UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

ÁLEFE FELIPE GONÇALVES PEREIRA DIAS

NICOLAS ABRIL

DETECÇÃO DE DISPAROS DE ARMAS DE FOGO EM UM AMBIENTE URBANO USANDO REDES CONVOLUCIONAIS EM UMA REDE DE SISTEMAS EMBARCADOS

CURITIBA

2022

ÁLEFE FELIPE GONÇALVES PEREIRA DIAS
NICOLAS ABRIL

DETECÇÃO DE DISPAROS DE ARMAS DE FOGO EM UM AMBIENTE URBANO USANDO REDES CONVOLUCIONAIS EM UMA REDE DE SISTEMAS EMBARCADOS

Gunshot detection using convolutional networks on an embedded systems network in a urban environment

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Marcelo de Oliveira Rosa

CURITIBA

2022

ÁLEFE FELIPE GONÇALVES PEREIRA DIAS
NICOLAS ABRIL

DETECÇÃO DE DISPAROS DE ARMAS DE FOGO EM UM AMBIENTE URBANO USANDO REDES CONVOLUCIONAIS EM UMA REDE DE SISTEMAS EMBARCADOS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 24/05/2022

Denise de Oliveira Carneiro Berejuk
Mestre em Engenharia Biomédica
Secretaria de Segurança Pública e Administração Penitenciária do Estado do Paraná

Amauri Amorin Assef
Doutor em Engenharia Elétrica e Informática Industrial
Universidade Tecnológica Federal do Paraná

João Alberto Fabro
Doutor em Engenharia Elétrica e Informática Industrial
Universidade Tecnológica Federal do Paraná

CURITIBA
2022

AGRADECIMENTOS

Agradecemos aos professores do curso de Engenharia de Computação, que nos forneceram conhecimento e motivação para sempre fazermos o melhor. Agradecimento em especial ao LASER [1], na pessoa do professor João Alberto Fabro, que possibilitou um aprendizado mais aprofundado em sistemas embarcados e inteligência computacional antes de se ingressar no mercado de trabalho, além do uso de competições de robótica como preparo para situações de estresse, onde se é necessário resolver problemas de forma eficiente e urgente.

[1] Laboratório Avançado de Sistemas Embarcados e Robótica

RESUMO

Este trabalha relata o desenvolvimento de um sistema de detecção de eventos sonoros de armas de fogo, para ser utilizado em um contexto de cidades inteligentes. Foi criada uma aplicação embarcada capaz de detectar sons de disparos de arma de fogo em tempo real, alertando um servidor central e se comunicando através de uma rede *mesh* Wi-SUN, de forma que seria possível replicar o dispositivo e utilizar uma rede de sensores para localizar a origem do evento. Para a detecção, foi utilizado um modelo baseado em redes neurais convolucionais que alcançou uma acurácia de 97% executando em um computador desktop e 88% após conversão do modelo para o dispositivo embarcado. O projeto foi desenvolvido em parceria com uma empresa, que forneceu recursos para o desenvolvimento e a estrutura de rede a ser utilizada.

Palavras-chave: detecção de eventos sonoros; segurança pública; cidades inteligentes; redes neurais convolucionais; wi-sun.

ABSTRACT

This work describes the creation of a sound event detection system for firearms, to be used in a smart cities context. An embedded application was created that is capable of detecting gunshots in real time, alerts a central server and communicates through a Wi-SUN mesh network, so that it would be possible to replicate the device and use a sensor network to identify the location of the event origin. For the detection, a convolutional neural network model was used that achieved 97% accuracy when running on a desktop computer and 88% after converting the model to work with the embedded device. This project was done in collaboration with a company, that provided development resources and the structure of the network that was used.

Keywords: sound event detection; public safety; smart cities; convolutional neural networks; wi-sun.

LISTA DE FIGURAS

Figura 1 – Representação de uma gravação do som de disparo de arma de fogo. Os números a esquerda mostram a intensidade da amostra (onde ± 1.0 é o pico do microfone) e os acima o tempo em segundos. A figura mostra um forte pico inicial, com duração em torno de 10ms, uma região onde se concentra a maior parte da energia, que dura em torno de 100ms, e um decaimento longo. 18

Figura 2 – Representação no espectro da frequência do som da Figura 1. Os números a esquerda são a escala de frequências em Hertz e os acima o tempo em segundos. As cores vermelhas e brancas indicam maior intensidade daquela frequência naquele instante de tempo, enquanto o cinza e azul representam menor intensidade. 19

Figura 3 – Visão geral do sistema 26

Figura 4 – Visualização de um modelo Rede Neural Convolucional, do inglês *Convolutional Neural Network* (CNN) 2D 33

Figura 5 – Fluxograma do *firmware* 41

Figura 6 – Dashboard servidor 42

LISTA DE FOTOGRAFIAS

Fotografia 1 – Dispositivo ShotSpotter instalado em um poste na cidade de Oakland, Estados Unidos 20
Fotografia 2 – Ao lado esquerdo, um exemplo de border router. Ao lado direito, um exemplo de node 27

LISTA DE QUADROS

Quadro 1 – Matriz de confusão para detecção *offline* do modelo A 43
Quadro 2 – Matriz de confusão para detecção *offline* do modelo B 43
Quadro 3 – Matriz de confusão para detecção offline do modelo C 44
Quadro 4 – Comparação entre os 3 melhores modelos 44

LISTA DE ABREVIATURAS E SIGLAS

Siglas

API	Interface de Programação de Aplicação, do inglês *Application Programming Interface*
AWS	*Amazon Web Services*
BLE	Bluetooth de Baixa Energia, do inglês *Bluetooth Low Energy*
BSP	Pacote de Suporte à Placa, do inglês *Board Support Package*
CMSIS	Padronização das interfaces de software comuns de microcontroladores, do inglês *Common Microcontroller Software Interface Standard*
CNN	Rede Neural Convolucional, do inglês *Convolutional Neural Network*
CoAP	Protocolo de aplicação restrita, do inglês *Constrained Application Protocol*
DMA	Acesso Direto à Memória, do inglês *Direct Memory Access*
FOTA	*Firmware* Transmitido por Ar, do inglês *Firmware Over The Air*
GPIO	Entrada/Saída de Propósito Geral, do inglês *General-Purpose Input/Output*
HTTP	Protocolo de Transferência de Hipertexto, do inglês *HyperText Transfer Protocol*
HTTPS	HTTPS *Secure*
I2S	*Inter-IC Sound*
IoT	Internet das Coisas, do inglês *Internet of Things*
MFCC	*Mel-Frequency Cepstral Coefficients*
PCM	Modulação por Código de Pulso, do inglês *Pulse-Code Modulation*
PDM	Modulação por Densidade de Pulso, do inglês *Pulse-Density Modulation*
QSPI	Interface Serial Periférica Quadruplica, do inglês *Quad Serial Peripheral Interface*
RPL	Protocolo de Roteamento para Redes de Baixa Potência e com Perdas, do inglês *Routing Protocol for Low-Power and Lossy Networks*
RTOS	Sistema Operacional de Tempo Real, do inglês *Real-Time Operating System*

SAI	Interface de Áudio Serial, do inglês *Serial Audio Interface*
SDIO	Entrada/Saída Digital Segura, do inglês *Secure Digital Input/Output*
SELD	Detecção e Localização de Eventos Sonoros, do inglês *Sound Event Localization and Detection*
SPI	Interface Serial Periférica, do inglês *Serial Peripheral Interface*
UART	Transmissor/Receptor Assíncrono Iniversal, do inglês *Universal Asynchronous Receiver/Transmitter*
UDP	Protocolo de Datagrama de Usuário, do inglês *User Datagram Protocol*
Wi-SUN	Rede Ubíqua Inteligente sem Fio, do inglês *Wireless Smart Ubiquitous Network*

SUMÁRIO

1 INTRODUÇÃO 14

1.1 Considerações iniciais 14

1.2 Objetivos 14

1.2.1 Objetivo geral 14

1.2.2 Objetivos específicos 14

1.3 Justificativa 15

1.4 Estrutura do trabalho 15

2 REFERENCIAL TEÓRICO 16

2.1 Domínio do Problema 16

2.1.1 SELD 16

2.1.2 Análise Espectrográfica e Temporal 17

2.1.3 Som em Ambientes urbanos 17

2.1.4 Sons balísticos 18

2.1.5 Cidades inteligentes e IoT 19

2.2 Soluções Existentes 20

2.3 Tecnologias Utilizadas 21

2.3.1 Sensores acústicos 21

2.3.2 Protocolos de comunicação com periféricos 22

2.3.3 Mbed 23

2.3.4 Wi-SUN 23

2.3.5 CoAP 23

2.3.6 Node-RED 24

2.3.7 CNN 24

3 DESENVOLVIMENTO 25

3.1 Especificação 25

3.1.1 Modelagem de Software 25

3.1.2 Modelagem de *Hardware* 26

3.2 Detecção de Eventos 28

3.2.1 Base de dados 28

3.2.2 Processamento de Áudio 29

3.2.3 Modelo de Detecção 30
3.2.4 Treinamento e Validação 31
3.3 Implementação do sistema embarcado 32
3.3.1 Sistema operacional 34
3.3.2 Arquitetura do *software* embarcado 34
3.3.3 Aquisição de áudio 35
3.3.4 Processamento de áudio 37
3.3.5 Detecção 38
3.4 Rede de comunicação 38
3.5 Sistema de Alerta 39
4 RESULTADOS 43
4.1 Detecção offline 43
4.2 Detecção online 44
4.3 Comunicação 45
5 CONCLUSÃO 46
5.1 Evolução e Desenvolvimentos Futuros 46
REFERÊNCIAS 48

1 INTRODUÇÃO

1.1 Considerações iniciais

Em 2019, o Brasil registrou 45503 homicídios, dos quais 30825, ou 67,7%, foram por arma de fogo (CERQUEIRA *et al.*, 2021). Isso representa uma diminuição 21,5% em relação a 2018 mas, ao mesmo tempo, a proporção de mortes violentas por causa indeterminada nesse período aumentou 189%, o que pode indicar entre outras coisas uma piora da investigação das causas dessas mortes.

Um outro estudo estimou que em 2015 apenas 20,7% dos homicídios dolosos no Brasil geram denúncias criminais (MORIN *et al.*, 2017). Isso indica uma baixa efetividade da investigação no Brasil, já que esse número decorre principalmente de não ser possível determinar o autor do crime após a investigação. Ribeiro e Lima (RIBEIRO; LIMA, 2018) apontam a falta de dados, em especial de um sistema unificado para o Brasil, como um dos agravantes do problema.

Com isso, um sistema de detecção de sons balísticos em tempo real pode ser útil para a segurança pública, em três aspectos principais: resposta em curto prazo, evidência para júri, e construção de uma base de dados que permite com que melhores análises sejam feitas no decorrer do tempo. A popularização dos sensores para cidades inteligentes abre portas para diversas aplicações usando Internet das Coisas. As redes existentes costumam ser projetadas para serem escaláveis e facilmente estendidas, possibilitando que outros sensores se conectem a uma rede já existente e consigam acesso à internet sem ter um *hardware* específico para isso. Por essas razões, decidiu-se criar um sistema que se conecte a uma rede para cidades inteligentes e detecte eventos balísticos.

1.2 Objetivos

1.2.1 Objetivo geral

Desenvolver um sistema de detecção de disparos de armas de fogo em ambientes urbanos.

1.2.2 Objetivos específicos

- Classificação e detecção *onboard* em tempo real em um sistema embarcado;
- Uso de rede convolucional para detecção e classificação do som;
- Uso de uma estrutura de rede *mesh* já existente para cidades inteligentes;
- Envio de alertas a um servidor remoto através da Internet.

1.3 Justificativa

Esse projeto tem o proposito de servir como um protótipo de uma aplicação real voltada para cidades inteligentes. Com isso, não será focado em obter os melhores resultados possíveis, mas sim em mostrar que é um projeto viável. Também não será focado no desenvolvimento da camada de rede, já que a ideia é usar uma estrutura de rede já pronta para dispositivos voltado para cidades inteligentes. Além disso, por se tratar de um sistema embarcado inteligente, é uma ótima aplicabilidade dos conhecimentos adquiridos durante o curso de Engenharia de Computação, principalmente das áreas de sistemas embarcados, sistemas operacionais, sistemas inteligentes, banco de dados, processamento digital de sinais e engenharia de *software*.

1.4 Estrutura do trabalho

O Capítulo 2 descreve características do problema, assim como detalha as tecnologias que foram utilizadas na solução proposta, comparando-a com outras soluções que já existem para esse mesmo problema ou problemas análogos.

O Capítulo 3 apresenta informações sobre como o projeto foi desenvolvido, explicando as escolhas feitas, os detalhes de implementação e as principais dificuldades encontradas em todas as fases do desenvolvimento, desde a modelagem do sistema, a implementação do algoritmo *offline*, a implementação no sistema embarcado e a integração de todas as partes.

O Capítulo 4 apresenta os resultados dos testes realizados com o sistema, tanto da detecção *offline* quando da detecção em tempo real com o dispositivo embarcado, abordando questões como desempenho dos modelos, tempo de processamento e escalabilidade.

Finalmente, o Capítulo 5 traz conclusões obtidas com a realização do trabalho, analisando o resultado final em comparação com objetivos do trabalho, além de trazer prospectos de futuros caminhos a serem tomados, expondo possíveis desenvolvimentos futuros que expandam a solução proposta.

2 REFERENCIAL TEÓRICO

2.1 Domínio do Problema

2.1.1 SELD

Detecção e Localização de Eventos Sonoros, do inglês *Sound Event Localization and Detection* (SELD) é a habilidade de, dado o conjunto de dados captados por um ou mais sensores acústicos, discernir se ocorreu algum som de interesse, extraindo informações como quando aconteceu o evento, de onde ele foi originado, que tipo de objeto causou o evento ou uma classificação do tipo de som (WANG *et al.*, 2021). É uma das habilidades fundamentais para a sobrevivência de aves e mamíferos, percebendo onde estão presas, predadores ou companheiros, e é a base da comunicação oral entre pessoas.

