Analog FM repeater anti piracy

In this article you will learn how to use machine learning for fighting analog FM repeater pirates. If you do not know what a repeater stations is, this is a ham radio station that re-transmits broadcasts from other ham radio operators.

This article describes how to make the AI model to classify transmissions, where pirates are unwanted and ham radio operators are legitimate users. The article does not describe how to interface with the repeater. So you have to add something to the detection script that can (temporarily) switch off the repeater when pirates are detected and then turn it back on after a while. My goal was to keep the repeater as-is, meaning analog, so hams can continue to use cheap radios to work it, but discourage pirates as much as possible in an automated fashion.

To be able to *use* machine learning you will need some computer that can receive the radio signal from the repeater and that is capable of running Python. You will also need some way to control the repeater. If your repeater can be controller via the command line or an API, you should be able to hook this into the detection script. Otherwise you may need to fabricate some computer controlled relay with an Arduino. For receiving the radio signal you can use an RTL-SDR or WebSDR stream if your repeater has one. In the article you will find a script that can be used to record audio, it will record the audio that is played back on the computer. This is the most flexible way, as it allows to record whatever the web browser is playing if you use a WebSDR or what gqrx is playing if you use RTL-SDR. You will need to do some tricks to get start the playback of the WebSDR or gqrx. (ChatGPT is your friend)

To create the machine learning model an i5 laptop without graphics cards will do the trick. Once the model is trained and you have installed the Python libraries no Internet connection is needed for the AI portion.

This article is not an exact step-by-step, it will explain the basics of using Tensorflow (the Python library to do AI work) but you will see a lot of hard-coded paths (/home/bar is my home directory) so expect some trial and error and if you have trouble getting it to work, you can contact me via email, find it on https://www.qrz.com/db/PC1K

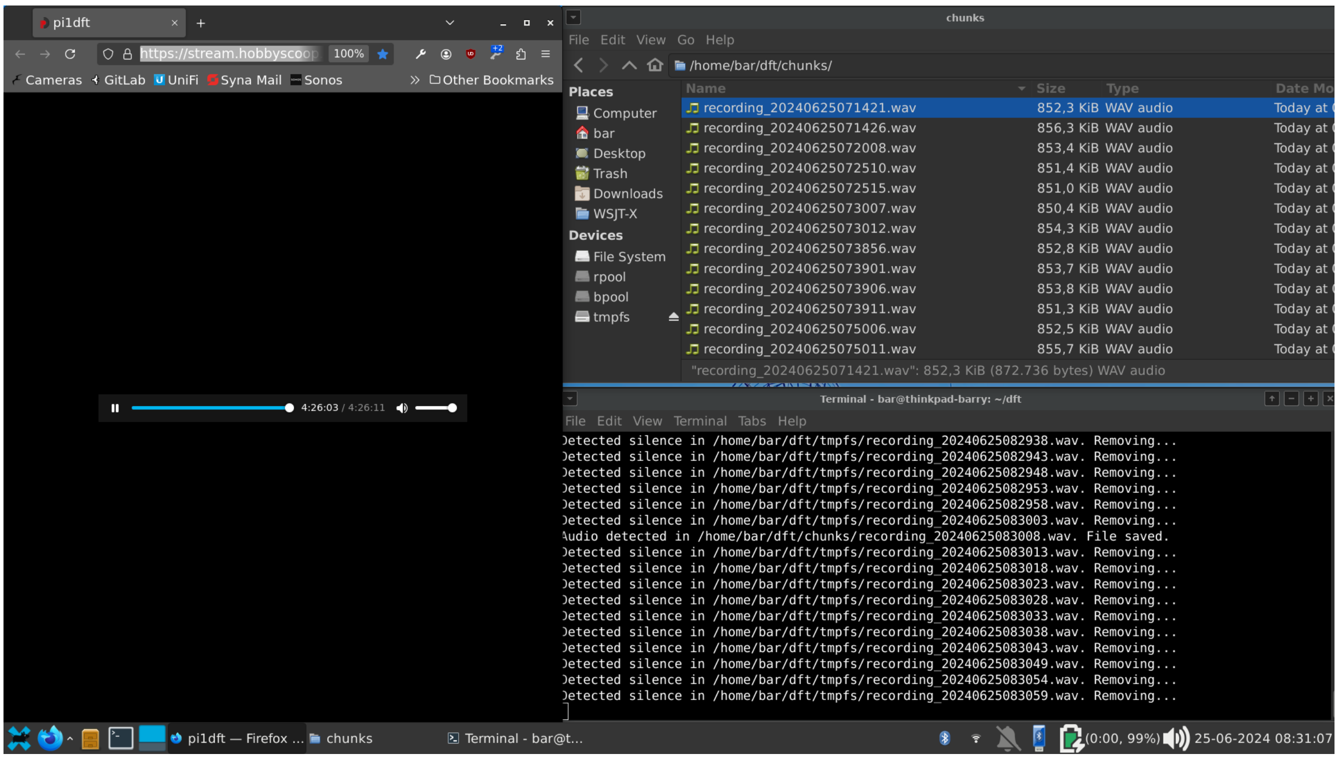
Before executing a script on Linux you need to mark it as executable, example: `chmod +x myscript.sh`. This needs to be done for .sh and .py files.

