CHAPTER 5
PROJECT DESCRIPTION

5.1 About the project

This project is designed to be an addition to the currently available home and vehicle security systems specifically focused to reduce and prevent armed violence and provide immediate notification and support to the owners and the concerned legal authorities. The current status Quo indicates that there has been an increase in the crimes involving guns over the past few years. With domestically produced guns available through backdoor sellers, it becomes important to ensure preventive measures against such scenarios.

Our project is a deep-learning based sound detection and classification system focused on detecting and isolating sounds of gunshot and gunfire from the other ambient environmental sounds. The sound model would consist of an image classifier which would take a sound sample spectrogram as an input and classify it among ambient and dangerous (gunfire) sounds. For any positive identification of a gunfire, the current location of the recording device would be sent to the pre-registered contacts and to the nearest police stations.

This would ensure that fast and efficient action can be taken against such crimes which, at present, go unnoticed for most cases. As soon as a single gunshot is detected, the real time location can be shared, to provide medical aid, and to ensure no further armed conflict takes place.

The main project structure includes a recorder which captures audio at fixed intervals via a microphone device. This audio is then trimmed and normalized to reduce the file size and reduce background noise. The cleaned audio is then processed into time sections where the higher frequencies are attenuated, and the audio is split into time specific sub audio files. These audio files are then sent to LibROSA python library for conversion into an audio spectrum graph which is then fed as input to the neural network. The network classifies the spectrum as either an environmental sound or a gunshot sound.

The model could be packaged as an API to be used alongside other mobile applications, as a standalone application, a web-based application, of using a low-cost hardware recorder and a microchip to be used in vehicles. For software-based solutions, the built-in device

recorders can be utilized, along with periodic sound detection and classification, using a low usage sound-spectrogram script and pre-trained model.

The current accuracy of the model, with over 4000 ambient sounds stands at and average of 99.74%, which are viable practical results.

5.2 Technology Stack

Python – Python 3.x would be used as the primary coding language where multiple python functionalities like class operations, functions, dictionaries and other features will be utilized for building the project. Python being a versatile scripting language provides robust environment for deep learning, tensor manipulation and API inclusion. It can also be run in the form of code snipped on low powered machines like Raspberry Pie.

Fastai - The fastai library is a PyTorch wrapper library which enables training fast and accurate neural nets using modern best practices. It's based on research in to deep learning best practices undertaken at fast.ai, including "out of the box" support for vision, text, tabular, and collab (collaborative filtering) models. We will build our classification model using this library.

Librosa - LibROSA is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems. The primary usage of LibROSA in this project is for the conversion of the audio into spectrograms which are then passed on to the classification model.

5.2.1 PyTorch [3]

PyTorch is a python based library built to provide flexibility as a deep learning development platform. The workflow of PyTorch is as close as you can get to python's scientific computing library – numpy.

- Easy to use API – It is as simple as python can be.
- Python support – As mentioned above, PyTorch smoothly integrates with the python data science stack. It is so similar to numpy that you might not even notice the difference.
- Dynamic computation graphs – Instead of predefined graphs with specific functionalities, PyTorch provides a framework for us to build computational graphs

as we go, and even change them during runtime. This is valuable for situations where we don't know how much memory is going to be required for creating a neural network.

- A few other advantages of using PyTorch are its multi GPU support, custom data loaders and simplified preprocessors.

PyTorch uses an imperative / eager paradigm. That is, each line of code required to build a graph defines a component of that graph. We can independently perform computations on these components itself, even before your graph is built completely. This is called "define-by-run" methodology.

(a) PyTorch Tensors

Tensors are nothing but multidimensional arrays. Tensors in PyTorch are similar to numpy's ndarrays, with the addition being that Tensors can also be used on a GPU.

As with numpy, it is very crucial that a scientific computing library has efficient implementations of mathematical functions. PyTorch gives you a similar interface, with more than 200+ mathematical operations you can use.

(b) Autograd module

PyTorch uses a technique called automatic differentiation. That is, we have a recorder that records what operations we have performed, and then it replays it backward to compute our gradients. This technique is especially powerful when building neural networks, as we save time on one epoch by calculating differentiation of the parameters at the forward pass itself.