O SELD pode ser decomposto em suas duas partes: a detecção e a localização. A tarefa de detecção consiste basicamente de buscar um conjunto de características no som que identifique o evento, encontrando um início e um fim da ocorrências delas. Essas características podem ser, entre outras coisas, certas frequências que ocorrem em uma sequência esperada, um devido perfil da intensidade do som, alguma diferença características entre os sons obtidos por sensores localizados em posições diferentes, ou uma mistura complexa de todos esses itens. Essas características podem ser buscadas diretamente no som, fazendo por exemplo um casamento entre as frequências medidas e as esperadas, ou indiretamente, como é o caso da solução apresentada no presente texto, em que os dados dos sensores são agregados e alimentam um modelo de rede neural convolucional. Como parte da detecção pode haver ou não a classificação do evento em uma ou mais categorias.

Para a localização, existem várias formas de realizá-la. A mais simples é calcular a defasagem entre os sons captados por sensores com uma distância conhecida entre eles. Tendo um par de sensores em um plano bidimensional e sabendo a diferença de tempo da chegada dos eventos sonoros entre os sensores, é possível encontrar uma curva com as posições possíveis para a origem do evento. Com três sensores, é possível agregar informação de cada par e calcular o ponto exato da fonte sonora. No entanto, como a precisão não é infinita, na realidade se obtém uma região da provável localização, que pode ser reduzida aumentando o número de sensores envolvidas no cálculo. A mesma ideia poderia ser utilizada para localização em 3 dimensões, mas para esse trabalho isso não era necessário.

Existem outras formas de calcular localização, como usando diferenças na magnitude e na fase do espectro de som obtido, diferenças na amplitude do som e em distorções do perfil de frequências que podem indicar uma direção de origem ou distância do sensor, mas são todas técnicas que exigem um conhecimento mais aprofundado dos sensores usados, do ambiente

e muitas vezes exigem um modelo treinado especificamente para uma configuração fixa de sensores, portanto foram descartadas para esse projeto.

2.1.2 Análise Espectrográfica e Temporal

As informações extraídas do sinal acústico ocorrem basicamente de duas formas. A primeira consiste em analisar o som em função do tempo e a outra em analisar as componentes em frequência (espectro) presentes nesse som. As duas formas costumam trazer informações diferentes sobre um mesmo sinal (GONZALEZ, 2013).

Na análise temporal, é possível obter informações como a energia de um trecho de som, a intensidade dos picos e a relação entre os momentos acústicos mais intensos e os mais silenciosos e o número de vezes que o sinal cruza o eixo de magnitude zero, invertendo a polaridade (taxa de cruzamento por zeros). Já na análise do espectro de frequências, é possível identificar o timbre dos sons presentes em um áudio, se existe alguma frequência e harmônicos dominantes, a quantidade de ruído ou é possível usar o espectro diretamente para tentar reconhecer a fonte de um som (LI *et al.*, 2000).

Essas duas análises se complementam e usando as duas é possível obter um melhor resultado na detecção e categorização de eventos sonoros (GONZALEZ, 2013).

2.1.3 Som em Ambientes urbanos

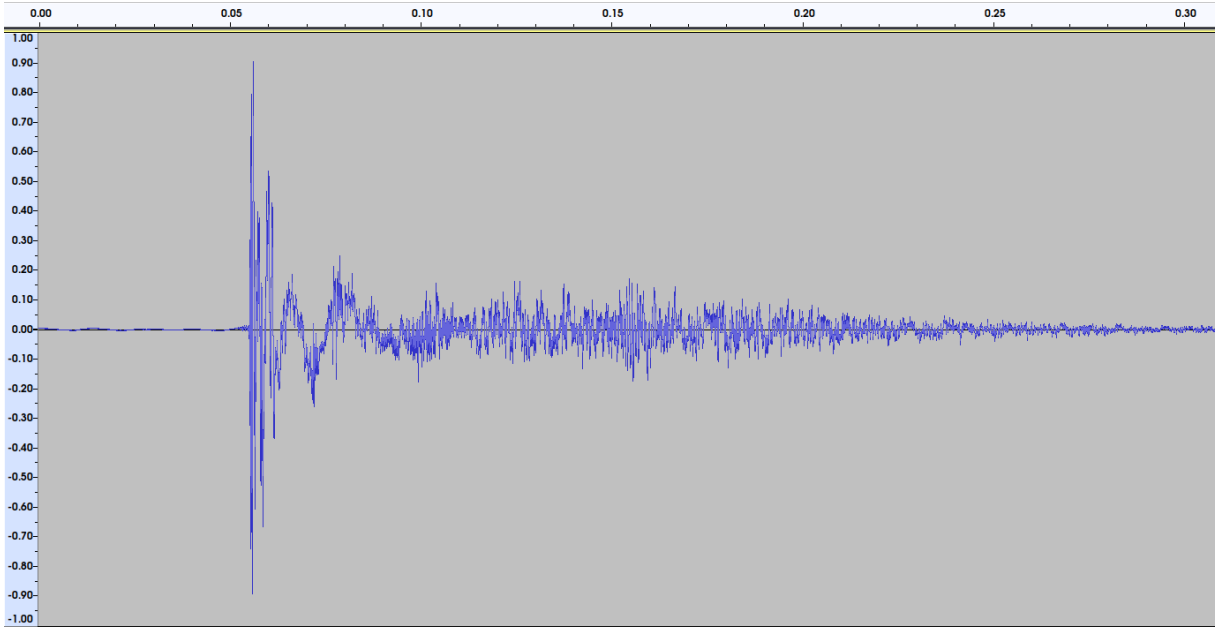
Comparado a ambientes fechados e controlados e ambientes totalmente abertos, os ambientes urbanos são bastante desafiadores, principalmente pela sua heterogeneidade. A principal característica de interesse dos ambientes urbanos é a grande variedade de ruídos diferentes e de grande intensidade, criados pelos veículos, pessoas e outros elementos presentes. Para o problema tratado nesse trabalho, são especialmente relevantes sons explosivos, como os oriundos de escapamentos de carros, palmas, rojões e trovões, que se mostram na literatura serem comumente confundidos com sons de disparo de arma de fogo (RAHMAN *et al.*, 2021) (MOREHEAD *et al.*, 2019) (FREIRE; APOLINARIO, 2010).

Além disso, a estrutura desses ambientes também varia bastante, havendo mudanças grandes nas barreiras criadas pelas construções, podendo ser desde áreas residenciais arborizadas a grandes avenidas cercadas de altos prédios. Essas variações causam diferentes perfis de atenuação/filtragem, refração e reflexão dos eventos sonoros. Para este trabalho, esse aspecto foi desconsiderado, considerando que os dispositivos detectores mais próximos quase sempre estarão em linha reta e em um espaço aberto e sem grandes obstáculos. No entanto, para um trabalho futuro que deseje calcular uma localização precisa e rejeitar detecções múltiplas devido ao eco, esse aspecto provavelmente precise ser levado em conta.

2.1.4 Sons balísticos

O som do sisparo de arma de fogo é composto de várias partes, sendo a principal e mais facilmente detectada o som da explosão da munição. Mas além desse componente, ainda é possível observar o som da própria bala viajando, do alvo sendo atingido, dos componentes mecânicos da arma que disparou e dos inúmeros reflexos, refrações e reverberações que o disparo causa no ambiente (MAHER, 2007). Em um ambiente ruidoso como o de uma cidade, dificilmente os componentes mais sutis serão detectáveis sobre o ruído do ambiente, de forma que foi decidido focar a detecção apenas na explosão, sem contar que a relação temporal e de intensidade desses outros sons varia muito a depender do ambiente, do tipo de arma e da posição entre a fonte do evento sonoro e do sensor que capta o som.

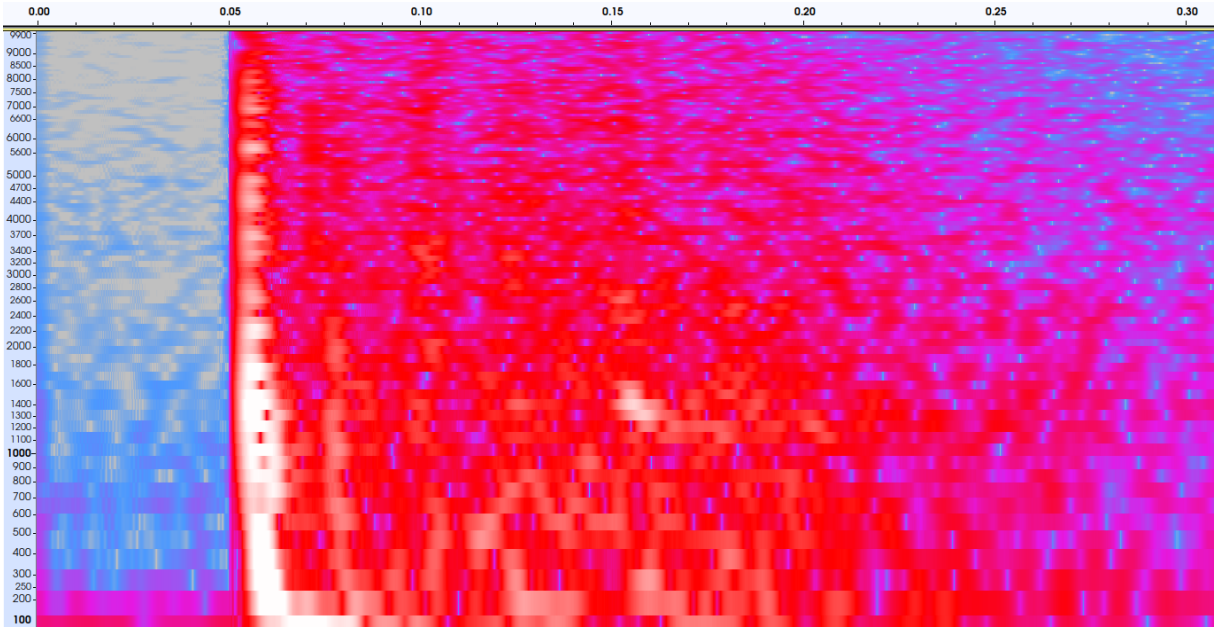
Figura 1 – Representação de uma gravação do som de disparo de arma de fogo. Os números a esquerda mostram a intensidade da amostra (onde $\pm$1.0 é o pico do microfone) e os acima o tempo em segundos. A figura mostra um forte pico inicial, com duração em torno de 10ms, uma região onde se concentra a maior parte da energia, que dura em torno de 100ms, e um decaimento longo.



Fonte: Autoria própria (2022).

Analisando o espectro de frequências do som balístico, observa-se um pico inicial e abrupto que corresponde à trajetória em linha reta da fonte do som ao sensor que está captando o som. Alguns milissegundos depois é captado o som das reflexões, que se dissipa ao longo do tempo com uma duração que depende do ambiente e da arma usada para o disparo, podendo ir desde poucas centenas de milissegundos a vários segundos (CHACON-RODRIGUEZ *et al.*, 2011). Além disso, se o microfone estiver na direção em que foi feita o disparo, é possível captar o som do projétil passando, mesmo antes do som da explosão no caso de um disparo ultrassônico (MAHER, 2007).

Figura 2 – Representação no espectro da frequência do som da Figura 1. Os números a esquerda são a escala de frequências em Hertz e os acima o tempo em segundos. As cores vermelhas e brancas indicam maior intensidade daquela frequência naquele instante de tempo, enquanto o cinza e azul representam menor intensidade.



Fonte: Autoria própria (2022).

2.1.5 Cidades inteligentes e IoT

Por cidades inteligentes entende-se cidades que usam mecanismos de captura de dados para melhorar a gerência da cidade como um todo. Esses mecanismos podem ser sensores ou cidadãos. Com isso, se procura melhorar a qualidade de vida, e poupar recursos (SHEA; BURNS, 2020).

Internet das Coisas, do inglês *Internet of Things* (IoT) é o nome que se dá para sensores espalhados por um ambiente aberto ou fechado, que possuem acesso à internet de forma direta ou indireta, transmitindo dados e/ou tomando decisões autônomas ou requisitadas pelo usuário. Isso gera uma quantidade massiva de dados que pode ser usada para fazer previsões, treinar algoritmos de aprendizado de máquina (como redes convolucionais, algoritmos genéticos, etc) ou gerir recursos de forma mais adequada (RANGER, 2020).

No contexto de cidades inteligentes, dispositivos IoT permitem o monitoramento e, por vezes, a tomada de decisão a distância. Por exemplo, dispositivos de telegestão são capazes de ler as grandezas de energia das luminárias públicas, e assim fornecer dados para que haja um melhor controle da iluminação, além de permitir que haja uma economia de energia sem prejudicar a quantidade de luminosidade mínima requerida. Em geral, esses dispositivos também são capazes de gerar alertas de forma automática, melhorando a eficiência para resolução ou prevendo problemas, como lâmpadas queimadas.

2.2 Soluções Existentes

O problema de detecção e localização de disparos é bastante antigo no meio militar, em que é importante saber a origem dos disparos de um franco-atirador ou de uma artilharia. No entanto, esses sistemas são bastante caros e especializados, e não conseguem ser aplicados em um ambiente urbano em larga escala.

Voltado especificamente para cidades, existem algumas soluções propostas e implementadas disponíveis na academia e no mercado. A mais conhecida é da empresa ShotSpotter (SHOTSPOTTER, 2022) que vende um sistema de detecção e localização de disparo de arma de fogo, que é usada em dezenas de cidades atualmente. A estrutura exata e os algoritmos usados desse sistema não são abertos, mas são usados sensores acústicos distribuídos pela cidade que fazem uma detecção inicial de potenciais eventos de disparos de armas. As informações das detecção são enviadas para um servidor central que aplica uma segunda camada de filtragem dos eventos e calcula a localização usando um algoritmo simples baseado no tempo de chegada dos eventos em cada sensor (CALHOUN *et al.*, 2021).