Step 1: Capturing training data

For training the machine learning model we need audio samples of transmissions made by hams and pirates.

Easiest way of getting these samples is opening a web browser and start the web stream of the repeater, https://stream.hobbyscoop.nl/pi1dft

The following script will be executed and creates a folder in RAM memory (tmpfs) here we are recording 5 seconds chunks of the stream. After 5 seconds the script checks if the recording was only silence or if there was an audio transmission. If so the wav file is moved into a folder for storage on disk.



Name	Size	Type	Date Mo
recording_20240625071421.wav	852,3 KiB	WAV audio	Today at
recording_20240625071426.wav	856,3 KiB	WAV audio	Today at
recording_20240625072008.wav	853,4 KiB	WAV audio	Today at
recording_20240625072510.wav	851,4 KiB	WAV audio	Today at
recording_20240625072515.wav	851,0 KiB	WAV audio	Today at
recording_20240625073007.wav	850,4 KiB	WAV audio	Today at
recording_20240625073012.wav	854,3 KiB	WAV audio	Today at
recording_20240625073856.wav	852,8 KiB	WAV audio	Today at
recording_20240625073901.wav	853,7 KiB	WAV audio	Today at
recording_20240625073906.wav	853,8 KiB	WAV audio	Today at
recording_20240625073911.wav	851,3 KiB	WAV audio	Today at
recording_20240625075006.wav	852,5 KiB	WAV audio	Today at
recording_20240625075011.wav	855,7 KiB	WAV audio	Today at

"recording_20240625071421.wav": 852,3 KiB (872.736 bytes) WAV audio

```
Detected silence in /home/bar/dft/tmpfs/recording_20240625082938.wav. Removing...
Detected silence in /home/bar/dft/tmpfs/recording_20240625082943.wav. Removing...
Detected silence in /home/bar/dft/tmpfs/recording_20240625082948.wav. Removing...
Detected silence in /home/bar/dft/tmpfs/recording_20240625082953.wav. Removing...
Detected silence in /home/bar/dft/tmpfs/recording_20240625082958.wav. Removing...
Detected silence in /home/bar/dft/tmpfs/recording_20240625083003.wav. Removing...
Audio detected in /home/bar/dft/chunks/recording_20240625083008.wav. File saved.
Detected silence in /home/bar/dft/tmpfs/recording_20240625083013.wav. Removing...
Detected silence in /home/bar/dft/tmpfs/recording_20240625083018.wav. Removing...
Detected silence in /home/bar/dft/tmpfs/recording_20240625083023.wav. Removing...
Detected silence in /home/bar/dft/tmpfs/recording_20240625083028.wav. Removing...
Detected silence in /home/bar/dft/tmpfs/recording_20240625083033.wav. Removing...
Detected silence in /home/bar/dft/tmpfs/recording_20240625083038.wav. Removing...
Detected silence in /home/bar/dft/tmpfs/recording_20240625083043.wav. Removing...
Detected silence in /home/bar/dft/tmpfs/recording_20240625083049.wav. Removing...
Detected silence in /home/bar/dft/tmpfs/recording_20240625083054.wav. Removing...
Detected silence in /home/bar/dft/tmpfs/recording_20240625083059.wav. Removing...
```

```
#!/bin/bash

#apt-get install sox
#https://stream.hobbyscoop.nl/pi1dft


# Directory to save recorded audio files
OUTPUT_DIR="/home/bar/dft/chunks"
mkdir -p $OUTPUT_DIR

# Directory to temporarily store files in tmpfs
TMPFS_DIR="/home/bar/dft/tmpfs"
mkdir -p $TMPFS_DIR

if mount | grep "on $TMPFS_DIR type tmpfs" > /dev/null; then
    echo "$TMPFS_DIR is already mounted with tmpfs."
else
    echo "$TMPFS_DIR is not mounted with tmpfs. Mounting now..."
    # Mount the directory with tmpfs
    sudo mount -t tmpfs -o size=100M,mode=0777 tmpfs $TMPFS_DIR
    if [ $? -eq 0 ]; then
        echo "Mounted $TMPFS_DIR with tmpfs successfully."
    else
        echo "Failed to mount $TMPFS_DIR with tmpfs."
        exit 1
    fi
fi


# Function to check if a file contains only silence
is_silence() {
  local file=$1
  local silence=$(sox "$file" -n stat 2>&1 | grep "Maximum amplitude" | awk '{print $3}')
  if (( $(echo "$silence < 0.01" | bc -l) )); then
    return 0 # Silence
  else
    return 1 # Not silence
  fi
}
```

```
while true; do
  TIMESTAMP=$(date +%Y%m%d%H%M%S)
  TEMP_FILE="$TMPFS_DIR/recording_$TIMESTAMP.wav"

  # Record audio in chunks of 5 seconds
  parec -d $(pactl list | grep 'Monitor Source' | head -n 1 | awk '{print $3}') --rate=44100 --channels=2 --file
-format=wav > "$TEMP_FILE" &
  PAREC_PID=$!
  sleep 5
  kill $PAREC_PID

  # Check if the recorded file is silence
  if is_silence "$TEMP_FILE"; then
    echo "Detected silence in $TEMP_FILE. Removing..."
    rm "$TEMP_FILE"
  else
    FINAL_FILE="$OUTPUT_DIR/recording_$TIMESTAMP.wav"
    mv "$TEMP_FILE" "$FINAL_FILE"
    echo "Audio detected in $FINAL_FILE. File saved."
  fi
done
```

Configure OUTPUT_DIR and TMPFS_DIR before use.