(c) Optim module

Torch Optim is a module that implements various optimization algorithms used for building neural networks. Most of the commonly used methods are already supported, so that we don't have to build them from scratch (unless you want to!).

(d) nn module

PyTorch autograd makes it easy to define computational graphs and take gradients, but raw autograd can be a bit too low-level for defining complex neural networks. This is where the

nn module can help.

The nn package defines a set of modules, which we can think of as a neural network layer that produces output from input and may have some trainable weights.

5.2.2 librosa [2]

1. **librosa.beat**

   Functions for estimating tempo and detecting beat events.

2. **librosa.core**

   Core functionality includes functions to load audio from disk, compute various spectrogram representations, and a variety of commonly used tools for music analysis. For convenience, all functionality in this sub module is directly accessible from the top-level librosa.* namespace.

3. **librosa.decompose**

   Functions for harmonic-percussive source separation (HPSS) and generic spectrogram decomposition using matrix decomposition methods implemented in scikit-learn.

4. **librosa.display**

   Visualization and display routines using matplotlib

5. **librosa.effects**

   Time-domain audio processing, such as pitch shifting and time stretching. This sub module also provides time-domain wrappers for the decompose sub module.

6. **librosa.feature**

   Feature extraction and manipulation. This includes low-level feature extraction, such as chromagrams, pseudo-constant-Q (log-frequency) transforms, Mel spectrogram, MFCC, and tuning estimation. Also provided are feature manipulation methods, such as delta features, memory embedding, and event-synchronous feature alignment.

7. **librosa.filters**

   Filter-bank generation (chroma, pseudo-CQT, CQT, etc.). These are primarily internal functions used by other parts of librosa.

8. **librosa.onset**

Onset detection and onset strength computation

9. **librosa.output**

   Text- and wav-file output (Deprecated)

10. **librosa.segment**

    Functions useful for structural segmentation, such as recurrence matrix construction, time-lag representation, and sequentially constrained clustering

11. **librosa.sequence**

    Functions for sequential modeling. Various forms of Viterbi decoding, and helper functions for constructing transition matrices

12. **librosa.util**

    Helper utilities (normalization, padding, centering, etc.)

5.2.3 Fastai [1]

(a) Training modules overview

The fastai library structures its training process around the Learner class, whose object binds together a PyTorch model, a dataset, an optimizer, and a loss function; the entire learner object then will allow us to launch training.

basic_train defines this Learner class, along with the wrapper around the PyTorch optimizer that the library uses. It defines the basic training loop that is used each time you call the fit method (or one of its variants) in fastai. This training loop is very bare-bones and has very few lines of codes; you can customize it by supplying an optional Callback argument to the fit method.

callback defines the Callback class and the CallbackHandler class that is responsible for the communication between the training loop and the Callback's methods.

The CallbackHandler maintains a state dictionary able to provide each Callback object all the information of the training loop it belongs to, putting any imaginable tweaks of the training loop within your reach.

callbacks implements each predefined Callback class of the fastai library in a separate

module. Some modules deal with scheduling the hyper parameters, like all backs.one_cycle, callbacks.lr_finder and callback.general_sched. Others allow special kinds of training like callbacks.fp16 (mixed precision) and callbacks.rnn.

The Recorder and callbacks.hooks are useful to save some internal data generated in the training loop.

Train then uses these callbacks to implement useful helper functions. Lastly, metrics contains all the functions and classes you might want to use to evaluate your training results; simpler metrics are implemented as functions while more complicated ones as subclasses of Callback. For more details on implementing metrics as Callback, please refer to creating your own metrics.

(b) Application fields

The fastai library allows you to train a Model on a certain DataBunch very easily by binding them together inside a Learner object. This module regroups the tools the library provides to help you preprocess and group your data in this format.

1. collab

   This sub module handles the collaborative filtering problems.

2. tabular

   This sub-package deals with tabular (or structured) data.

3. text

   This sub-package contains everything you need for Natural Language Processing.

4. vision

   This sub-package contains the classes that deal with Computer Vision.

5. Module structure

   In each case (except for collab), the module is organized this way:

6. transform

   This sub-module deals with the pre-processing (data augmentation for images, cleaning for tabular data, tokenizing and numericalizing for text).