Fotografia 1 – Dispositivo ShotSpotter instalado em um poste na cidade de Oakland, Estados Unidos

Fonte: East Bay Express (2014).

Na literatura, vários algoritmos diferentes para detecção e localização podem ser encontrados, sendo trazidos aqui alguns exemplos que mostram a diversidade de abordagens. Freire e Apolinário (FREIRE; APOLINARIO, 2010) comparam o uso de uma correlação simples entre o áudio capturado e um exemplo de evento a ser identificado usando três modelos ocultos de Markov, cada um usando diferentes *features* extraídas do áudio, um com *Linear Predictive Coding*, um baseado em *Mel-Frequency Cepstral Coefficients* (MFCC) e o terceiro com a impulsividade. Morehead *et al* (MOREHEAD *et al.*, 2019) comparam o uso de três modelos baseados em redes neurais convolucionais que recebem como entrada os trechos de áudio como estão ou espectrograma dos trechos, dependendo do modelo. Esses modelos foram implementados pelos autorores em um Raspberry Pi, alcançando resultados esperados durante os testes *offline*

e com um desempenho um pouco abaixo nos testes em campo. Hershey *et al* (HERSHEY *et al*., 2017) compara o uso de algumas redes convolucionais desenvolvidas para classificação de imagens sendo usadas para a classificação de áudios. Rahman *et al* (RAHMAN *et al.*, 2021) descreve um sistema que aplica um extenso conjunto de *features* extraídas a 3 classificadores diferentes, comparando ainda duas versões dos modelos, uma em que o áudio é passado diretamente ao modelo e outra em que é feito um pré-processamento, eliminando partes que não alcançam um nível mínimo de energia.

2.3 Tecnologias Utilizadas

2.3.1 Sensores acústicos

Sensores acústicos são dispositivos capazes de captar sons a sua volta, e gerar um sinal analógico ou digital representando o som, que pode ser interpretado por um microcontrolador ou outro dispositivo. Alguns exemplos de sensores acústicos são microfones, que apenas escutam sons (KAUR, 2013), e alguns sensores de distância, que emitem e captam uma onda sonora, sabendo assim a distância até um obstáculo (JOST, 2019).

Os sensores acústicos que geram um sinal analógico são mais comuns no mercado, com *hardware* de baixa complexidade, sendo a precisão referente ao conversor analógico digital presente no dispositivo de, na maioria dos casos, 8 a 24 bits (ASTELS, 2018). Como, em geral, o sinal resultante é ruidoso, pode-se colocar um filtro, ou pelo menos um comparador com um dispositivo capaz de gerar tensão diferencial, como um potenciômetro, para servir como *threshold*, e assim melhorar um pouco o sinal amostrado (ARDUINO, 2019). O dispositivo captando o sinal analógico irá definir a frequência de amostragem, baseado em quantas leituras analógicas podem ser feita por segundo, lembrando que quanto maior a frequência de amostragem, melhor o sinal. É importante salientar que, quanto maior a frequência de amostragem e a quantidade de bits por amostra, maior o preço do dispositivo e maior o custo de processamento, sendo algumas vezes necessário sacrificar precisão em troca de redução de custos.

Sensores acústicos digitais tendem a ter uma qualidade melhor de captação, já que possuem circuitos integrados para redução de ruído, entre outras características (JOURNAL, 2012). Eles produzem um sinal onde a informação é codificada usando algum tipo de modulação, em geral Modulação por Densidade de Pulso, do inglês *Pulse-Density Modulation* (PDM), na qual quanto mais denso o sinal, maior a amplitude do sinal captado, e o *clock* de amostragem é dado pelo dispositivo captando o sinal. Esse é formato não é muito utilizado em análises e reprodução de sons, e por isso é necessário converter esse sinal para o formato Modulação por Código de Pulso, do inglês *Pulse-Code Modulation* (PCM), que após setar a quantidade de bits que representa uma amostra e a frequência de amostragem, pode ser tratada e ouvida como um som cru (KITE, 2012).

2.3.2 Protocolos de comunicação com periféricos

Os periféricos se comunicam com o microcontrolador usando alguns protocolos de comunicação já bem definidos no mercado. Por isso é importante conhecer as características de cada um, e ver qual deles se adéqua melhor a sua aplicação. Nesse trabalho, foram usados os seguintes protocolos:

Interface Serial Periférica, do inglês *Serial Peripheral Interface* (SPI) é um protocolo de comunicação serial capaz de alcançar altas taxas de transmissão, composto por 4 barramentos e capaz de comando agrupado. É usado pela maioria dos dispositivos que transmitem muitos dados, como sensores em geral, dispositivos de comunicação e dispositivos de armazenamento (DHAKER, 2018).

Interface de Áudio Serial, do inglês *Serial Audio Interface* (SAI), protocolo usado por dispositivos de áudio, sendo composto por um ou mais *clocks*, e um barramento para transmissão de dados. Suporta protocolos específicos como *Inter-IC Sound* (I2S) e PCM (KEIL, 2022).

Entrada/Saída Digital Segura, do inglês *Secure Digital Input/Output* (SDIO) é um protocolo serial usado para comunicação com barramentos SD, como cartão SD, e pode ser composto por até 8 barramentos (LI, 2016).

Transmissor/Receptor Assíncrono Iniversal, do inglês *Universal Asynchronous Receiver/Transmitter* (UART) é o protocolo usado para comunicação serial usado em geral em dispositivos USB ou RS232 (PEÑA; LEGASPI, 2020).

Em aplicações de tempo real se deseja ter um controle de quanto tempo se demora em cada processo, para que o processo fique dentro dos requisitos. Atualmente, existem três modos de usar os protocolos de comunicação:

- *Pooling* - Esse é o método mais comum em aplicações simples, o que garante melhor qualidade em casos de processos dedicados e também o que demanda mais recurso do microcontrolador. Ele consiste em sempre verificar uma troca de estado, que pode ser uma mudança de uma entrada Entrada/Saída de Propósito Geral, do inglês *General-Purpose Input/Output* (GPIO), espera da resposta de um comando dado num protocolo serial, entre outros.

- Interrupção - Nesse método se pede para ser avisado caso uma troca de estado ocorra. Ao contrário do *pooling*, a aplicação não fica presa na espera do evento, apenas tratando o resultado após o evento ter ocorrido. É o mais comum em aplicações mais complexas, e caso configurado corretamente pode garantir que nenhum evento seja perdido (SINGH, 2019).

- Acesso Direto à Memória, do inglês *Direct Memory Access* (DMA) - Uma mistura dos dois métodos anteriores. Nesse caso o microcontrolador possui áreas da memória que podem ser acessadas pelo periférico. Após essa área de memória ser corretamente inicializada, a comunicação com o periférico se dá de forma indireta, através dessa

área. Esse método é preferível ao de interrupção caso se esteja trabalhando com uma alta transmissão de dados, já que seria possível gerar uma quantidade muito grande de interrupções que acarrete em perda de dados, entre outros problemas. Dessa maneira o dado fica na região de memória configurada à espera de uma leitura, podendo gerar uma interrupção para avisar que a transmissão terminou e o dado está disponível para ser tratado (T, 2019).

2.3.3 Mbed

Mbed é um Sistema Operacional de Tempo Real, do inglês *Real-Time Operating System* (RTOS) (UPADHYAY, 2021) para microcontroladores, programado em C++ e voltado para dispositivos usados em IoT. Com uma quantidade grande de desenvolvedores, ele possui estruturas prontas para conexão com internet, Bluetooth, armazenamento, segurança, entre outras características. Caso se programe usando a forma proposta pela arquitetura, a aplicação se torna independente do microcontrolador e dos demais *drivers*, tornando os sistema embarcado dedicado apenas à aplicação, no lugar de específico para um único tipo de microcontrolador (MBED, 2022).

2.3.4 Wi-SUN

Rede Ubíqua Inteligente sem Fio, do inglês *Wireless Smart Ubiquitous Network* (Wi-SUN) é um protocolo de comunicação sem fio para IoT, de alta escabilidade, que utiliza o padrão IEEE 802.15.4 (IEEE, 2010). Possui protocolos de roteamento usando Protocolo de Roteamento para Redes de Baixa Potência e com Perdas, do inglês *Routing Protocol for Low-Power and Lossy Networks* (RPL) (WINTER *et al.*, 2012) de forma a criar uma rede *mesh* (CILFONE *et al.*, 2019), formada por *nodes*, que não tem acesso à internet diretamente, e por *border routers*, que possuem acesso à internet, através de Ethernet, 4G ou outras tecnologias, e encaminham as mensagens dos *nodes* de forma transparente. Além de fornecer protocolos de segurança e criptografia para a comunicação entre os dispositivos, o protocolo gera um IP público para cada dispositivo, o que permite diversas aplicabilidades (ALLIANCE, 2022).

2.3.5 CoAP

Protocolo de aplicação restrita, do inglês *Constrained Application Protocol* (CoAP) um protocolo de camada de aplicação para comunicação com a internet desenvolvido para sistemas embarcados, possuindo funções parecidas com o do Protocolo de Transferência de Hipertexto, do inglês *HyperText Transfer Protocol* (HTTP). Pode usar também um protocolo de segurança similar ao HTTPS *Secure* (HTTPS). Por ter um cabeçalho pequeno, é ideal para sistemas distri-

buídos que possuem uma quantidade grande de mensagens encaminhadas até chegar ao dispositivo com acesso direto à internet. É usado o protocolo Protocolo de Datagrama de Usuário, do inglês *User Datagram Protocol* (UDP) (WILLIAMS, 2021) mas com *acknowledge*, de forma que a aplicação do lado do cliente e do servidor pode ter certeza sobre a troca de mensagens (SHELBY; HARTKE; BORMANN, 2014).

2.3.6 Node-RED

Node-RED é uma aplicação de programação voltada para IoT, que propõe facilitar a integração entre usuário final ou Interface de Programação de Aplicação, do inglês *Application Programming Interface* (API) (ALTEXSOFT, 2021), e os dispositivos em campo. O editor pode ser executado diretamente em um navegador de internet, e possui vários nós já prontos ou que podem ser programados usando Node.js. Esses nós são ligados por fios de forma que se possui uma visão de fluxograma do sistema, em que cada parte é programável. Devido ao grande número de contribuintes, possui vários nós prontos, facilitando integração com protocolos de *hardware*, de internet, acesso a banco de dados ou serviços como *Amazon Web Services* (AWS) (AMAZON, 2022), além de permitir que a aplicação seja escalável e suporte um grande número de requisições (NODE-RED, 2022).

2.3.7 CNN

CNN é uma rede neural composta de múltiplas camadas, onde o operador para cada elemento da camada é uma convolução com as saídas do elemento da camada anterior, usando um núcleo de tamanho fixo, com os parâmetros sendo os valores de cada núcleo. Os valores extraídos de cada camada são passados para a camada seguinte, sendo capaz de otimizar cada uma para gerar uma saída ótima, o que permite que o sistema aprenda e possa ser melhorado a partir de uma etapa anterior caso se adicione mais dados na rede. Devido a essa característica de aplicar filtros na entrada, CNN são muito usadas em aplicações que envolvem imagens, mas podem ser aplicadas em outros contextos, como áudio (SAHA, 2018).

3 DESENVOLVIMENTO

3.1 Especificação

3.1.1 Modelagem de Software

O sistema como um todo foi dividido em 4 partes:

- *Software* de treinamento: *software* responsável por criar um modelo capaz de detectar o som de disparo de arma de fogo, e exportar o modelo para ser usado pelo sistema embarcado. Os detalhes serão abordados na seção 3.2.
- Sistema embarcado do *node*: *hardware* com sensor e capacidade de comunicação local, responsável por executar o modelo em um *hardware* portátil, detectando disparos de arma de fogo e enviando a informação pra um servidor. Os detalhes serão abordados na Seção 3.3
- Sistema embarcado do *border router*: *hardware* como *gateway* e roteador, responsável por criar e fornecer acesso a internet para os *nodes* da rede. Os detalhes serão abordados na seção 3.4.
- Servidor: recebe a informação das detecções de eventos vinda da rede de sensores, armazena em um meio não volátil relacionando a qual dispositivo detectou o evento, e mostra o alerta para o usuário. Os detalhes serão abordados na seção 3.5.

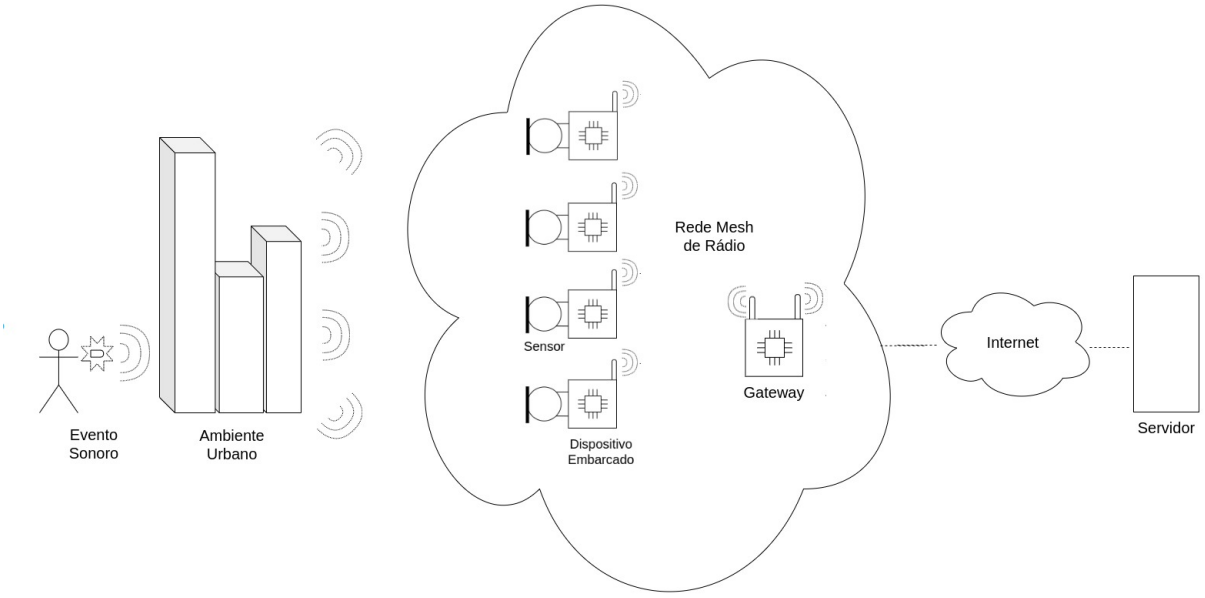
Na Figura 3 está um diagrama da visão geral do sistema.