Step 2: Manual sorting the training data

This is a tedious process, to ease the process another script is used. This script plays back every wav recording chunk. At the end of each play back the script asks what you want to do with the chunk. Options are:

- Move it into the ham folder
- Move it into the pirate folder
- Delete it

Bonus, since playback is done using mplayer, if you already hear a pirate at 1 second into the playback you can hit enter key to skip the last 4 seconds of playback and go to the options prompt immediately.

I remote mount the laptop running the audio capture and run this script every now and then so I do not get to far behind. Remote mount is needed as playing back audio chunks on the recording laptop, will cause them to be re-recorded resulting in dupes.

Script options, use keyboard key: `h`, `p` or `d` for (h)am, (p)irate, or (d)elete.

```
#!/bin/bash

# sshfs bar@thinkpad-barry:/home/bar/ /media/local1

echo "nolirc=yes" > /home/bar/.mplayer/config
echo "nosub=yes" >> /home/bar/.mplayer/config
echo "noautosub=yes" >> /home/bar/.mplayer/config

# Directories
```

```
SOURCE_DIR="/media/local1/dft/chunks"
HAM_DIR="/media/local1/dft/ham"
PIRATE_DIR="/media/local1/dft/pirate"

# Ensure the target directories exist
mkdir -p "$HAM_DIR"
mkdir -p "$PIRATE_DIR"

# Function to prompt user and get a single character input without pressing Enter
get_char() {
    local char
    IFS= read -r -n1 char
    echo "$char"
}

# Iterate over all wav files in the source directory
for filepath in "$SOURCE_DIR"/*.wav; do
    [ -e "$filepath" ] || continue  # Skip if no wav files are found
    filename=$(basename "$filepath")

    # Play the file using mplayer
    mplayer "$filepath"

    # Prompt the user for input
    echo -n "Move '$filename' to (h)am, (p)irate, or (d)elete? "
    char=$(get_char)

    case $char in
        h)
            mv "$filepath" "$HAM_DIR"
            echo "Moved to ham folder: $filename"
            ;;
        p)
            mv "$filepath" "$PIRATE_DIR"
            echo "Moved to pirate folder: $filename"
            ;;
        d)
            rm "$filepath"
            echo "Deleted file: $filename"
            ;;
        *)
            echo "Invalid input. Skipping file."
            ;;
    esac
done
```

Before using configure SOURCE_DIR, HAM_DIR and PIRATE_DIR

Example output:

```
bar@barry-desktop:~$ sshfs bar@thinkpad-barry:/home/bar/ /media/local1
bar@barry-desktop:~$ cd /media/local1/dft/
bar@barry-desktop:/media/local1/dft$ ./categorize.sh

MPlayer 1.4 (Debian), built with gcc-11 (C) 2000-2019 MPlayer Team

Playing /media/local1/dft/chunks/recording_20240625071421.wav.
libavformat version 58.76.100 (external)
Audio only file format detected.
==========================================================================
Opening audio decoder: [pcm] Uncompressed PCM audio decoder
AUDIO: 44100 Hz, 2 ch, s16le, 1411.2 kbit/100.00% (ratio: 176400->176400)
Selected audio codec: [pcm] afm: pcm (Uncompressed PCM)
==========================================================================
AO: [pulse] 44100Hz 2ch s16le (2 bytes per sample)
Video: no video
Starting playback...
A:   4.0 (04.0) of 4.0 (04.0)  1.1%


Exiting... (End of file)
Move 'recording_20240625071421.wav' to (h)am, (p)irate, or (d)elete? hMoved to ham folder: recording_20240625071421.wav
```

Step 3: Train machine learning (AI) model

This is done using a Python script and TensorFlow, following the guide published at https://medium.com/@oluyaled/audio-classification-using-deep-learning-and-tensorflow-a-step-by-step-guide-5327467ee9ab

Model training took about 45 minutes when used with 477 pirate wav files and 1446 ham wav files on an i5 laptop with no GPU.