7. data

   This sub-module defines the dataset class (es) to deal with this kind of data.

8. models

   This sub-module defines the specific models used for this kind of data.

9. learner

   When it exists, this sub-module contains functions that will directly bind this data with a suitable model and add the necessary callbacks.

[*Note – This is a summarization provided of the libraries functionality taken from the documentation. For complete information, kindly refer to the official library documentation]

5.3 Project Blueprint

The project is divided into stages which will be followed in incremental order throughout the project duration. Since this project involves some hardware integration, an early hardware-based prototype has to be considered as future work; however a software simulation for the working of the project can be created for display purpose.

- At the initial stages of the project, we will collect data of various gunshot sounds from multiple sources. These sounds will include single round shots, multiple firing sounds, and reloading sounds and burst fire sounds. Since no such dataset is available, we will build our dataset using manual data collection from web scraping, audio extraction from relevant videos, gun sound imitations and direct audio downloads.
- These audio files would then be cleaned from background or white noise, converted into single channel time restricted audio samples, separated in time intervals and normalized from peak-to-peak for the purpose of spectrogram generation.
- Since the project would be used alongside ambient environmental sounds, it is also important to collect audio samples of ambient noises and to differentiate between the gun sounds and the latent environmental sounds. These samples can be extracted from the ESC-50 dataset which have various prerecorded single channel ambient audio samples.

- The sound spectrograms would then be generated on the fly using LibROSA which will be fed to the classifier for class outputs. These spectrograms are generated on a bit by bit basis and utilize multiple audio samplings to create accurate and sample specific images.
- The generated images will be used by the classifier model built using fastai and Residual Networks to classify between Gunshot and Non-Gunshot sounds and to distinguish between ambient noise and outliers.
- After attaining a decent accuracy, we can then use the classifier for the test cases by using, as inputs, real time sounds at defined periods to detect armed crimes and gunshots.
- This built prediction model can be used as a modular backend for safety application or paired with recording devices and a GPS tag for use in homes and vehicles. During a time of positive detection, the location of scene can be sent to trusted authorities automatically without delay.

5.4 Use Cases

As an API - We can run the independent prediction model on a server where different mobile and web application can send requests with their image spectrogram and the model will return results in a convenient JSON or XML format indicating the predicted result.

As a module – For faster processing a precompiled and stored model can be used as an add-on module for mobile application, computer systems or smart devices for direct prediction and result analysis.

Paired with hardware – As stated before, the primary task for this project is to ensure a reduction in violent crimes involving guns. Hence paired with appropriated audio recording hardware, we can install a low-cost safety system in vehicles which detect these sounds on the streets and roads, where most of these crimes take place.

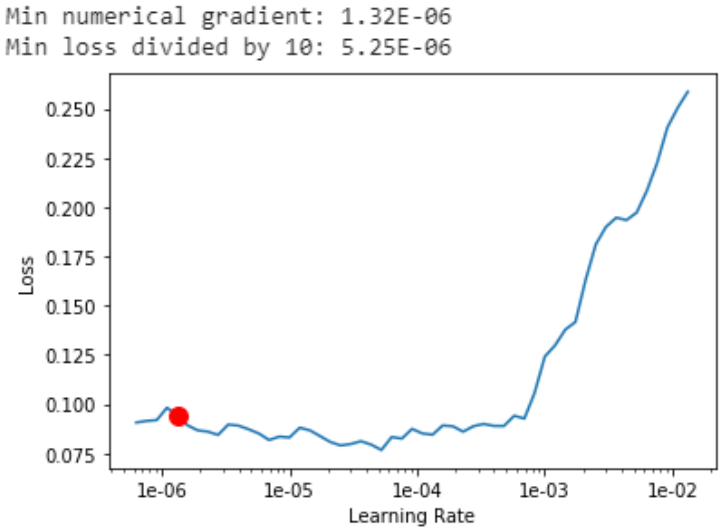
5.5 Pre-existing Work in the field

The performance of Sound Classification like ESC-50 Classification, Environmental Sound Classification, etc. has been significantly improved with the evolution of Deep Neural Networks (DNN). Models which use Convolutional Neural Networks (CNN) for classification of sound based on the spectrograms obtained for different sound samples

have been proposed regarding the given images. These along with the other model can be used for separation of various image spectrograms.