O protocolo de comunicação entre cada uma das partes foi projetado para ser o mais transparente possível, sendo passível de aprimoramento como um todo ou de um dos componentes, sem prejudicar ou gerar uma grande manutenção nos demais.

Para o *firmware* do *node* foram projetados 4 módulos, cujas tarefas são aqui detalhadas:

- Áudio
  - Captação contínua de áudio;
  - Tratamento do áudio captado, adaptando-o para o formato esperado pelo modelo de detecção.
- Detecção:
  - Captação de características do áudio tratado;
  - Avaliação das características do áudio, classificando em uma das categorias treinadas.

Figura 3 – Visão geral do sistema



Fonte: Autoria própria (2022).

- Sistema de arquivos:
  - Guardar o segmento de áudio detectado como um disparo de arma de fogo;
  - Preparar esse dado para o servidor caso requisitado.
- Rede:
  - Encontrar e se conectar num *border router*, realizando manutenção na rede (se necessário servir de link para outro *node*);
  - Enviar ao servidor os dados de uma detecção de um disparo de arma de fogo, caso ocorra;
  - Encaminhar as requisições do servidor para o módulo correspondente, devolvendo uma resposta se necessário.

Para o *border router*, foi utilizado o projeto *Nanostack border router* (PELION, 2022), que suporta alguns protocolos de comunicação sem fio, além de diferentes *hardware* para acesso à internet (como internet cabeada ou 4G) e comunicação via rádio. Boa parte do módulo de rede foi baseada no projeto *Mbed OS Example Mesh Minimal* (ARMMBED, 2022).

3.1.2 Modelagem de *Hardware*

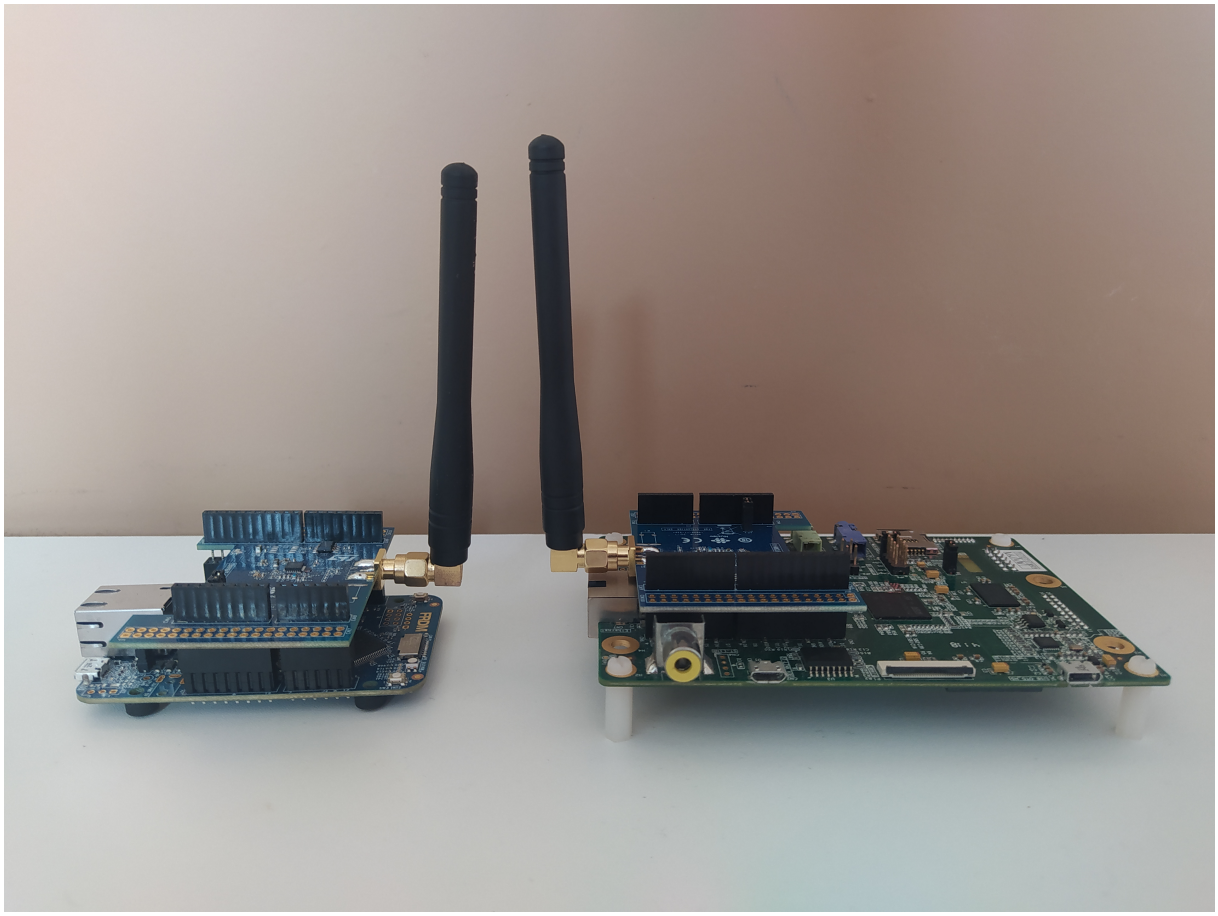
Por se tratar de um protótipo, foram escolhidas placas e módulos que fornecem o máximo de componentes necessários, evitando assim problemas inesperados de *hardware*. Isso é

importante no desenvolvimento, pois uma vez que um sistema embarcado é composto de *hardware* e *software*, em caso de problemas a origem pode estar em uma das ou ambas as partes. Usando um *hardware* confiável a probabilidade do problema estar no *software* é muito maior, o que reduz o tempo de procura pelo problema.

Para o *node* foi usado a placa STM32H747I-DISCO, pertencente a série H da STM (ST, 2022b). Essa série é voltada para alta performance, ideal para uso envolvendo inteligência artificial. A placa usada faz parte de um kit de desenvolvimento, com microfone digital de 16 bits e saída de áudio inclusos. Não foi a primeira escolha, sendo que dois outros modelos foram testados anteriormente, mas mostrou-se ser a mais equilibrada para a aplicação. Há também uma interface de Arduino para fácil adição de módulos, onde um módulo contendo um comunicador por rádio frequência S2-LP foi inserido (ST, 2022c).

Para o *border router* foi usado a placa FRDM-K64F, da NXP (NXP, 2022). Essa placa tem interface para conexão tipo Arduino na qual foi conectado o módulo S2-LP, usado pra se comunicar com os nós da rede *mesh*. Também possui uma entrada Ethernet, que a liga a um roteador conectado à internet. A Fotografia 2 mostra um exemplo de *hardware* utilizado.

Fotografia 2 – Ao lado esquerdo, um exemplo de border router. Ao lado direito, um exemplo de node



Fonte: Autoria própria (2022).

3.2 Detecção de Eventos

Por ser o elemento mais central e complexo desse trabalho, o sistema de detecção de eventos foi o primeiro a ser modelado e implementado, com uma parcela expressiva de tempo da realização do trabalho tendo sido dedicada a pesquisar as soluções já existentes, as técnicas para processamento do áudio, os modelos de detecção e classificação de eventos sonoros, as bases de dados preexistentes que trazem bons resultados e as ferramentas mais adequadas para implementar o sistema de detecção. O produto desse período de preparação foi, além da confiança no método usado, uma compreensão de qual o resultado esperado, quais são as limitações dessas técnicas e o que é possível fazer no futuro para alcançar um resultado mais próximo das soluções comerciais que existem.

3.2.1 Base de dados

Uma base de dados é uma coleção de dados, onde esse conjunto tem uma característica em comum, que deseja-se entender ou extrair novas informações. Caso cada dado desse *dataset* possua uma identidade, chamamos essa característca de label, que indica o que esse dado representa.

A base de dados utilizada para treinar o modelo de detecção de disparo de armas de fogo foi o Dataset-AOB (OSPINA, 2020), um *dataset* criado para a classificação de eventos em um contexto urbano que reúne amostras de diversas outras bases de dados, totalizando mais de 5000 sons, com aproximadamente 500 sendo de disparo de arma de fogo. Os áudios são divididos em 10 categorias diferentes: disparo de arma de fogo, alarme, criança, cachorro, motor, passos, vidro quebrando, trem, chuva e grito. A escolha desse *dataset* foi principalmente pelo seu volume e variedade, além de conter a maior parte dos sons das principais outras bases de som de disparo de arma de fogo como a SESA (SPADINI, 2019), TUT (DIMENT *et al.*, 2017), Mivia (FOGGIA *et al.*, 2015) e UrbanSounds8k (SALAMON; JACOBY; BELLO, 2014).

A desvantagem do Dataset-AOB é ele conter apenas áudios limpos, com apenas o som dos eventos isolados. Para obter um melhor resultado seria necessário misturar essas amostras com outros sons de fundo, ruído e outras amostras de eventos sonoros. No entanto, por conter áudios isolados e sem ruídos, se torna mais fácil realizar esse processo de mixagem, que poderá ser realizado em um trabalho futuro. Outra técnica que poderia ter sido utilizada para melhorar a variedade do *dataset*, mas não foi usada por questão de tempo, é a realização de algumas modificações nos sons, como variações de fase, intensidade e posição do evento dentro da amostra, conhecida como *Data Augmentation* (PARK *et al.*, 2019) (MOREHEAD *et al.*, 2019).

Além dessa base de dados já pronta, também foi gravado um conjunto de eventos em um estande para disparo de arma de fogo que foi usado para uma segunda etapa de validação, após a validação com o Dataset-AOB. Esses sons foram gravados durante uma sessão de trei-

namento do Instituto de Criminalística e têm um perfil um pouco diferente, por serem gravados em um ambiente aberto e atirando contra um alvo de metal.

3.2.2 Processamento de Áudio

O áudio na forma como ele é capturado, como uma sequência temporal de amostras, não é adequado como entrada de um modelo de detecção, sendo difícil distinguir os eventos observando apenas essas amostras. Existem algumas técnicas puramente temporais como a demonstrada em (AHMED; UPPAL; MUHAMMAD, 2013), mas elas costumam ter grau de confusão mais alto com outros sons parecidos com o de um disparo de arma de fogo, como batidas ou estouros.

Uma alternativa é utilizar o espectrograma do áudio como entrada do modelo. Nesse formato, os dados de uma janela de de áudio de duração finita são organizados em forma matricial, no qual um dos eixos representa o tempo, o outro eixo representa as frequências do sinal. O valor em cada um dos pontos do espectrograma é a magnitude da transformada de Fourier de tempo curto daquela faixa de frequência (FLANAGAN, 1971).

Por produzir uma matriz, esse método permite a utilização de técnicas normalmente associadas a visão computacional nos dados sonoros como as redes convolucionais, que vêm trazendo resultados inovadores no mundo das imagens (LIU *et al.*, 2020). Apesar dessa técnica conseguir produzir bons resultados, ela necessita bases de dados mais extensas e modelos de detecção mais complexos, de forma que não se encaixa no escopo desse projeto (GUIRGUIS *et al.*, 2021)(HERSHEY *et al.*, 2017).

A solução utilizada nesse trabalho foi aplicar uma camada adicional de processamentos sobre o espectrograma, extraindo dados que explicam melhor o conteúdo dos áudios. Cada gravação é dividida em segmentos contíguos de duração fixa, chamados de janelas, e de cada janela é um extraído um conjunto de *features* que ao final são agregadas, obtendo estatísticas que são usadas como a entrada do modelo de detecção de evento.

As *features* utilizadas são *root mean square*, *zero crossing rate*, *spectral rolloff*, *spectral flatness*, MFCC e *chromagram* (CARMEL; YESHURUN; MOSHE, 2017) (STARK; PLUMBLEY, 2009). As duas primeiras são extraídas do domínio do tempo e as próximas duas da frequência, uma vez para cada janela de cada áudio. São obtidos os valores máximo, mínimo, o médio, a mediana e o desvio padrão desses quatro valores entre cada janela.

Já as duas últimas *features* também são tiradas da frequência, mas produzem resultados em uma forma diferente. Essas duas transformações geram uma matriz com valores no tempo e na frequência assim como o espectrograma e para cada faixa de frequência das respectivas matrizes é extraído o valor médio entre cada uma das janelas. Essas *features* foram escolhidas por serem mencionadas em outros trabalhos de detecção de sons (RAHMAN *et al.*, 2021) (CARMEL; YESHURUN; MOSHE, 2017).

Poderiam ter sido utilizadas mais *features* e mais parâmetros estatísticos de cada uma, mas isso iria aumentar o tempo de processamento do áudio e o tamanho do modelo de detecção, tornando o treinamento do mesmo mais difícil e exigindo mais do *hardware* embarcado, que já estava sendo usado perto do seu limite de recursos. Foi necessário encontrar um equilíbrio entre um sistema de detecção com bons resultados mas que exige recursos demasiadamente e um sistema leve e rápido, que no entanto produz um grau elevado de falsos positivos e negativos.