Move the wav recording that have been manually categorized into the folder `/home/bar/dft/train` in this folder there should be a folder `ham` and a folder `pirate`. Then run the script. If using different folder name update the data_dir variable.

```
#!/bin/python3
import os
import librosa
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.image import resize
from tensorflow.keras.models import load_model

# Define your folder structure
data_dir = '/home/bar/dft/train'
classes = ['ham', 'pirate']

# Load and preprocess audio data
def load_and_preprocess_data(data_dir, classes, target_shape=(128, 128)):
    print('Load and preprocess data')
    data = []
    labels = []

    for i, class_name in enumerate(classes):
        class_dir = os.path.join(data_dir, class_name)
        for filename in os.listdir(class_dir):
            if filename.endswith('.wav'):
                file_path = os.path.join(class_dir, filename)
                audio_data, sample_rate = librosa.load(file_path, sr=None)
                # Perform preprocessing (e.g., convert to Mel spectrogram and resize)
                mel_spectrogram = librosa.feature.melspectrogram(y=audio_data, sr=sample_rate)
                mel_spectrogram = resize(np.expand_dims(mel_spectrogram, axis=-1), target_shape)
                data.append(mel_spectrogram)
                labels.append(i)

    return np.array(data), np.array(labels)

print('Split training and test data')
# Split data into training and testing sets
data, labels = load_and_preprocess_data(data_dir, classes)
labels = to_categorical(labels, num_classes=len(classes))  # Convert labels to one-hot encoding
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)

# Create a neural network model
input_shape = X_train[0].shape
input_layer = Input(shape=input_shape)
x = Conv2D(32, (3, 3), activation='relu')(input_layer)
x = MaxPooling2D((2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu')(x)
x = MaxPooling2D((2, 2))(x)
x = Flatten()(x)
x = Dense(64, activation='relu')(x)
output_layer = Dense(len(classes), activation='softmax')(x)
model = Model(input_layer, output_layer)

# Compile the model
print('Compile model')
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
print('Train model')
model.fit(X_train, y_train, epochs=200, batch_size=32, validation_data=(X_test, y_test))

print('Test accuracy')
test_accuracy=model.evaluate(X_test,y_test,verbose=0)
print(test_accuracy[1])

# Save the model
model.save('audio_classification_model.keras')
```

Step 4: Manual test the model on new data

For this I created a `test.py` script that can be executed as follows:

```
./test.py chunks/recording_20240627085410.wav
```

This recording is of me calling `Pappa Charly One Kilo Test`, the script provides the following prediction:

```
Class: ham, Probability: 1.0000
Class: pirate, Probability: 0.0000
The audio is classified as: ham
Accuracy: 1.0000
```

Contents of the test script:

```
#!/bin/python3
import sys

# load_model_sample.py
from keras.models import load_model
from keras.preprocessing import image
import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf
import librosa
from tensorflow.image import resize

# Load the saved model
model = load_model('audio_classification_model.keras')

# Define the target shape for input spectrograms
target_shape = (128, 128)

# Define your class labels
classes = ['ham', 'pirate']

# Function to preprocess and classify an audio file
def test_audio(file_path, model):
    # Load and preprocess the audio file
    audio_data, sample_rate = librosa.load(file_path, sr=None)
    mel_spectrogram = librosa.feature.melspectrogram(y=audio_data, sr=sample_rate)
    mel_spectrogram = resize(np.expand_dims(mel_spectrogram, axis=-1), target_shape)
    mel_spectrogram = tf.reshape(mel_spectrogram, (1,) + target_shape + (1,))

    # Make predictions
    predictions = model.predict(mel_spectrogram)

    # Get the class probabilities
    class_probabilities = predictions[0]

    # Get the predicted class index
    predicted_class_index = np.argmax(class_probabilities)
    return class_probabilities, predicted_class_index

# Test an audio file, passed as argument
test_audio_file = sys.argv[1]
class_probabilities, predicted_class_index = test_audio(test_audio_file, model)

# Display results for all classes
for i, class_label in enumerate(classes):
    probability = class_probabilities[i]
    print(f'Class: {class_label}, Probability: {probability:.4f}')

# Calculate and display the predicted class and accuracy
predicted_class = classes[predicted_class_index]
accuracy = class_probabilities[predicted_class_index]
print(f'The audio is classified as: {predicted_class}')
print(f'Accuracy: {accuracy:.4f}')
```

Step 5: Use the model for generate more training data

We have validated the model manually, but to build confidence in it, the next step is to use the model to sort new transmissions. This is a nice intermediate step between manually sorting the data and using the AI to control the repeater when pirates are detected.

So in this step we will update the recording script to call the AI, and move the recordings into the `ham` and `pirate` folders. We will then manually verify if the sorting was correct. And if needed make corrections and re-train the model. And repeat the process untill the model is accurate.

The `predict.py` script is an improved version of `test.py` that includes code to move the wav file into the `ham` and `pirate` folders.

```
#!/bin/python3
import sys

# load_model_sample.py
from keras.models import load_model
from keras.preprocessing import image
import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf
import librosa
from tensorflow.image import resize
import shutil

def move_file(destination_subfolder):
    # Step 1: Get the file path from sys.argv[1]
    source_file_path = sys.argv[1]
    # Step 2: Define the base destination directory
    base_destination_directory = os.path.expanduser("/home/bar/dft")

    # Step 3: Define the full destination directory based on the subfolder argument
    destination_directory = os.path.join(base_destination_directory, destination_subfolder)

    # Step 4: Ensure the destination directory exists
    os.makedirs(destination_directory, exist_ok=True)

    # Step 5: Construct the full path for the destination file
    destination_file_path = os.path.join(destination_directory, os.path.basename(source_file_path))

    # Step 6: Move the file
    shutil.move(source_file_path, destination_file_path)
    print(f"File moved to {destination_file_path}")


# Load the saved model
model = load_model('audio_classification_model.keras')

# Define the target shape for input spectrograms
target_shape = (128, 128)

# Define your class labels
classes = ['ham', 'pirate']

# Function to preprocess and classify an audio file
def test_audio(file_path, model):
    # Load and preprocess the audio file
    audio_data, sample_rate = librosa.load(file_path, sr=None)
    mel_spectrogram = librosa.feature.melspectrogram(y=audio_data, sr=sample_rate)
    mel_spectrogram = resize(np.expand_dims(mel_spectrogram, axis=-1), target_shape)
    mel_spectrogram = tf.reshape(mel_spectrogram, (1,) + target_shape + (1,))

    # Make predictions
    predictions = model.predict(mel_spectrogram)

    # Get the class probabilities
    class_probabilities = predictions[0]

    # Get the predicted class index
    predicted_class_index = np.argmax(class_probabilities)

    return class_probabilities, predicted_class_index

# Test an audio file, passed as argument
test_audio_file = sys.argv[1]
class_probabilities, predicted_class_index = test_audio(test_audio_file, model)

# Display results for all classes
for i, class_label in enumerate(classes):
    probability = class_probabilities[i]
    print(f'Class: {class_label}, Probability: {probability:.4f}')

# Calculate and display the predicted class and accuracy
predicted_class = classes[predicted_class_index]
accuracy = class_probabilities[predicted_class_index]
```

```
print(f'The audio is classified as: {predicted_class}')
print(f'Accuracy: {accuracy:.4f}')

move_file(predicted_class)
```