5.6 Outputs and Discussion

Learning Rate vs. Loss Graph [*Fig 5.1*]:



Accuracy and Epoch Graph [*Fig 5.2*]:

epoch	train_loss	valid_loss	accuracy	time
0	0.498642	0.127842	0.967370	05:44
1	0.297514	0.066820	0.971209	00:38
2	0.193615	0.080352	0.965451	00:38
3	0.134032	0.058392	0.975048	00:38

(a) Stage 1

epoch	train_loss	valid_loss	accuracy	time
0	0.081352	0.054738	0.971209	00:50
1	0.072155	0.044250	0.984645	00:50

(b) Stage 2

epoch	train_loss	valid_loss	accuracy	time
0	0.049927	0.039738	0.984645	00:50
1	0.061518	0.035547	0.986564	00:50
2	0.056182	0.033683	0.986564	00:50
3	0.056638	0.033492	0.986564	00:50

(c) Stage 3

epoch	train_loss	valid_loss	accuracy	time
0	0.047696	0.032697	0.984645	00:50
1	0.061321	0.027519	0.986564	00:50
2	0.059754	0.025358	0.988484	00:50
3	0.053487	0.025141	0.986564	00:50

(d) Stage 4

epoch	train_loss	valid_loss	accuracy	time
0	0.070634	0.027704	0.986564	00:50
1	0.054947	0.027253	0.988484	00:50
2	0.046620	0.023687	0.992322	00:50

(e) Stage 5

epoch	train_loss	valid_loss	accuracy	time
0	0.031165	0.016169	0.994403	00:50
1	0.032670	0.018078	0.994242	00:50
2	0.028532	0.014768	0.995403	00:50

(f) Stage 6.1

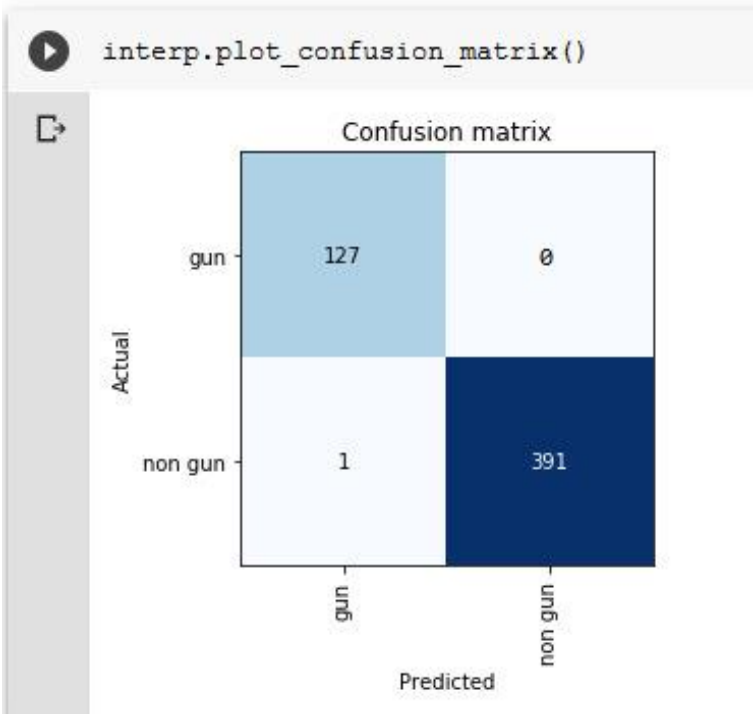
epoch	train_loss	valid_loss	accuracy	time
0	0.027511	0.014902	0.995742	00:50
1	0.021778	0.011828	0.996042	00:50
2	0.023910	0.011618	0.996842	00:50

(g) Stage 6.2

epoch	train_loss	valid_loss	accuracy	time
0	0.019044	0.011357	0.997442	00:50
1	0.018898	0.011309	0.997482	00:50

(h) Stage 7

Confusion Matrix [*Fig 5.3*]:



Class Prediction [*Fig 5.4***]:**

```
[ ]  data.classes

     ['gun', 'non gun']
```

As we can infer from the outputs, we attain a maximum accuracy of **99.74% with a learning rate of 1e-6,** we then predicted a number of test images with high accuracy. As can be seen from the confusion matrix, the correct prediction for a gunshot was 127 out of 128 times, with a false negative of 1 over the entire set. Similarly, the detection of non-gun sounds was very accurate with no false positives recorded.