Inicialmente, o processamento de áudio foi implementado usando a biblioteca de Python librosa (MCFEE *et al.*, 2015), que providencia uma grande quantidade de algoritmos de processamento de som, incluindo todos os utilizados nesse trabalho. Todavia, ela não tem uma implementação em C ou C++, sendo preciso usar uma outra biblioteca para processar o som no dispositivo embarcado, a Gist (STARK, 2021). Após alguns testes, percebeu-se que a implementação dos algoritmos usados era diferente entre as duas bibliotecas. A solução encontrada para esse problema foi adaptar a Gist para poder ser usada no Python, trazendo algumas mudanças de implementação do librosa que produziam um melhor resultado na classificação.

3.2.3 Modelo de Detecção

A literatura apresenta muitos modelos diferentes de detecção e categorização de sons, usando diversas abordagens e com variados níveis de complexidade (GONG; CHUNG; GLASS, 2021) (GUIRGUIS *et al.*, 2021) (AHMED; UPPAL; MUHAMMAD, 2013) (CHACON-RODRIGUEZ *et al.*, 2011). Nesse trabalho, foram testados alguns métodos clássicos de classificação que apresentaram bons resultados, como *Support Vector Machine*, *Multilayer Perceptron* e *Convolutional Neural Network* (RAHMAN *et al.*, 2021) (MOREHEAD *et al.*, 2019). Entre todos os modelos testados o que produziu a melhor classificação foi uma rede convolucional composta de uma conjunto de camadas de convoluções 1D, seguida de uma camada densa de neurônios e uma camada de ativação na saída do modelo.

O modelo foi implementado usando a biblioteca de Python Keras, que permite construir de forma simplificada modelos de *machine learning*, muito utilizada para classificação de dados. Ainda, essa biblioteca é construída em cima da plataforma TensorFlow que possui uma versão para dispositivos embarcados, simplificando a implementação da detecção online na placa embarcada, sendo preciso apenas exportar o modelo treinado usando Python e carregá-lo no microcontrolador.

A solução usando librosa e Keras foi baseada em um projeto existente que usava um processo similar para classificação de sons de ambiente (DAVE, 2019).

3.2.4 Treinamento e Validação

Após vários testes com o *dataset* e decisão pelo modelo usando CNN devido aos melhores resultados na média, focou-se em melhorar o modelo usado. Para o aprendizado, usou-se a técnica de validação cruzada (ALHAMID, 2020), onde se busca a generalização de um *dataset*. Se a generalização for boa o suficiente, é possível predições para novas entradas na CNN. Para isso, é necessário particionar um *dataset* em subconjuntos que não se conhecem, onde algumas partes serão usadas para treinamento, e outras para validação e teste do sistema.

Para se treinar de forma adequada, é necessário que se tenha 3 *datasets* diferentes. Um deles, com mais dados, é usado para aprendizado, ou seja, é nele que a CNN irá buscar um modelo que classifique os dados corretamente. Como parte desse aprendizado, é usado um segundo *dataset* como teste, onde a CNN gerá uma label e compara com a label real. Conforme o desempenho do modelo, ela evolui para um modelo final. Por fim, um terceiro *dataset* é usado para validação do modelo final, dessa vez sendo executado por um sistema externo, como um ser humano, para verificar se o modelo gerado está adequado para a aplicação.

O *dataset* de treinamento foi dividido 80% para aprendizado e 20% para teste. Ambas as partes são aleatórias, portanto um áudio que estava no *dataset* de teste na rodada anterior poderia estar no *dataset* de aprendizado na rodada atual. Aliado ao fato de que não foi tratado o *dataset* para padronizar os áudios, cortar partes desnecessárias, entre outras coisas, uma rodada muitas vezes apresentava valores relativamente diferentes ao da rodada anterior. O *dataset* de validação possui aproximadamente um décimo do tamanho comparado ao de treinamento, em quantidade de áudios. Para garantir um possível resultado, a validação sempre foi rodada seguida do treinamento, já que por conter várias classes de áudio, o resultado de aprendizado mostrado no treinamento poderia ser bom para uma classe diferente da de disparo de arma de fogo. Esse processo levou em torno de 5 a 10 minutos (usando CPU), dependendo da complexidade do modelo usado.

Para avaliação da performance dos modelos, foi gerada uma matriz de confusão para a validação, valorizando-se os falsos positivos, já que numa aplicação real, cada alerta gerado requisitará uma ação policial, e com um número grande de falsos positivos o produto seria inviável, já que desperdiçaria recursos de segurança pública. A matriz de confusão nesse caso é composta dos seguintes parâmetros:

- VP (Verdadeiro positivo)- áudio que contém um disparo de arma de fogo e classificado como disparo de arma de fogo;
- FP (Falso positivo) - áudio que não contém um disparo de arma de fogo mas é classificado como disparo de arma de fogo;
- VN (Verdadeiro negativo) - áudio que não contém um disparo de arma de fogo e não é classificado como disparo de arma de fogo;

- FN (Falso negativo) - áudio que contém um disparo de arma de fogo mas não é classificado como disparo de arma de fogo.

Após muitas horas de testes, nas quais se foi melhorando as características do áudio que seriam usadas, quantidade de áudios e classes, chegou-se na conclusão que um esforço manual seria inviável para alcançar um bom resultado. Portanto, foi criado um *shell script* (BRENT, 2021) para gerar modelos, treiná-los e validá-los. Após alguns dias, chegou-se em dois modelos ótimos, com o modelo A possuindo uma melhor classificação mas com muitos falsos positivos, e o modelo B perdendo um pouco na classificação mas gerando menos falsos positivos. Os resultados estão detalhados na Capítulo 4.

Todos os modelos gerados possuem uma estrutura composta por um número de camadas de convolução unidimensional com ativação ReLU (BROWNLEE, 2019b) e diferentes tamanhos de filtros, podendo ter entre elas camadas de agrupamento (BROWNLEE, 2019a), seguido de uma camada de média (COOK, 2017), uma camada de *dropout* (MAKLIN, 2019), uma camada densa para ligação com as *labels* finais (SHARMA, 2020) e por fim uma camada de ativação usando *softmax* (BROWNLEE, 2020). A primeira camada serve de entrada, recebendo como argumento as características do áudio, e na última camada está a pontuação para cada uma das *labels*. Na Figura 4 é possível ter uma visualização de um modelo composto de uma estrutura parecida com a informada, caso se tratasse de um modelo 2D (onde a entrada é dado como se fosse uma imagem). Se percebe que a entrada da camada atual é sempre a saída da camada anterior, e essa entrada se torna cada vez menor conforme a entrada vai sendo transformada. As últimas camadas não são de convolução, portanto não mudam o tamanho da entrada, apenas o conteúdo, formatando para a saída esperada. O modelo A possui 9 camadas de convolução unidimensional, enquanto que o modelo B possui 11 camadas deste tipo.

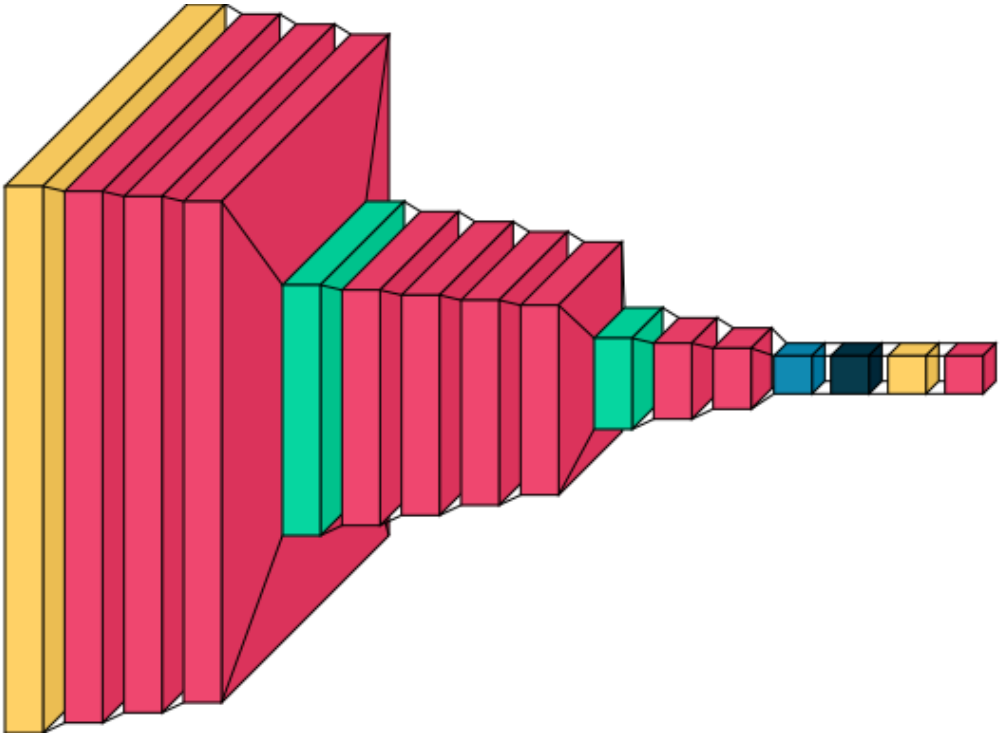
Infelizmente, ao colocar os modelos no sistema embarcado, os resultados não foram bons, devido a troca da biblioteca para captação das características dos áudios. Após a troca no *software* de treinamento, não foi possível obter modelos bons quanto os encontrados usando a biblioteca librosa.

3.3 Implementação do sistema embarcado

Uma vez com o modelo pronto, se pode usar algumas ferramentas para implementação no sistema embarcado. A escolhida foi a TensorFlow *Lite* (TENSORFLOW, 2022). Esse biblioteca permite que o modelo treinado num computador seja reduzido, a ponto de ser possível executar num sistema embarcado por exemplo.

O primeiro passo é verificar se o modelo criado tem suporte do TensorFlow Lite, uma vez que é um projeto que ainda está em desenvolvimento. Caso não haja suporte para o modelo, é necessário tentar fazer conversões. Depois disso é necessário converter o modelo original

Figura 4 – Visualização de um modelo CNN 2D



Fonte: Autoria própria (2022).

para um modelo *lite*. O TensorFlow disponibiliza uma ferramenta que faz esses processos, além de permitir uma visualização do modelo *lite* criado. Por fim, é necessário exportar esse modelo para a linguagem alvo que o TensorFlow Lite estará sendo executado. O modelo C é o resultado dessa conversão, além de alguns ajustes como a biblioteca esperar que o modelo seja fornecido na forma de um vetor e com as diferenças na biblioteca de processamento de áudio mencionadas na seção 3.2.2. A conversão da entrada para um *array* de C pode ser feita facilmente usando uma ferramenta do Linux como a xxd (WEIGERT, 1999).

Uma vez no sistema embarcado, é necessário fazer algumas calibrações de uso de memória. Uma delas é quanto de memória RAM reservada o modelo terá para executar. Outro ponto importante é que, embora a biblioteca forneça um *resolver* genérico para o seu modelo, no qual se aloca todos os recursos da biblioteca, o ideal é usar a ferramenta de análise do TensorFlow e ver exatamente qual a estrutura do modelo *lite*, alocando apenas os recursos necessários. Independente disso, o tempo de execução para ambos os modelos é o mesmo, então esse aprimoramento pode ser feito apenas no final, caso não haja problema de falta de memória no *hardware* usado.

Por fim, o modelo no sistema embarcado produz saídas semelhantes às do modelo original para uma mesma entrada, com a diferença sendo decorrente da capacidade do microprocessador no processamento de números reais. Com isso, os problemas de detecção ficam restritos apenas à entrada do modelo.

3.3.1 Sistema operacional

O sistema operacional escolhido foi o Mbed, devido à familiaridade de um dos integrantes da equipe com o mesmo. Apesar disso, o Mbed possui uma interface amigável de programação, similar ao *Arduino*. Então, com um conhecimento básico de *C++*, Padronização das interfaces de software comuns de microcontroladores, do ingês *Common Microcontroller Software Interface Standard* (CMSIS) e RTOS qualquer pessoa consegue programar para um microcontrolador usando essa arquitetura.

Por ser um sistema *open source*, existem muitos contribuintes, o que possibilita que haja uma grande variedade de recursos. A plataforma Mbed tem grande portabilidade, suportando uma variedade grande de microcontroladores, sendo possível executar o mesmo código em vários dispositivos diferentes sem precisar fazer nenhuma ou quase nenhuma modificação entre eles. Os projetos de rede usados como base para esse trabalho também usam Mbed, facilitando a adaptação do código.

Apesar do grande número de recursos fornecidos pelo Mbed, não há um aprofundamento em todos os detalhes dos microcontroladores, já que é implementado apenas o essencial para a maioria das aplicações. A empresa que produz a placa utilizada, STM, fornece uma biblioteca com vários *drivers* específicos, chamada de Pacote de Suporte à Placa, do inglês *Board Support Package* (BSP) além de alguns exemplos de como usá-los. Mas apenas adicionar essa biblioteca no projeto não resolve os problemas, já que a implementação dos *drivers* do microcontrolador é diferente para se adaptar às demais partes do Mbed, sendo então necessário modificar o BSP para gerar a compatibilidade. Por último, foi necessário modificar o *linker* do microcontrolador, para mapear recursos que não vem mapeados por padrão no Mbed.

3.3.2 Arquitetura do *software* embarcado

Em questão de arquitetura, o código do sistema embarcado possui 4 *threads*:

- *Thread* principal: responsável por inicializar os periféricos e módulos. Nessa *thread* também são executados os demais processos assíncronos para captura de áudio, ou seja, nessa thread é executado o módulo de áudio;
- *Thread* de detecção: responsável por gerar as características e classificar um áudio. Nessa *thread* é executado o módulo de detecção, e é também onde é feita a escrita dos arquivos após uma detecção;
- *Thread* de envio: responsável por receber e enviar informações e áudios para o servidor. Nessa *thread* é executado parte do sistema de arquivos (leitura) e o módulo de rede;

- *Thread watchdog*: responsável por sinalizar o *watchdog* do microcontrolador, para que em caso de algum problema, a placa seja reiniciada.