We add predict.py to the record.sh for the filtering as follows:

```
/home/bar/dft/predict.py "$FINAL_FILE" &
```

So the complete record.sh script now looks like this:

```
#!/bin/bash

#apt-get install sox
#https://stream.hobbyscoop.nl/pi1dft


# Directory to save recorded audio files
OUTPUT_DIR="/home/bar/dft/chunks"
mkdir -p $OUTPUT_DIR

# Directory to temporarily store files in tmpfs
TMPFS_DIR="/home/bar/dft/tmpfs"
mkdir -p $TMPFS_DIR

if mount | grep "on $TMPFS_DIR type tmpfs" > /dev/null; then
    echo "$TMPFS_DIR is already mounted with tmpfs."
else
    echo "$TMPFS_DIR is not mounted with tmpfs. Mounting now..."
    # Mount the directory with tmpfs
    sudo mount -t tmpfs -o size=100M,mode=0777 tmpfs $TMPFS_DIR
    if [ $? -eq 0 ]; then
        echo "Mounted $TMPFS_DIR with tmpfs successfully."
    else
        echo "Failed to mount $TMPFS_DIR with tmpfs."
        exit 1
    fi
fi


# Function to check if a file contains only silence
is_silence() {
  local file=$1
  local silence=$(sox "$file" -n stat 2>&1 | grep "Maximum amplitude" | awk '{print $3}')
  if (( $(echo "$silence < 0.01" | bc -l) )); then
    return 0 # Silence
  else
    return 1 # Not silence
  fi
}

while true; do
  TIMESTAMP=$(date +%Y%m%d%H%M%S)
  TEMP_FILE="$TMPFS_DIR/recording_$TIMESTAMP.wav"

  # Record audio in chunks of 5 seconds
  parec -d $(pactl list | grep 'Monitor Source' | head -n 1 | awk '{print $3}') --rate=44100 --channels=2 --file
-format=wav > "$TEMP_FILE" &
  PAREC_PID=$!
  sleep 5
  kill $PAREC_PID
  # Check if the recorded file is silence
  if is_silence "$TEMP_FILE"; then
    echo "Detected silence in $TEMP_FILE. Removing..."
    rm "$TEMP_FILE"
  else
    FINAL_FILE="$OUTPUT_DIR/recording_$TIMESTAMP.wav"
    mv "$TEMP_FILE" "$FINAL_FILE"
    echo "Audio detected in $FINAL_FILE. File saved."
    /home/bar/dft/predict.py "$FINAL_FILE" &
  fi
done
```