5.7 Sample Inputs

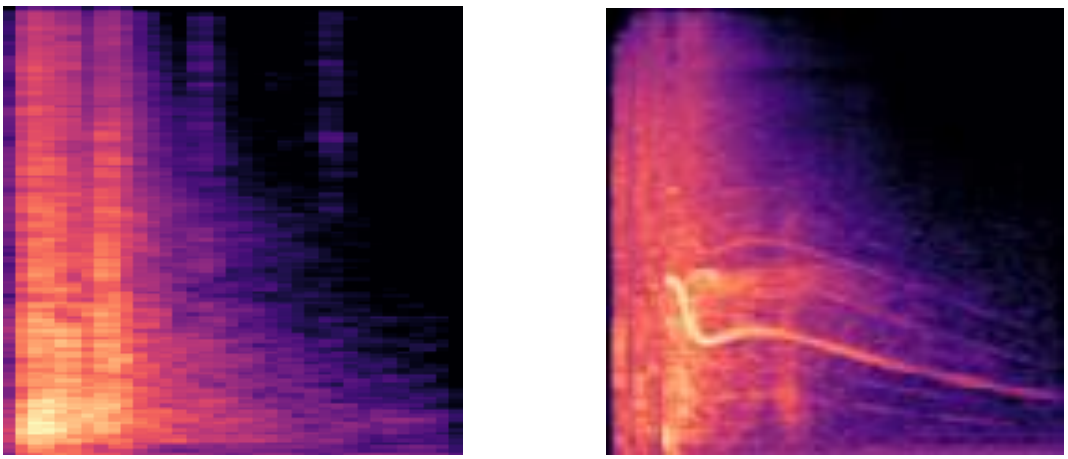


Fig 5.5 Gunshot Sounds as Input

The gunshot spectrograms were created from various gun classes which have been used in the dataset like ak-47, m4a4 etc. The dataset was collected from various sources and was cleaned, compiled and normalized by us. The gunshot sounds were primarily available of various FPS gaming systems, whose audio files we extracted. Apart from them, online open source audio samples were taken from multiple sites and then checked for quality and consistency with the current dataset. If the new audio files were considered feasible, they were normalized, split into convenient sub-audio sets and converted into spectrograms using librosa.