Na Figura 5 pode ser visto um fluxograma dos passos executados pelo *firmware*.

3.3.3 Aquisição de áudio

Para fazer a aquisição de áudio, é necessário definir qual tipo de sensor será usado, e qual metodologia será capaz de captar os dados sem prejudicar o resto da aplicação. Foi escolhido usar o microfone que vem integrado ao *kit* de desenvolvimento utilizado.

Como um disparo de arma de fogo é um evento de milissegundos, captar o áudio de forma bloqueante seria totalmente inviável, já que a única coisa que o microcontrolador faria seria escutar o áudio. Por se tratar de um *dual core*, num caso extremo seria possível usar um dos *cores* para essa função, e ele escreveria a informação em uma região da memória compartilhada entre os *cores*. O microfone usado não fornece uma interface de interrupção, mas se houvesse, poderia-se pensar num valor limite de intensidade de energia, onde seria gerado a interrupção e lido o áudio a partir desse momento. Entretanto, seria necessário que pensar numa máquina de estado para essas interrupções, caso contrário haveriam vários áudios sobrescritos, uma vez que um disparo de arma de fogo poderia gerar mais que uma interrupção. E também seria preciso garantir que o tratamento do evento fosse rápido o suficiente para não encher a fila de interrupção, gerando um erro que reiniciaria o código, perdendo assim o evento.

A solução final usou o DMA, no qual o dado de um sensor é diretamente armazenado numa região específica da memória. O controle desse evento é feito de forma transparente ao usuário, sendo gerado uma interrupção cada vez que a região alcança um limite preconfigurado de preenchimento. Para garantir que o microfone estava funcionando corretamente, usou-se um dos exemplos fornecidos pela STM para a placa usada, projeto esse para linguagem C, codificado para o Cube IDE, uma IDE para produtos da STM com várias ferramentas que auxiliam na configuração dos periféricos do microcontrolador (ST, 2022a). Nesse exemplo é possível falar e ouvir o eco usando a saída P2 presente na placa, com um fone de ouvido ou outro eletrônico compatível. Foram testadas algumas frequências de amostragem, e verificou-se que 22 KHz era o suficiente para ouvir um som limpo e distinto. O microfone se comunica via DMA através de uma interface SAI, gerando um dado no formato PDM. A STM fornece uma biblioteca compilada (sem acesso ao código fonte) para fazer a conversão para PCM, que é o formato usado pelos decodificadores de áudio.

Após verificação dos periféricos, aplicou-se o exemplo para o Mbed. Essa foi a parte mais demorada da implementação no sistema embarcado, com uma complexidade muito maior que a esperada, tanto que não havia sido levantado problemas relacionados a isso na análise de risco. Por se tratar de algo específico, não havia suporte para SAI no Mbed, sendo necessário usar partes do BSP, convertendo o que fosse necessário para gerar compartibilidade com o

Mbed. Por se tratar de um microcontrolador complexo e com várias divisões de memória, o SAI fornece dados para uma região de memória não mapeada pelo Mbed, sendo necessário mudar o *linker* do microcontrolador. Outra questão é que o áudio resultante é *stereo*, porém o modelo usa áudio *mono*, e pela biblioteca que faz a conversão para PDM ser uma biblioteca compilada, não se sabia qual era o padrão usado para separar os dois canais. O que dificultou ainda mais o processo é que o codificador de saída para escutar o áudio resultante só suporta áudio *stereo*, portanto o áudio resultante não parecia correto.

A princípio, não estava sendo usado a mesma biblioteca para geração de características do áudio, e quando se testou um áudio na entrada do modelo, a categoria resultante estava totalmente incoerente. O problema poderia ser na incompatibilidade das bibliotecas, ou na conversão do áudio para mono. Para resolver essa questão, resolveu-se usar um cartão SD, onde seria guardado o áudio e testado no *software* de treinamento.

O Mbed tem a capacidade de gerar um sistema de arquivos similar ao do Linux, disponibilizando uma interface genérica que independe do tipo de dispositivo sendo usado, podendo ser a memória interna, memória *flash* externa, cartão SD, etc. Para isso, é necessário fornecer uma interface chamada de *Block Device*, que provem as operações específicas do dispositivo de armazenamento. Geralmente, a comunicação é dada por SPI, porém nesse caso o mapeamento dos pinos do controlador SD estão em *I/O* comuns, usando para comunicação o protocolo SDIO diretamente. Portanto foi necessário criar essa interface, já que a mesma não é fornecida pelo Mbed. Infelizmente, por usar alguns recursos em comum, ao tentar usar o SD por DMA é gerado ruído no canal SAI, resultando num áudio não usável. Logo, foi usado escrita e leitura bloqueante. Entretanto, por estar numa *thread* separada da captura do áudio, não atrasa o processo geral.

Com o uso do cartão SD, a conversão para *mono* foi corrigida e verificada a incompatibilidade entre as bibliotecas de captura de características do sistema embarcado e do *software* de treinamento, sendo necessário manter a mesma para ambas as partes.

Para que a detecção ocorra sem perder partes de áudio, é necessário que a classificação não demore mais que o processo de captura do áudio, de forma que enquanto se está capturando um trecho de áudio, o trecho anterior já tenha sido classificado. É importante também assegurar que as amostras iniciais e finais não sejam perdidas, podendo inclusive haver sobreposição entre trechos de áudio adjacentes. Entretanto, quanto maior a janela de áudio, mais RAM é necessária para guardar esse trecho. Além disso, é necessário fazer uma cópia de cada trecho para que o processo de classificação possa usá-lo enquanto o *buffer* original é modificado pela captura usando DMA. Portanto, para uma janela de 1 segundo de áudio mono, haveria a necessidade de 88000 bytes, além do espaço necessário para replicar o final de uma janela de áudio no início de outra. Para efeito de comparação, um Arduino Mega possui 8 kB disponíveis para toda operação de RAM, portanto o tamanho da janela é um valor crítico para o funcionamento do sistema como um todo. A placa usada possui 512 kB de RAM para o *core* utilizado, então num primeiro momento o valor da janela não influenciava no funcionamento do

sistema. Para reduzir possíveis problemas, foi redirecionado o *buffer* com o áudio original para a mesma região de memória onde é feita a comunicação DMA, gerando um limitador no tamanho máximo do áudio gravado dependente do tamanho dessa nova região mapeada.

3.3.4 Processamento de áudio

Como mencionado em 3.2.2, o processamento de áudio realizado no dispositivo embarcado é uma cópia do que é em Python como parte do treinamento do modelo de detecção, extraindo o mesmo conjunto de *features*. Para diminuir a quantidade de RAM utilizada, o processamento foi dividido em dois, percorrendo o trecho de áudio duas vezes, separando o cálculo do cromagrama, que gasta mais memória que o resto. Similarmente, foram calculadas em tempo de compilação o máximo o possível de estruturas como janelas e constantes usadas nos filtros do processamento, colocando-as na memória estática. Essas modificações deram espaço o suficiente para aumentar o comprimento do trecho de áudio para em torno de 1,5 segundos. Similarmente, foram copiadas outras características do processamento, como o tamanho do *frame* de áudio analisado em cada passo, a função de janelamento usada e a frequência de amostragem.

Diferente do treinamento, que toma como trecho a ser processado um arquivo de áudio inteiro, no embarcado o áudio precisa ser processado continuamente em tempo real, analisando trechos de duração fixa. Isso pode levar um evento a ser perdido caso ele ocorra muito próximo do final do trecho, cortando-o ao meio e dificultando a detecção. Para evitar esse problema, é guardado junto com o trecho mais recente, também parte do trecho anterior, criando uma sobreposição que reduz a perda de eventos. Para determinar qual a duração da parte mais importante do áudio de um disparo de arma de fogo, que é o pico de energia, recorreu-se à literatura, como mencionado em 2.1.4, e também foi feito uma medição nos arquivos do *dataset* assim como nos gravados para avaliação do modelo, e chegou-se a um comprimento de em torno de 100ms. Então, foi escolhido uma sobreposição de 300ms com o último trecho processado, o que garante que qualquer evento fará parte por inteiro de pelo menos um trecho de áudio.

Uma das maiores dificuldades encontradas no processamento é que, para um mesmo áudio, os valores obtidos com o código que executava no microcontrolador era diferente daqueles do treinamento. Após análise constatou-se que alguns algoritmos, apesar de terem o mesmo nome no librosa e no Gist, eram efetivamente diferentes. Alguns, como o *zero crossing rate* diferiam na normalização ou algum outro aspecto pequeno. O MFCC, no entanto, tinha uma implementação bastante diferente, usando uma lógica distinta, além de usar outras constantes e filtros. Como reimplementar o algoritmo usado pelo librosa se mostrou muito complicado, optou-se por fazer o contrário e usar o Gist no código de treinamento.

3.3.5 Detecção

O modelo de detecção foi implementado em Python usando o TensorFlow, convertido para um modelo para TensorFlow Lite e então incluído no código embarcado, conforme explicado em 3.2.3 e em 3.3. O modelo é carregado pelo TensorFlow e, a cada trecho de áudio, as *features* calculadas no processamento são usadas como as entradas para o modelo, que gera uma lista de confiança de classificações em quinze categorias, as mesmas que já eram usadas no Dataset-AOB, como mencionado na Seção 3.2.1. Caso a classificação com maior pontuação seja de disparo de arma de fogo e esteja acima de um limiar, o evento é salvo no cartão SD e enviado para a rede.

O TensorFlow Lite tem opções para converter o modelo para usar apenas números inteiros e modificar a precisão para diminuir o tamanho, mas não foi necessário fazer nenhuma dessas conversões, sendo utilizado um modelo bastante próximo do treinado. Esse modelo padrão era pequeno e rápido o suficiente para ser usado no microcontrolador e com o resto do código em paralelo, mas essas simplificações são um caminho possível a serem explorados em um trabalho futuro.

3.4 Rede de comunicação

A rede de comunicação foi baseada no exemplo para Wi-SUN fornecido pelo Mbed, além de experiência anterior de um dos integrantes do grupo, o que deixou o processo rápido. Além disso, boa parte do código de manutenção da rede, conexão com servidor, entre outras partes, foi reaproveitado de projetos paralelos feitos por um dos integrantes do grupo. O rádio usado para comunicação sem fio possui conexão por SPI. Como só há outro periférico se comunicando por SPI no projeto, mas em outro barramento, não houve necessidade de se preocupar com barramento compartilhado. Entretanto, por ser um processo muito sensível ao tempo, o ideal é que, em caso de múltiplas conexões SPI, não se use barramento compartilhado com a placa de comunicação por radiofrequência.

Após configuração da rede do *node* e do *border router*, é feita a comunicação entre ambos, gerando uma conexão com a internet de forma transparente ao usuário e um endereço IP próprio para o *node*, que pode ser acessado de qualquer lugar seguindo as restrições de acesso do *border router* (por exemplo, usando um roteador, provavelmente só será possível encontrar outro equipamento dentro da própria rede, mas com um 4G o acesso é livre). Com isso, se gera uma gama de possibilidades para acesso a internet. A escolhida foi o CoAP, devido a experiência anterior de um dos integrantes do grupo. Os pacotes enviados possuem um cabeçalho menor que o de um HTTP, possuindo o mesmo número de funções, incluindo de segurança (não usada nesse projeto).

Cada dispositivo possui duas chaves, uma interna, e uma gerada no servidor. Após comunicação com a rede, os métodos CoAP só são registradores depois que o dispositivo con-

segue uma comunicação efetiva com o servidor, em que se confirma que o servidor conhece o dispositivo, e o dispositivo conhece o servidor. Após isso, o servidor reconhece qual o dispositivo que enviou uma informação a partir do endereço IP do mesmo. Não foi usado CoAP *Secure*, que é o CoAP com adição de uma camada de segurança no protocolo, devido estar ainda em desenvolvimento para o Mbed, o que consumiria mais tempo da equipe. Apesar disso, um ataque externo ao node pode ocorrer após o registro dos métodos, mas para isso é necessário que seja quebrada a criptografia AES-128 (DANIEL, 2021) da rede Wi-SUN.

Com a adição da rede de comunicação, apareceram problemas de falta de memória RAM que, por serem problemas de tempo de execução, têm uma correção bastante demorada. Com a memória de uma parte do código próxima ao limite, ocorrem comportamentos inesperados, como travamento do código em partes aleatórias, uso de memória por algumas partes maior do que o esperado, além de não ser gerado um erro preciso (ou algumas vezes nenhum erro), pois não há memória suficiente para realizar o ciclo de erro. A placa usada possui uma SDRAM externa de 32 MB, porém não foi possível inicializar a mesma de forma adequada, portanto usá-la como parte de *stack* das *threads* gerava endereçamento inválido. Usar apenas como área para guardar os *buffers*, como foi feito com o áudio original, corrompia o conteúdo, similar ao problema encontrado ao usar o cartão SD por DMA.

Após algumas tentativas, nas quais se tentou mudar os endereçamentos de memória para tentar usar um pouco do espaço reservado para o outro *core*, decidiu-se colocar o *buffer* de cópia do áudio na memória Flash externa fornecida pela placa. Como o acesso é feito por Interface Serial Periférica Quadruplica, do inglês *Quad Serial Peripheral Interface* (QSPI), o tempo para essa cópia e demais acessos não influenciou significativamente no tempo total necessário para fazer a detecção do áudio.