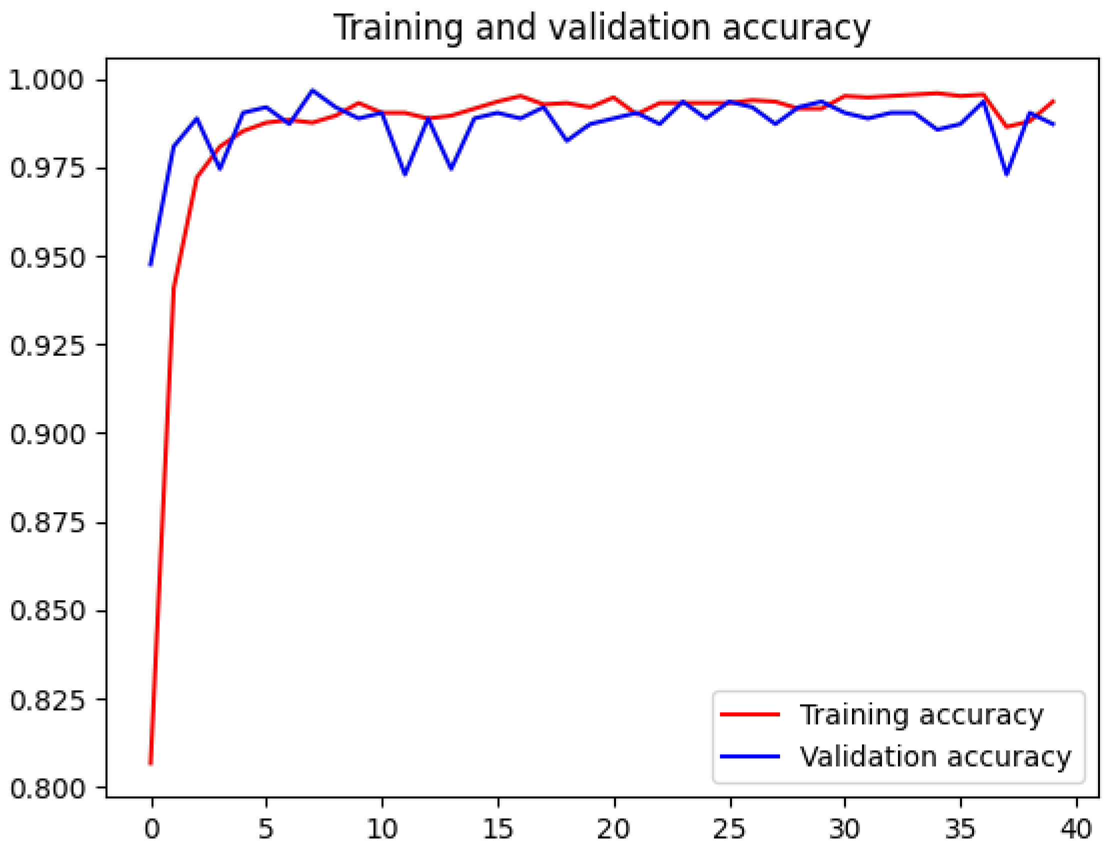
Update model training for overfitting prevention and early stopping

Based on the first couple of iterations of model training, the `train.py` script was improved in the following ways:

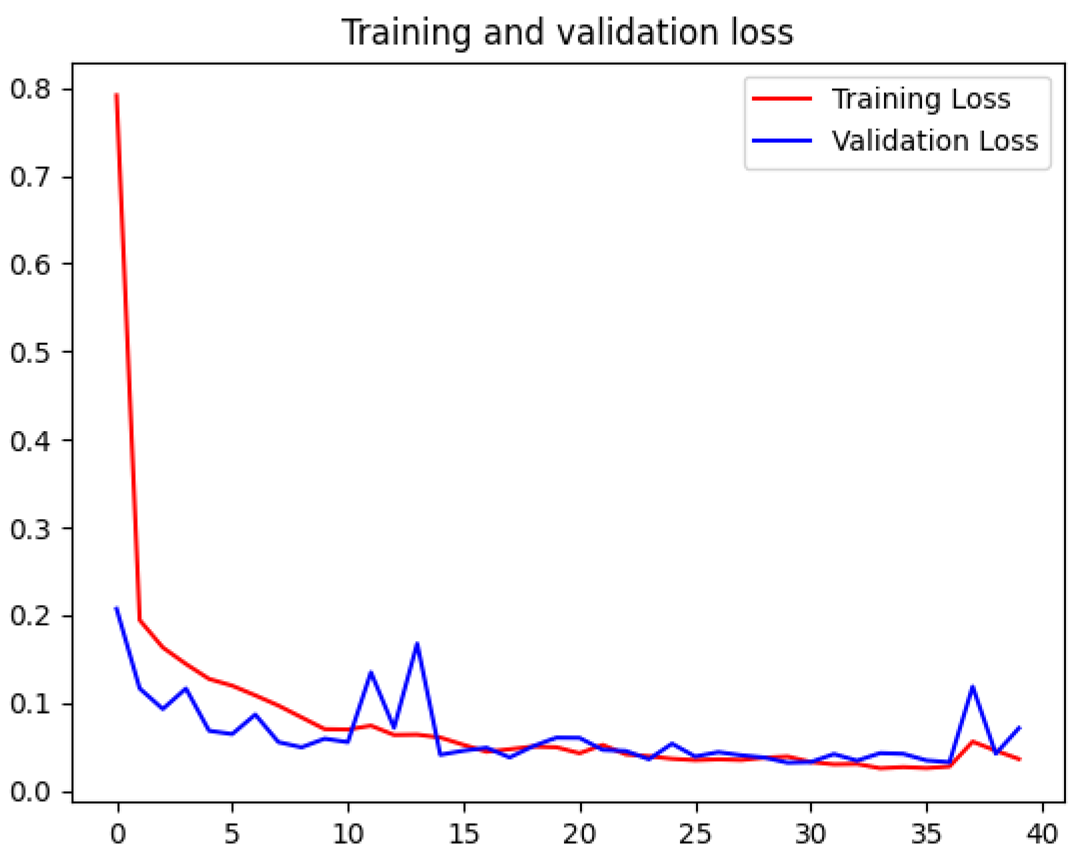
- Dropout Layers: Added after each pooling layer and before the output layer to randomly set a fraction of the input units to 0 during training.
- L2 Regularization: Added to the Dense layer to penalize large weights.
- Early Stopping: Monitors the validation loss and stops training when it stops improving, helping to avoid overfitting.

