**Spectrum Sentinel: Cognitive RF Anomaly Detection**

**🛡️ Project Overview**

**Spectrum Sentinel** is a **hobby project** that mixes **Software-Defined Radio (SDR)** with **Unsupervised Deep Learning** to experiment with real-time, autonomous spectrum awareness. The idea isn’t new, but it’s interesting — and worth exploring for myself.

I’m using a **Nooelec NESDR Smart** as my SDR. It’s a simple, affordable device — not the most capable option compared to high-end SDRs like USRPs or BladeRFs — but it’s perfect for learning. This project is really about understanding the **principles behind Electronic Warfare (EW)**, not building a professional system.

The idea is simple: capture wideband RF data from public bands, convert it into spectrogram slices, and then train an **autoencoder** to spot signals that don’t look like the “normal” background. These anomalies might represent **unknown transmitters**, **weird interference**, or just something new happening in the airwaves.

I’m not trying to reinvent Electronic Warfare or build a deployable SIGINT tool — this is more like a **playground for experimenting** with cognitive radio concepts and seeing how well deep learning can make sense of noisy, real-world spectrum data.

**⚠️ Legal & Ethical Notes**

* Do **not** attempt to decode or record private communications that are illegal to intercept in your country.
* Stick to **public, unencrypted signals**: FM radio, aircraft ADS-B, public telemetry, amateur (ham) bands if permitted, weather stations.
* Always check your local laws before capturing signals.

**🚀 Key Capabilities**

* **Zero-Day Threat Detection:** Flags unknown and novel signals by identifying anomalies in the RF spectrum using reconstruction loss.
* **Blind Signal Classification:** Automatically clusters similar, uncatalogued signals into logical groups, enabling rapid pattern-of-life analysis.
* **Cognitive Architecture:** Built on SDR principles, allowing dynamic re-configuration of the acquisition and processing chain based on real-time ML inference.
* **High-Fidelity Feature Extraction:** Uses the latent space of a Deep Autoencoder to generate robust, low-dimensional features that represent the core signal characteristics, making clustering highly effective.

**🛠️ Technical Architecture**

Three-layer pipeline: **Acquisition → Feature Engineering → Cognitive Processing**

**1. Acquisition Layer (SDR Front-End)**

| **Component** | **Technology** | **Function** |
| --- | --- | --- |
| **RF Front-End** | Nooelec NESDR Smart | Wideband capture of raw I/Q samples from selected frequency bands. |
| **Digitisation** | Internal ADC | Converts RF to digital I/Q streams. |

**2. Feature Engineering Layer**

| **Component** | **Method** | **Function** |
| --- | --- | --- |
| **Time-Frequency Transform** | Short-Time Fourier Transform (STFT) | Converts raw I/Q into 2D spectrogram slices (time × frequency × power). |
| **Normalisation** | Scaling | Ensures stable training of neural networks. |

**3. Cognitive Processing Layer (Unsupervised Deep Learning)**

**A. Autoencoder for Anomaly Detection**

* **Model:** Convolutional Autoencoder (CAE)
* **Training:** On “normal” spectrogram slices only.
* **Mechanism:** Input spectrogram → latent vector (encoder) → reconstructed (decoder).
* **Anomaly Score:** Reconstruction loss . High score = novel/unseen signal.

**B. Clustering for Blind Classification**

* **Input:** Latent vectors from trained encoder.
* **Algorithm:** DBSCAN or K-Means.
* **Function:** Groups signals into distinct, automatically-identified types.