3.5 Sistema de Alerta

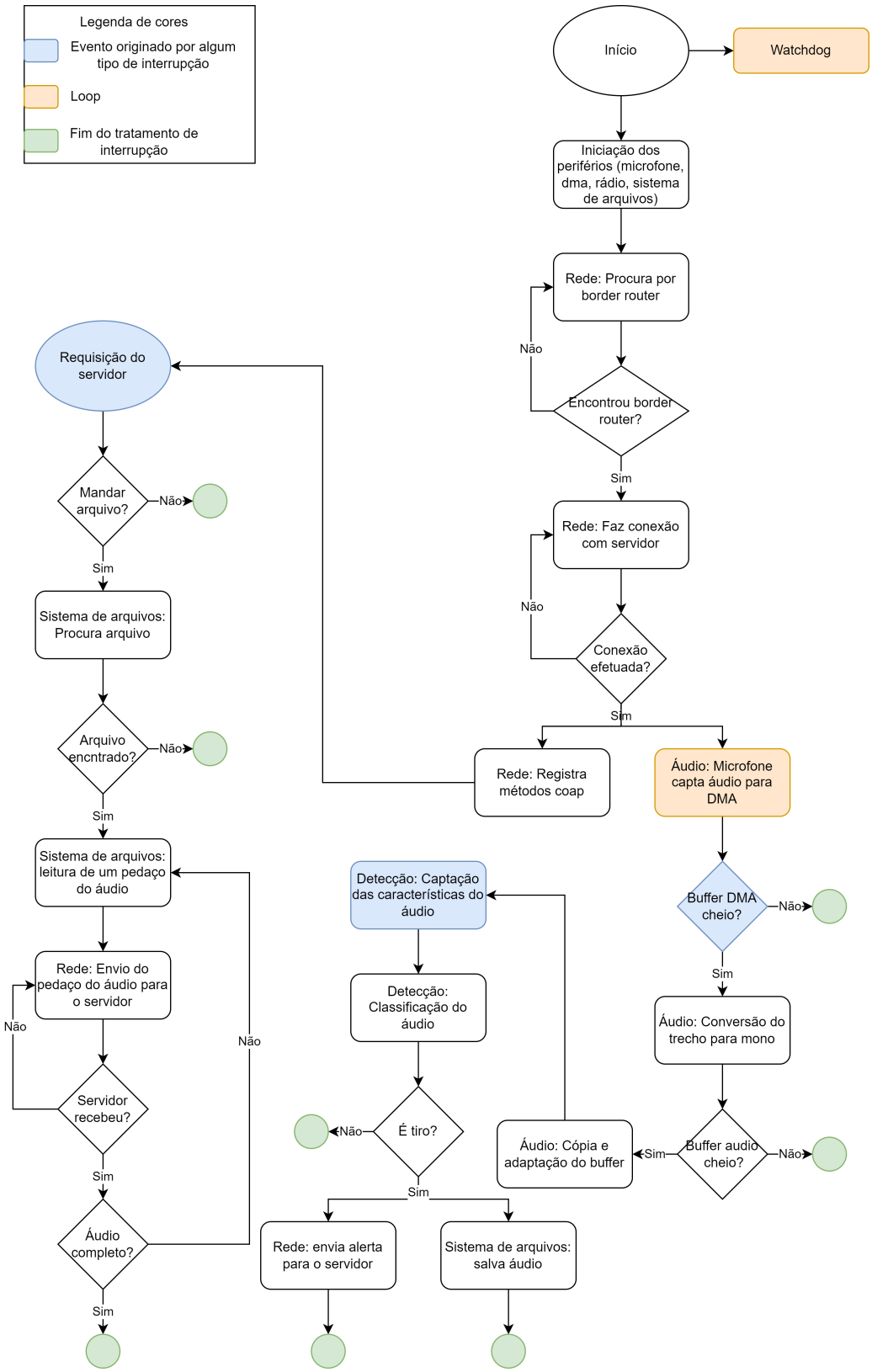
Ao detectar um disparo de arma de fogo, o *node* envia essa informação para o servidor, desenvolvido em Node-RED, escolhido devido experiência anterior de um dos integrantes do grupo. Uma pequena interface foi desenvolvida, na qual um alerta é mostrado na tela quando há um disparo de arma de fogo, informando o dispositivo, a localização (previamente cadastrada no banco de dados), qual o horário, e a porcentagem de acerto. Essa informação também é guardada numa tabela do banco de dados. Além disso, é enviado uma mensagem para um meio de comunicação que tenha as credenciais pré-cadastradas, como WhatsApp ou Telegram, avisando do ocorrido. Na Figura 6 pode ser visto a interface criada.

Não foi necessário criar nós no servidor, apenas escrever os de função, já que aqueles para as demais atribuições (interface para usuário, requisições HTTP e CoAP, acesso ao banco de dados) já existem. Boa parte do código foi reutilizada de outros projetos de um dos integrantes da equipe.

Como foi necessário guardar o áudio original do disparo de arma de fogo, foi decidido criar um método para enviar esse áudio para o servidor, permitindo que o usuário possa escutar e tomar uma decisão mais segura. Como o tamanho do áudio é muito grande para mandar em apenas um pacote, já que no CoAP há limitação de tamanho máximo de pacotes próximo de 1280 bytes, foi repartido o áudio e enviado 1 kB por vez. Embora a comunicação por rádio seja UDP no CoAP há confirmação de recebimento de pacote, então é enviado um pacote cada vez que o servidor confirma o recebimento do pacote anterior, até completar o envio do áudio todo. Esse processo pode ser mais rápido se os pacotes forem enviados de forma seguida, e ao final reenviar apenas os pacotes que o servidor não recebeu. Para o tamanho usado no projeto, após a requisição do usuário para recebimento do áudio, se leva quase 1 minuto para receber o áudio. Vale lembrar que os processos foram feitos para serem de escaláveis e distribuídos, portanto o servidor consegue lidar com vários pacotes simultâneos de dispositivos diferentes.

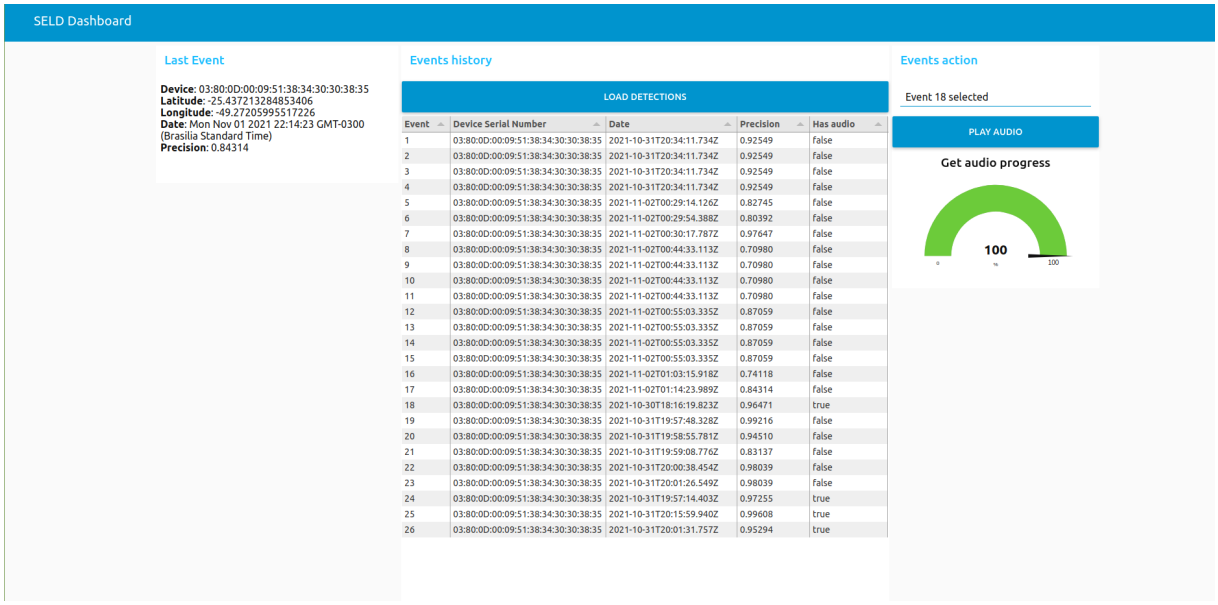
A implementação do servidor foi feita localmente, já que, caso haja um produto final, o servidor utilizado será o mesmo que é usado para os demais projetos de um dos participantes, sendo necessário apenas exportar as funções e tabelas criadas no servidor local. Além disso, a interface para o usuário será feita nos sites já existentes para clientes.

Figura 5 – Fluxograma do *firmware*



Fonte: Autoria própria (2022).

Figura 6 – Dashboard servidor



Event	Device Serial Number	Date	Precision	Has audio
1	03:80:0D:00:09:51:38:34:30:30:38:35	2021-10-31T20:34:11.734Z	0.92549	false
2	03:80:0D:00:09:51:38:34:30:30:38:35	2021-10-31T20:34:11.734Z	0.92549	false
3	03:80:0D:00:09:51:38:34:30:30:38:35	2021-10-31T20:34:11.734Z	0.92549	false
4	03:80:0D:00:09:51:38:34:30:30:38:35	2021-10-31T20:34:11.734Z	0.92549	false
5	03:80:0D:00:09:51:38:34:30:30:38:35	2021-11-02T00:29:14.126Z	0.82745	false
6	03:80:0D:00:09:51:38:34:30:30:38:35	2021-11-02T00:29:54.388Z	0.80392	false
7	03:80:0D:00:09:51:38:34:30:30:38:35	2021-11-02T00:30:17.787Z	0.97647	false
8	03:80:0D:00:09:51:38:34:30:30:38:35	2021-11-02T00:44:33.113Z	0.70980	false
9	03:80:0D:00:09:51:38:34:30:30:38:35	2021-11-02T00:44:33.113Z	0.70980	false
10	03:80:0D:00:09:51:38:34:30:30:38:35	2021-11-02T00:44:33.113Z	0.70980	false
11	03:80:0D:00:09:51:38:34:30:30:38:35	2021-11-02T00:44:33.113Z	0.70980	false
12	03:80:0D:00:09:51:38:34:30:30:38:35	2021-11-02T00:55:03.335Z	0.87059	false
13	03:80:0D:00:09:51:38:34:30:30:38:35	2021-11-02T00:55:03.335Z	0.87059	false
14	03:80:0D:00:09:51:38:34:30:30:38:35	2021-11-02T00:55:03.335Z	0.87059	false
15	03:80:0D:00:09:51:38:34:30:30:38:35	2021-11-02T00:55:03.335Z	0.87059	false
16	03:80:0D:00:09:51:38:34:30:30:38:35	2021-11-02T01:03:15.918Z	0.74118	false
17	03:80:0D:00:09:51:38:34:30:30:38:35	2021-11-02T01:14:23.989Z	0.84314	false
18	03:80:0D:00:09:51:38:34:30:30:38:35	2021-10-30T18:16:19.823Z	0.96471	true
19	03:80:0D:00:09:51:38:34:30:30:38:35	2021-10-31T19:57:48.328Z	0.99216	false
20	03:80:0D:00:09:51:38:34:30:30:38:35	2021-10-31T19:58:55.781Z	0.94510	false
21	03:80:0D:00:09:51:38:34:30:30:38:35	2021-10-31T19:59:08.776Z	0.83137	false
22	03:80:0D:00:09:51:38:34:30:30:38:35	2021-10-31T20:00:38.454Z	0.98039	false
23	03:80:0D:00:09:51:38:34:30:30:38:35	2021-10-31T20:01:26.549Z	0.98039	false
24	03:80:0D:00:09:51:38:34:30:30:38:35	2021-10-31T19:57:14.403Z	0.97255	true
25	03:80:0D:00:09:51:38:34:30:30:38:35	2021-10-31T20:15:59.940Z	0.99608	true
26	03:80:0D:00:09:51:38:34:30:30:38:35	2021-10-31T20:01:31.757Z	0.95294	true

Fonte: Autoria própria (2022).

4 RESULTADOS

Devido aos problemas não esperados para captação do áudio, a dificuldade para gerar um modelo capaz de detectar disparo de arma de fogo com uma precisão aceitável um aumento considerável na carga horária de trabalho da equipe no ambiente de trabalho, além de problemas pessoais, e principalmente devido à pandemia de COVID-19, não se testou o protótipo em campo, com disparos de arma de fogo de verdade. Porém, foi possível obter sons similares ao de um disparo, pelo menos em relação ao que se é captado pelo microfone do dispositivo. Os testes foram realizados num ambiente fechado, porém com ruídos comuns a esse ambiente, como maquinário, vozes, barulho de trânsito, entre outros. Além da simulação do disparo, foi tocado através de altos falantes os sons do *dataset* usado na validação, além de outros sons de fontes diversas, como YouTube com vídeos envolvendo uso de armas de fogo.

4.1 Detecção offline

Após se obter os melhores modelos, foi verificado o resultado para detecção dos disparos, sempre buscando o menor número de falso positivos. No Quadro 1 se vê a matriz de confusão para o modelo A, e no Quadro 2 a matriz para o modelo B.

Quadro 1 – Matriz de confusão para detecção *offline* do modelo A

Esperado / Encontrado	**Disparo de Arma**	**Não Disparo de Arma**
Disparo de Arma	40	7
Não Disparo de Arma	0	201

Fonte: Autoria própria (2022).

Quadro 2 – Matriz de confusão para detecção *offline* do modelo B

Esperado / Encontrado	**Disparo de Arma**	**Não Disparo de Arma**
Disparo de Arma	37	4
Não Disparo de Arma	3	204

Fonte: Autoria própria (2022).

Como é possível verificar, o modelo A foi capaz de detectar todos os disparos de arma de fogo do *dataset*, com um baixo índice de falso positivos. A ideia inicial era usar os dois modelos, o modelo A seria usado no sistema embarcado, e ao detectar-se um disparo, as caraterísticas passariam pelo modelo B no servidor, que gerou menos falsos positivos. Caso o disparo se confirmasse, seria gerado o alerta.