In addition the validation and loss of the training are displayed in a graph, if running the script remotely over SSH, you will need to use X forwarding to see the graphs. For example: `ssh -XC user@mymachine.pc1k.nl`.

The early stopping reduces the time to train the model to about 6 minutes, instead of 45 minutes, while achieving an accuracy of 99%.



Accuracy, higher is better.



Loss, lower is better.

Here is the updated `train.py`:

```
#!/bin/python3
import os
import librosa
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.image import resize
from tensorflow.keras.models import load_model
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt

# Define your folder structure
data_dir = '/home/bar/dft/train'
classes = ['ham', 'pirate']

# Load and preprocess audio data
def load_and_preprocess_data(data_dir, classes, target_shape=(128, 128)):
    print('Load and preprocess data')
    data = []
    labels = []

    for i, class_name in enumerate(classes):
        class_dir = os.path.join(data_dir, class_name)
        for filename in os.listdir(class_dir):
            if filename.endswith('.wav'):
                file_path = os.path.join(class_dir, filename)
                audio_data, sample_rate = librosa.load(file_path, sr=None)
                # Perform preprocessing (e.g., convert to Mel spectrogram and resize)
                mel_spectrogram = librosa.feature.melspectrogram(y=audio_data, sr=sample_rate)
                mel_spectrogram = resize(np.expand_dims(mel_spectrogram, axis=-1), target_shape)
                data.append(mel_spectrogram)
                labels.append(i)

    return np.array(data), np.array(labels)

print('Split training and test data')
# Split data into training and testing sets
data, labels = load_and_preprocess_data(data_dir, classes)
labels = to_categorical(labels, num_classes=len(classes))  # Convert labels to one-hot encoding
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)

# Create a neural network model
input_shape = X_train[0].shape
input_layer = Input(shape=input_shape)
x = Conv2D(32, (3, 3), activation='relu')(input_layer)
x = MaxPooling2D((2, 2))(x)
x = Dropout(0.25)(x)

x = Conv2D(64, (3, 3), activation='relu')(x)
x = MaxPooling2D((2, 2))(x)
x = Dropout(0.25)(x)

x = Flatten()(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.5)(x)

output_layer = Dense(len(classes), activation='softmax', kernel_regularizer=l2(0.01))(x)
model = Model(input_layer, output_layer)

# Compile the model
print('Compile model')
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

# Early Stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)


# Train the model
print('Train model')
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test),
callbacks=[early_stopping])

print('Test accuracy')
test_accuracy=model.evaluate(X_test,y_test,verbose=0)
print(test_accuracy[1])

# Save the model
model.save('audio_classification_model.keras')

# Plot the chart for accuracy and loss on both training and validation
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()

plt.plot(epochs, loss, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

Step 6 Using the model

Setting up Python

To make sure the model works reliable, it is advised to use Python in a virtual environment. This will also make it survive OS upgrades. I download a Python tgz from https://www.python.org/ftp/python/ and extract it into `/opt/Python-X.XX.XX` folder. Then I create a virtual environment as follows:

```
mkdir /opt/tensorflow
virtualenv --python="/opt/Python-3.10.11/python" "/opt/tensorflow"
cd /opt/tensorflow
source bin/activate
pip install tensorflow scikit-learn pandas matplotlib seaborn librosa
```

Then to run the prediction script, I create a `action.sh` wrapper with the following content:

```
#!/bin/bash

cd /opt/tensorflow/
source bin/activate
python /opt/tensorflow/action.py $1
```

I can use the `action.sh` wrapper to automatically run whenever we receive a new wav recording. The prediction takes around 3 seconds.

Using the model

Now that we have validated the model as much as possible it is time to implement it. For this we make a copy of the `test.py` and update it to perform an action if a pirate is detected with >99.7% accuracy. The code used to make this distinction is:

```
accuracy_string = f'{accuracy:.4f}'

if predicted_class == 'pirate':
  if '0.99' in accuracy_string or '0.98' in accuracy_string or '0.97' in accuracy_string or '1.' in accuracy_string:
    # Your code to execute when accuracy is greater than 99.6
    print("This is a pirate with at least 99.7% accuracy")
```

Arguably there should be a way in Python to work with the percentage directly and not do it via string comparison, but I have not figured that one out yet.

The complete `action.py` is defined as follows:

```
#!/bin/python3
import sys

# load_model_sample.py
from keras.models import load_model
from keras.preprocessing import image
import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf
import librosa
from tensorflow.image import resize

# Load the saved model
model = load_model('audio_classification_model.keras')

# Define the target shape for input spectrograms
target_shape = (128, 128)

# Define your class labels
classes = ['ham', 'pirate']

# Function to preprocess and classify an audio file
def test_audio(file_path, model):
    # Load and preprocess the audio file
    audio_data, sample_rate = librosa.load(file_path, sr=None)
    mel_spectrogram = librosa.feature.melspectrogram(y=audio_data, sr=sample_rate)
    mel_spectrogram = resize(np.expand_dims(mel_spectrogram, axis=-1), target_shape)
    mel_spectrogram = tf.reshape(mel_spectrogram, (1,) + target_shape + (1,))