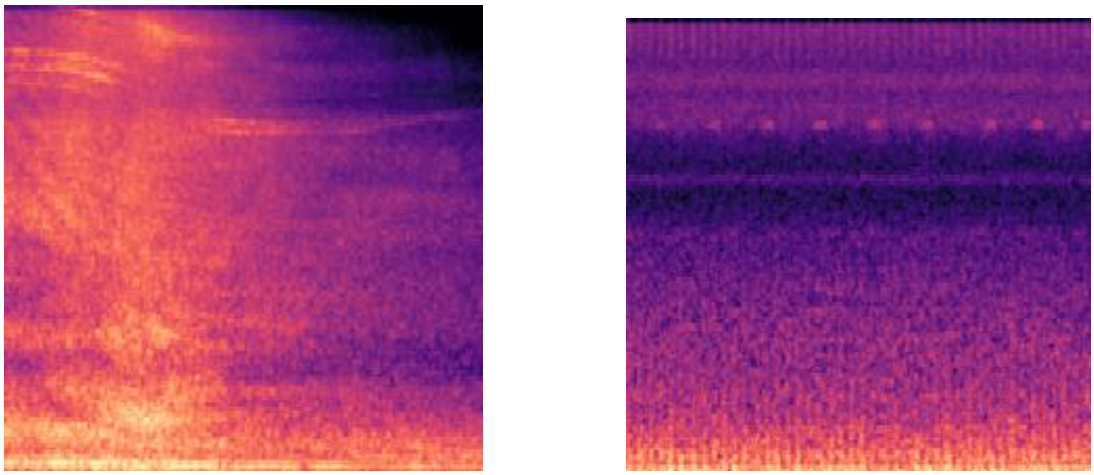


Fig 5.6 Environmental Sounds as Input

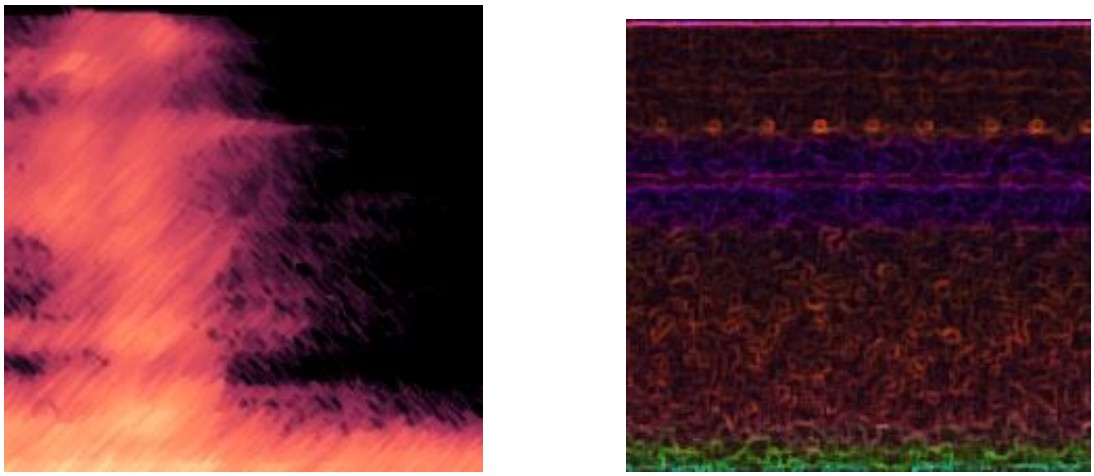


Fig 5.7 Color Channel Highlights

As we can see from the channel highlights, there are clear spikes in the RGB spectrums which are used by the classifier to generate the relevant image attributes. Filtering out unused noise gives us a clear and limited channel image from which it is easy to see if the image contains a gunshot or was an environmental image.

Images in this dataset have been physically removed from open field chronicles assembled by the Freesound.org venture. The dataset has been prearranged into folds for tantamount cross-approval, ensuring that sections from a similar unique source document are contained in a solitary overlay. These images were pre organized and normalized and hence required little to no normalization of their own. The images were just resized and grouped together to be used in the model.

