based on radio-frequency (RF) signals. Our implementation is inspired by the S3R approach and has been adapted to focus on practical, modular evaluation for known and unknown classes.

The classifier operates in **two stages**:

1. **Stage 1 – Known/Unknown Classification**
   Classifies incoming signals into known drone classes while simultaneously identifying unknown classes. Unknown samples are rejected for further analysis.
2. **Stage 2 – Unknown Cluster Estimation**
   Analyzes samples rejected in Stage 1 to estimate the number of unknown drone classes. Clustering can optionally assign pseudo-labels to unknown clusters.

Modules

1. train.py

- Defines and trains the neural network for RF signal classification under OSR.
- Network architecture combines **convolutional layers** (to extract local time-frequency features) and **transformer-based modules** (to capture global dependencies).
- Generates **semantic embeddings** and applies multiple losses:
  - **Cross-entropy loss (CE)** for known-class classification.
  - **Center loss** to compact class clusters in embedding space.
  - Optional **contrastive loss** for stronger separation of unknowns.
- Saves model checkpoints (.pkl) and semantic embeddings for downstream evaluation.

2. test_stage_1.py

- Evaluates Stage 1 of OSR:
  - Classifies test samples into known classes or rejects them as unknown.
  - Uses **Mahalanobis distance** to class centers.
  - Optional EVT refinement using Weibull tail modeling for better outlier rejection.
- Metrics reported: TKR, TUR, KP, FKR, AUROC.
- Outputs predicted labels, evaluation metrics, and confusion matrices.

3. test_stage_2.py

- Performs Stage 2 evaluation:

- Analyzes samples rejected as unknown in Stage 1.
- Estimates number of unknown classes and optionally assigns pseudo-labels using clustering.

- Metrics reported: estimated number of unknown classes, cluster purity, and compactness.

4. tune_outlier_params.py

- **Optional utility script** for optimizing Stage 1 parameters after training:
  - Performs grid search over **percentiles** (distance thresholds) and **EVT cutoffs**.
  - Evaluates TKR, KP, TUR, FKR for each combination.
  - Returns the configuration maximizing the chosen score.
- Can be run **once after training**, with results reused for future evaluations or deployment.

5. outlier_utils.py

- Provides functions for **class-wise statistics**, Mahalanobis distances, per-class thresholds, and EVT modeling:
  - compute_class_stats() – computes class means, covariances, and precision matrices.
  - batch_mahalanobis_sq() – vectorized computation of squared Mahalanobis distances.
  - per_class_thresholds_percentile() – calculates per-class thresholds based on in-class distance percentiles.
  - fit_weibull_per_class() and weibull_outlier_probability() – fits EVT tails and computes outlier probabilities.
- Designed for **Stage 1 unknown detection**, using numpy for stability and optionally torch tensors.
- Supports optional **scipy** and **sklearn** for EVT fitting and covariance regularization.

6. visualize_cm.py

- Loads and visualizes per-class confusion matrices for model performance monitoring:
  - Normalizes rows to represent per-class accuracy.
  - Uses **heatmaps** with annotations via seaborn.
  - Displays evolution of class-wise accuracy across epochs.
- Useful for detecting misclassifications, drift, and imbalanced performance.

7. umap_visualization.py

- Visualizes high-dimensional semantic embeddings in 2D using **UMAP**:
  - Compares **ground-truth labels** vs **model predictions**.
  - Plots known vs unknown samples in separate subplots.
  - Uses tab20 colormap for known classes; unknowns marked in red ‘x’.
- Helps inspect **embedding separability** and diagnose Stage 1 misclassifications.

Workflow

1. **Training**
    - Train the network with train.py for a predefined number of epochs.
    - Save model weights as .pkl checkpoints for evaluation.
2. **Stage 1 Evaluation**
    - Load the trained checkpoint.
    - Extract semantic embeddings for the test dataset.
    - Compute distances to class centers and apply threshold/EVT to reject unknowns.
    - Evaluate performance using TKR, TUR, KP, FKR, and AUROC.
3. **Stage 2 Evaluation**
    - Take Stage 1's rejected samples.
    - Estimate number of unknown classes using clustering on embeddings.
    - Optionally assign pseudo-labels for unknown samples.

Training Details

- Input: RF signal time-frequency representation (reshaped as 2D images).
- Optimizer: Adam with learning rate 1e-4.
- Batch size: 32
- Embedding dimension: 128
- Loss: Combined CE + Center Loss (optionally contrastive).
- Training duration: 20 epochs (more epochs generally improve Stage 1 performance).

Here's an updated version of your **Evaluation Notes** section incorporating your new results and clarifying that the model is now starting to classify known classes (even if poorly) after 2 epochs:

Evaluation Notes

- The current model was trained for **20 epochs** due to time constraints, instead of the planned 251 epochs.
- As a consequence, **Stage 1 parameters** (percentile thresholds and EVT cutoffs) could not be fully optimized.
- The tune_outlier_params script was not run to completion with all epochs and data. Metrics reported, therefore, **underestimate the model's full potential** for open-set detection.
- During early training (first 20 epochs), it was observed that the model **classified most or all samples as unknown**.
    - This behavior is expected in an open-set recognition system that is **conservative by design**:
        - Stage 1 of the classifier is tuned to **minimize false positives for unknown detection**.
        - Until embeddings become sufficiently discriminative, distance-based thresholds (and optional EVT probabilities) may label most test samples as unknown.
    - While this leads to **high AUROC values** (reflecting good potential separation between known and unknown classes), it also results in **low known-class accuracy** early on.

- After **tuning the Stage 1 parameters** and training for **2 epochs** (best we could do with cpu), the classifier is now able to **classify some drones as known**, although accuracy remains low.
    - Metrics show non-zero precision and recall for several classes, indicating that the model is starting to learn discriminative embeddings.
    - Stage 1 conservatism still prevents many samples from being classified as known, but the trend is moving in the expected direction.
- **Future work should include**:
    - Training for the **full number of epochs** (251) to reach better embedding quality.
    - Systematic **Stage 1 parameter tuning** using EVT and percentile thresholds to improve known/unknown separation.
    - **Extended evaluation** to validate Stage 2 clustering and unknown class estimation.

Deployment Considerations

- Modular design: Stage 1 and Stage 2 can run independently.
- Flexible: EVT-based rejection can be enabled/disabled depending on deployment scenario.
- Reproducibility: Semantic embeddings and checkpoints are saved for audit and analysis.
- Performance monitoring: All evaluation metrics and confusion matrices are logged for analysis.
- Unknown detection: Stage 2 allows discovering unknown drone types not seen during training.