**💻 Step-by-Step Setup & Usage**

**A. Hardware & Software**

**Hardware:**

* Nooelec NESDR Smart (or similar SDR)
* Antenna suitable for the bands you will capture
* PC (Linux preferred; Windows or macOS also possible)

**Software:**

* Python 3.8+
* rtl-sdr, pyrtlsdr, numpy, scipy, matplotlib, tensorflow or torch, scikit-learn, pandas

Install example on Ubuntu:

sudo apt update

sudo apt install -y git build-essential rtl-sdr python3-pip

pip3 install numpy scipy matplotlib pandas scikit-learn tensorflow librosa pyrtlsdr

**B. Bands to Capture**

Focus on **public, legal bands**:

| **Band** | **Frequency** | **Notes** |
| --- | --- | --- |
| FM Radio | 88–108 MHz | Strong broadcast signals |
| ADS-B | 1090 MHz | Aircraft transponders |
| ISM | 433 / 868 MHz | IoT, telemetry |
| Ham (VHF) | 144–146 MHz | Listen only, if licensed |
| Optional | 2.4 GHz ISM | Wi-Fi / Bluetooth (receive only) |

**C. Automated Capture in Python**

from rtlsdr import RtlSdr

import numpy as np

from scipy.signal import stft

import os

# CONFIGURATION

bands = [

{'name': 'fm', 'freq': 100e6, 'duration': 60\*5}, # 5 min

{'name': 'adsb', 'freq': 1090e6, 'duration': 60\*30} # 30 min

]

sample\_rate = 2.048e6

gain = 40

slice\_seconds = 2

nfft = 1024

hop = 256

out\_dir = 'data/spectrograms'

os.makedirs(out\_dir, exist\_ok=True)

sdr = RtlSdr()

for band in bands:

print(f"Capturing {band['name']} at {band['freq']/1e6} MHz")

sdr.sample\_rate = sample\_rate

sdr.center\_freq = band['freq']

sdr.gain = gain

iq\_samples = sdr.read\_samples(int(band['duration']\*sample\_rate))

iq\_samples = iq\_samples.astype(np.complex64)

n\_slice = int(slice\_seconds \* sample\_rate)

num\_slices = len(iq\_samples)//n\_slice

band\_dir = os.path.join(out\_dir, band['name'])

os.makedirs(band\_dir, exist\_ok=True)

for i in range(num\_slices):

slice\_iq = iq\_samples[i\*n\_slice:(i+1)\*n\_slice]

f, t, Zxx = stft(slice\_iq, fs=sample\_rate, nperseg=nfft, noverlap=nfft-hop)

Sxx = np.abs(Zxx)

np.save(os.path.join(band\_dir, f'spec\_{i:04d}.npy'), Sxx)

if i % 10 == 0:

print(f"Saved slice {i}/{num\_slices} for {band['name']}")

sdr.close()

print("Capture complete.")

**Notes:** No modulation setup is required; the CAE learns normal patterns automatically.

**D. CAE Training (Python / Keras example)**

1. Load all normal spectrogram slices for training.
2. Convert to 128×128 greyscale images (or numpy arrays).
3. Define a small Conv Autoencoder, train with MSE loss.

# pseudo-code / simplified

# encoder -> Conv layers -> flatten -> Dense(latent\_dim)

# decoder -> Dense -> reshape -> ConvTranspose -> output 128x128x1

# model.fit(X\_normal, X\_normal, epochs=30, batch\_size=32)

Threshold: mean + 3×std of reconstruction loss on normal slices → flags anomalies.

**E. Clustering**

from sklearn.cluster import DBSCAN, KMeans

# Z = latent vectors from encoder

cl = DBSCAN(eps=0.5, min\_samples=5).fit(Z)

# or

km = KMeans(n\_clusters=8).fit(Z)

**F. Output**

* data/spectrograms/\* → processed slices
* models/ → trained CAE
* results/anomaly\_log.csv → timestamp, frequency, reconstruction loss, cluster ID
* Optional plots: spectrogram, reconstruction, cluster visualisation

**G. Typical Day Plan**

1. 0:00–0:15 — setup environment
2. 0:15–0:45 — quick scan and test RTL-SDR
3. 0:45–2:00 — capture bands with Python automation
4. 2:00–3:30 — slice I/Q → spectrograms
5. 3:30–5:00 — train CAE on normal slices
6. 5:00–6:00 — run inference, flag anomalies, cluster latents

**🤝 Contribution**

This is a fun, personal learning project. If you’re into SDRs, RF or deep learning, feel free to fork, tweak, or experiment. PRs, bug reports, or interesting experiments are welcome!