    # Make predictions
    predictions = model.predict(mel_spectrogram)

    # Get the class probabilities
    class_probabilities = predictions[0]

    # Get the predicted class index
    predicted_class_index = np.argmax(class_probabilities)

    return class_probabilities, predicted_class_index

# Test an audio file, passed as argument
test_audio_file = sys.argv[1]
class_probabilities, predicted_class_index = test_audio(test_audio_file, model)

# Display results for all classes
for i, class_label in enumerate(classes):
    probability = class_probabilities[i]
    print(f'Class: {class_label}, Probability: {probability:.4f}')

# Calculate and display the predicted class and accuracy
predicted_class = classes[predicted_class_index]
accuracy = class_probabilities[predicted_class_index]
print(f'The audio is classified as: {predicted_class}')
print(f'Accuracy: {accuracy:.4f}')
```

```
accuracy_string = f'{accuracy:.4f}'

if predicted_class == 'pirate':
  if '0.99' in accuracy_string or '0.98' in accuracy_string or '0.97' in accuracy_string or '1.' in accuracy_string:
    # Your code to execute when accuracy is greater than 99.6
    print("This is a pirate with at least 99.7% accuracy")
```

Bonus: Mel Spectrogram

In the blog post https://medium.com/@oluyaled/audio-classification-using-deep-learning-and-tensorflow-a-step-by-step-guide-5327467ee9ab we can read the following:

Instead of using raw audio data, we convert it into a Mel spectrogram. A Mel spectrogram is a visual representation of audio data that's easier for a neural network to process. We create these spectrograms using librosa.

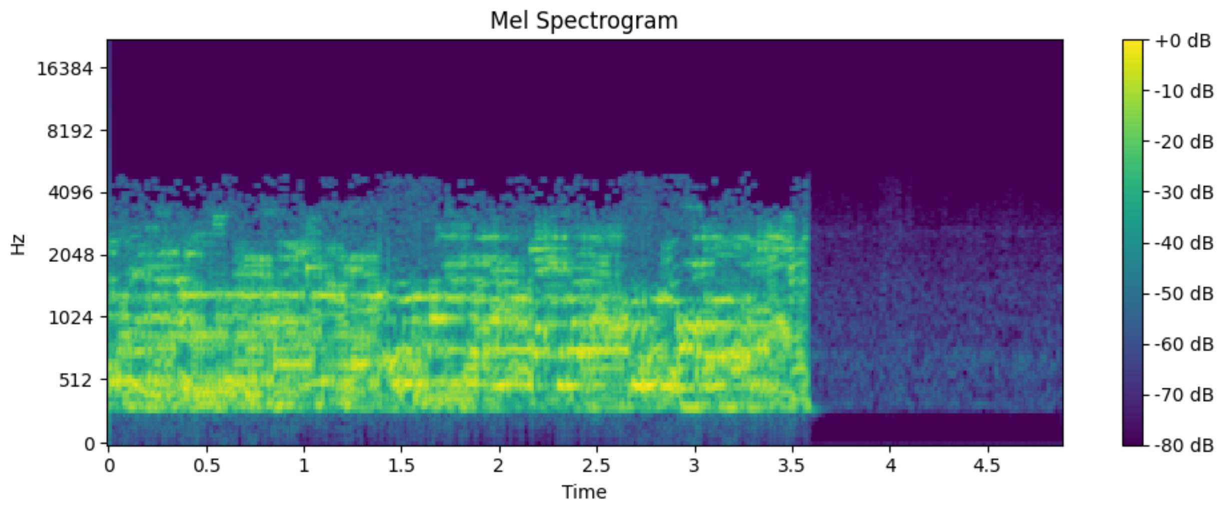
Let's see what a Mel Spectrogram looks like, so we get an idea of what the machine model is learning with. For this we change the test.py code to output the spectrogram on the screen.

```
def test_audio(file_path, model):
    # Load and preprocess the audio file
    audio_data, sample_rate = librosa.load(file_path, sr=None)
    mel_spectrogram = librosa.feature.melspectrogram(y=audio_data, sr=sample_rate)

  Bonus code to show spectrogram

    # Convert to dB (log scale)
    S_dB = librosa.power_to_db(mel_spectrogram, ref=np.max)

    # Plot the mel spectrogram using matplotlib
    plt.figure(figsize=(10, 4))
    librosa.display.specshow(S_dB, sr=sample_rate, x_axis='time', y_axis='mel', cmap='viridis')
    plt.colorbar(format='%+2.0f dB')
    plt.title('Mel Spectrogram')
    plt.tight_layout()
    plt.show()
```



Here we listened to a pirate playing some music, as you can see the music only lasted for 3.5 seconds followed by silence.

Links

It is not on Github, cause I do no like MS, here are some links:

- https://pc1k.nl/repeater-ai/Analog%20FM%20repeater%20anti%20piracy.pdf
- https://pc1k.nl/repeater-ai/action.py
- https://pc1k.nl/repeater-ai/analyze.sh
- https://pc1k.nl/repeater-ai/audio_classification_model_v7.keras
- https://pc1k.nl/repeater-ai/categorize.sh
- https://pc1k.nl/repeater-ai/predict.py
- https://pc1k.nl/repeater-ai/record.sh
- https://pc1k.nl/repeater-ai/test.py
- https://pc1k.nl/repeater-ai/train.py
- https://pc1k.nl/repeater-ai/recordings/