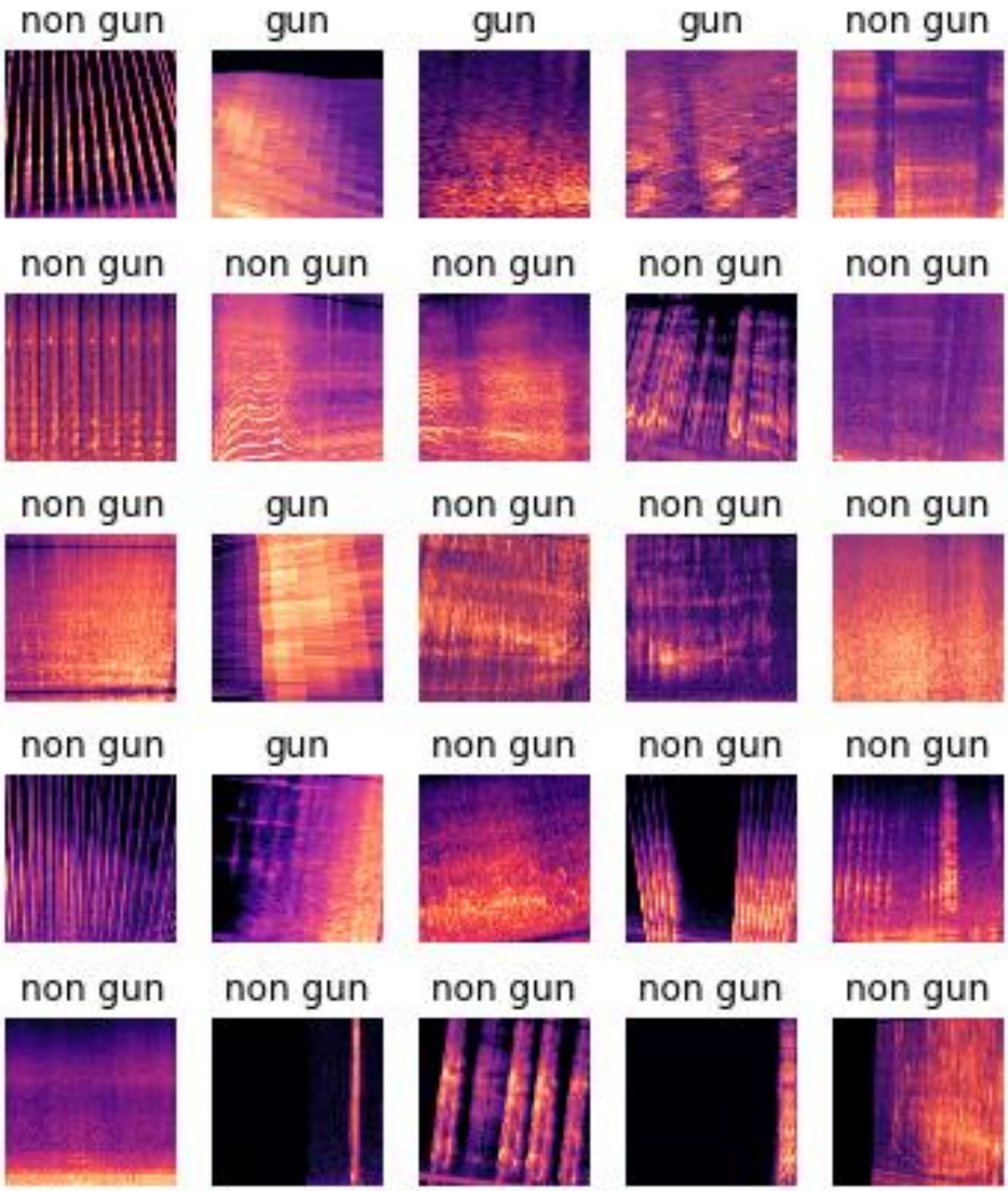


Fig 5.8 Gun/No-Gun Classification

5.8 The Environmental Dataset [4]

The ESC-50 dataset is a labelled collection of 2000 environmental audio recordings suitable for benchmarking methods of environmental sound classification.

The dataset consists of 5-second-long recordings organized into 50 semantical classes (with 40 examples per class) loosely arranged into 5 major categories:

Animals	Natural soundscapes & water sounds	Human, non-speech sounds	Interior/domestic sounds	Exterior/urban noises
Dog	Rain	Crying baby	Door knock	Helicopter
Rooster	Sea waves	Sneezing	Mouse click	Chainsaw
Pig	Crackling fire	Clapping	Keyboard typing	Siren
Cow	Crickets	Breathing	Door, wood creaks	Car horn
Frog	Chirping birds	Coughing	Can opening	Engine
Cat	Water drops	Footsteps	Washing machine	Train
Hen	Wind	Laughing	Vacuum cleaner	Church bells
Insects (flying)	Pouring water	Brushing teeth	Clock alarm	Airplane
Sheep	Toilet flush	Snoring	Clock tick	Fireworks
Crow	Thunderstorm	Drinking, sipping	Glass breaking	Hand saw

Fig 5.9 Different sounds present in environment data

Clips in this dataset have been manually extracted from public field recordings gathered by the Freesound.org project. The dataset has been prearranged into 5 folds for comparable cross-validation, making sure that fragments from the same original source file are contained in a single fold.

5.9 Future Work

We plan on including other audio phrase detection which is commonly used during a crime scene like ‘Help’, ‘Catch’, ‘Gun’, and ‘Ambulance’ etc., to further spread the scope of the project beyond the current capabilities and to increase the model accuracy even higher and test under practical situations.