4.2 Detecção online

Conforme explicado no desenvolvimento, não foi possível replicar a captura de características do áudio no sistema embarcado, sendo necessário mudar de biblioteca, e com isso não foi possível obter um modelo tão bom de detecção quanto o anterior. O modelo atual (modelo C) criado possui estrutura similar aos demais modelos, com 11 camadas de convolução. A matriz de confusão do modelo atual (modelo C) no dispositivo para detecção offline pode ser vista no Quadro 3:

Quadro 3 – Matriz de confusão para detecção offline do modelo C

Esperado / Encontrado	Disparo de Arma	Não Disparo de Arma
Disparo de Arma	11	1
Não Disparo de Arma	29	207

Fonte: Autoria própria (2022).

Além disso, ao se tocar um áudio com um disparo de arma de fogo, mesmo o áudio usado no treinamento da rede, dificilmente o sistema embarcado reconhece como disparo. Isso provavelmente se deve à distorção do áudio quando tocado por um alto falante.

Para uma melhor comparação dos 3 modelos gerados, pode-se usar os seguintes parâmetros:

- *Acurácia* - Definido como $\frac{VP+VN}{VP+VN+FP+FN}$, indica o desempenho geral do modelo;
- *Sensibilidade* - Definido como $\frac{VP}{VP+FN}$, indica, dos eventos que eram disparo de arma de fogo, qual a porcentagem classificada corretamente pelo modelo;
- *Especificidade* - Definido como $\frac{VN}{VN+FP}$, é o complemento da sensibilidade, indicando a porcentagem dos **não**-disparos classificados corretamente.

Com isso, chega-se no Quadro 4. Percebe-se a diferença de sensibilidade e especificidade nos modelos A e B, enquanto que o modelo C possui uma especificidade maior que os demais, indicando o baixo número de falsos positivos.

Quadro 4 – Comparação entre os 3 melhores modelos

Modelo	Acurácia	Sensibilidade	Especificidade
A	97.18%	100%	96.63%
B	97.18%	92.50%	98.08%
C	87.55%	26.83%	99.52%

Fonte: Autoria própria (2022).

4.3 Comunicação

Apesar de ter sido usado apenas um dispositivo para esse protótipo, comunicação é a parte do projeto que já está validada pelos dispositivos de telegestão da empresa parceira. Já foi verificado que o sistema é escalável, continuando a manter a transparência de acesso para cada *node*. Cabe ao servidor ser implementado de forma a ser capaz de tratar todos os eventos assíncronos e algumas vezes paralelos. Já foi verificado que após alguns milhares de requisições por segundo, o servidor usado começa a apresentar certa lentidão, mas caso esse seja um problema, sua arquitetura poderá ser revista, sem ser necessário fazer uma mudança nas partes desse projeto.

Em relação aos testes feitos com um *node*, o alerta é mostrado para o usuário cerca de 2 segundos depois que o mesmo ouve o disparo de arma de fogo. O alerta em si chega no servidor em menos de 500ms após finalização da análise pelo sistema embarcado. O áudio leva cerca de 1 minuto para chegar até o servidor, mas esse tempo pode ser reduzido para menos da metade caso se faça um envio UDP, com um sistema para pegar as partes faltantes do áudio, aliado a um sistema embarcado de compressão do mesmo.

A manutenção da rede já é feita pelo Wi-SUN, porém é necessário adicionar mais pacotes CoAP para manter o servidor atualizado do estado do dispositivo, já que o servidor se comunica e conhece o dispositivo pelo endereço IP, e uma troca deste quebra a comunicação entre o servidor e o dispositivo, ou então em caso de sistema de arquivos cheio pegar mais áudios ou mandar deletar alguns, entre outras ações.

5 CONCLUSÃO

Este projeto foi uma oportunidade para revisitar e aprender alguns conceitos novos de áudio, que não foram muito explorados durante a graduação. Apesar dos resultados finais não serem os melhores possíveis, ficaram dentro do esperado, levando em conta a dificuldade do projeto, além dos imprevistos pessoais decorrentes desse período pandêmico.

Uma vez que todos os objetivos foram atingidos, considera-se que o projeto fez o papel de servir como um protótipo. A simulação de disparo de arma de fogo, da perspectiva do som captado pelo dispositivo, mostrou-se boa o suficiente para validação do projeto, além dos eventuais áudios tocados onde foi detectado o disparo. Apesar de não haver uma certeza da aplicabilidade de dispositivos como esse nas cidades, esse projeto tem futuro caso alguma empresa deseje prosseguir com o mesmo. O estudo dos modelos de inteligência artificial que melhor se aplicariam ao projeto mostrou-se adequado, uma vez que foram encontrados modelos que resultaram em 100% de sensibilidade mostrando que, com as ferramentas de captação das características do áudio corretas, pode-se obter um produto altamente confiável.

5.1 Evolução e Desenvolvimentos Futuros

Por se tratar de um protótipo, há muito o que ser feito ainda para gerar um produto final, em questão de *hardware* e detecção de disparo de arma de fogo.

Por ser um produto voltado para cidades inteligentes, considerando a aplicação atual da empresa parceira com dispositivos para telegestão, é necessário pensar em como será feita a integração com esses dispositivos em campo. Uma primeira opção seria fazer um dispositivo pequeno, substituindo a comunicação Wi-SUN por Bluetooth de Baixa Energia, do inglês *Bluetooth Low Energy* (BLE) (M, 2019), o que poderia facilitar a integração com demais sensores usando essa comunicação nos dispositivos de telegestão. Com isso, alguns problemas de memória seriam resolvidos. Porém, usando o microcontrolador atual seria necessário integrar um segundo dispositivo para poder fazer essa comunicação.

Uma segunda opção seria fazer um dispositivo misto, tendo a função de detectar disparos de arma de fogo além das já existente no dispositivo para telegestão. Com isso seria necessário resolver os problemas de memória, além de aumentar em partes a complexidade do *hardware*, uma vez que o processador usado nesse projeto possui alta complexidade de roteamento, além da precisão necessária para poder soldar o mesmo numa placa própria. Para decidir qual a melhor opção em custo e aceitação do público, é necessário fazer uma pesquisa de mercado, lembrando que mesmo que um produto com todas as funcionalidades seja preferível, talvez seja mais eficiente apenas manter dois produtos separados numa mesma embalagem.

Em questão de detecção de disparo de arma de fogo, é necessário melhorar o algoritmo para que a biblioteca Gist gere um desempenho tão bom quanto a librosa. Porém, em termos de manutenção do produto, talvez seja mais vantajoso criar uma versão da librosa para C++, pois

a mesma possui mais recursos, além de ser mais utilizada, o que gera uma melhor resolução de problemas. O *dataset* utilizado também precisa ser incrementado com áudios captados do dispositivo, já que mesmo usando a Gist, o resultado online para mesmos sons usados na validação offline é muito pior. Portanto, até que se tenha uma versão próxima do final, é necessário manter a gravação de áudio para retreinamento. Mesmo no futuro, manter alguma realimentação de áudio para a rede é importante para o melhoramento do algoritmo. Para isso é necessário gastar parte do recurso embarcado para compactar o áudio, de forma a reduzir o gasto de dados da rede. Mandar apenas as características do áudio pode não ser suficiente para melhorar a rede no começo, pois é necessário que um ser humano ouça o áudio e classifique se é realmente um som de disparo. Além disso, como visto em produtos já disponíveis no mercado, pode ser necessário ter um ser humano para dar a palavra final sobre a detecção do disparo. Logo, até se alcançar uma resolução sobre a autonomia dos serviços autônomos inteligentes, é preciso ter esse áudio guardado até para uma possível audição jurídica.

Para se ter um melhor uso comercial do projeto, é necessário incluir a localização do disparo de arma de fogo. O protocolo criado já envia as informações necessárias para implementar essa função. Por consequência, mudanças no servidor provavelmente serão suficientes para se obter uma base do problema, para se verificar se será ou não necessário um relógio melhor nos dispositivos, ou um algoritmo mais complexo.

Pensando numa manutenção em campo, ainda é necessário colocar mais pacotes no *firmware*, para se obter o estado do dispositivo, envio de configurações, além de ser necessário implementar um *Firmware* Transmitido por Ar, do inglês *Firmware Over The Air* (FOTA) para atualização em campo. E também é necessário fazer um teste de alimentação, decidindo qual bateria melhor se adequa ao uso, caso o dispositivo não seja integrado com o dispositivo de telegestão da empresa. Porém, por se tratar de um processamento pesado e contínuo, esse provavelmente será um problema complicado de ser resolvido com um baixo custo.

Para concluir, muitos testes em situações reais de disparo de arma de fogo precisam ser feitos. Como os testes teoricamente só serão feitos num ambiente controlado, como em treinamentos de disparo por policiais ou clubes de armas de fogo, é necessário que a legislação permita que esses dispositivos sejam colocados em campo para serem treinados conforme os eventos forem ocorrendo, sabendo que no começo os resultados não serão os melhores, mas que serão melhorados com o tempo.

REFERÊNCIAS

AHMED, T.; UPPAL, M.; MUHAMMAD, A. Improving efficiency and reliability of gunshot detection systems. In: **2013 IEEE International Conference on Acoustics, Speech and Signal Processing**. [*S.l.*: *s.n.*], 2013. p. 513–517.

ALHAMID, M. **What is Cross-Validation?** 2020. https://towardsdatascience.com/what-is-cross-validation-60c01f9d9e75. Acesso em: 30 mai. 2022.

ALLIANCE, W.-S. **Our Vision**. 2022. https://wi-sun.org/our-vision/. Acesso em: 14 jan. 2022.

ALTEXSOFT. **What is API: Definition, Types, Specifications, Documentation**. 2021. https://www.altexsoft.com/blog/engineering/what-is-api-definition-types-specifications-documentation/. Acesso em: 14 jan. 2022.

AMAZON. **Cloud Service - Amazon Web Services**. 2022. https://aws.amazon.com. Acesso em: 14 jan. 2022.

ARDUINO. **How to Use KY-037 Sound Detection Sensor with Arduino © GPL3+**. 2019. https://create.arduino.cc/projecthub/electropeak/how-to-use-ky-037-sound-detection-sensor-with-arduino-a757a7. Acesso em: 13 jan. 2022.

ARMMBED. **mbed-os-example-mesh-minimal**. 2022. https://github.com/ARMmbed/mbed-os-example-mesh-minimal. Acesso em: 15 jan. 2022.

ASTELS, D. **Analog Input**. 2018. https://learn.adafruit.com/mcus-how-do-they-work/analog-input. Acesso em: 13 jan. 2022.

BRENT, M. **Shell Scripting Tutorial: How to Create Shell Script in Linux/Unix**. 2021. https://www.guru99.com/introduction-to-shell-scripting.html. Acesso em: 17 jan. 2022.

BROWNLEE, J. **A Gentle Introduction to Pooling Layers for Convolutional Neural Networks**. 2019. https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/. Acesso em: 12 abr. 2022.

BROWNLEE, J. **A Gentle Introduction to the Rectified Linear Unit (ReLU)**. 2019. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/. Acesso em: 12 abr. 2022.

BROWNLEE, J. **Softmax Activation Function with Python**. 2020. https://machinelearningmastery.com/softmax-activation-function-with-python/. Acesso em: 12 abr. 2022.

CALHOUN, R. B. *et al.* **Precision and accuracy of acoustic gunshot location in an urban environment**. 2021.

CARMEL, D.; YESHURUN, A.; MOSHE, Y. Detection of alarm sounds in noisy environments. In: **2017 25th European Signal Processing Conference (EUSIPCO)**. [*S.l.*: *s.n.*], 2017. p. 1839–1843.

CERQUEIRA, D. *et al.* **Atlas da Violência 2021**. 2021. https://www.ipea.gov.br/atlasviolencia/arquivos/artigos/1375-atlasdaviolencia2021completo.pdf. Acesso em: 17 abr. 2022.

CHACON-RODRIGUEZ, A. *et al.* Evaluation of gunshot detection algorithms. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 58, n. 2, p. 363–373, 2011.

CILFONE, A. *et al.* Wireless mesh networking: An iot-oriented perspective survey on relevant technologies. https://www.mdpi.com/1999-5903/11/4/99/htm, Fevereiro 2019. Acesso em: 13 jan. 2022.

COOK, A. **Global Average Pooling Layers for Object Localization**. 2017. https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/. Acesso em: 12 abr. 2022.

DANIEL, B. **What Is AES Encryption? [The Definitive Q&A Guide]**. 2021. https://www.trentonsystems.com/blog/aes-encryption-your-faqs-answered. Acesso em: 15 jan. 2022.

DAVE, A. **Environmental-Sound-Classification**. [*S.l.*]: GitHub, 2019. https://github.com/apoorva-dave/Environmental-Sound-Classification.

DHAKER, P. **Introduction to SPI Interface**. 2018. https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html. Acesso em: 14 jan. 2022.

DIMENT, A. *et al.* **TUT Rare sound events, Development dataset**. Zenodo, 2017. The license terms are specified in the LICENSE.txt file. Disponível em: https://doi.org/10.5281/zenodo.401395.

FLANAGAN, J. L. Speech analysis, synthesis and perception. In: . [*S.l.*: *s.n.*], 1971.

FOGGIA, P. *et al.* Reliable detection of audio events in highly noisy environments. **Pattern Recognition Letters**, v. 65, p. 22–28, 2015. ISSN 0167-8655. Disponível em: https://www.sciencedirect.com/science/article/pii/S0167865515001981.

FREIRE, I.; APOLINARIO, J. Gunshot detection in noisy environments. In: . [*S.l.*: *s.n.*], 2010.

GONG, Y.; CHUNG, Y.-A.; GLASS, J. **AST: Audio Spectrogram Transformer**. arXiv, 2021. Disponível em: https://arxiv.org/abs/2104.01778.

GONZALEZ, R. Better than mfcc audio classification features. In: **The Era of Interactive Media**. New York, NY: Springer New York, 2013. p. 291–301. ISBN 978-1-4614-3501-3.

GUIRGUIS, K. *et al.* Seld-tcn: Sound event localization amp; detection via temporal convolutional networks. In: **2020 28th European Signal Processing Conference (EUSIPCO)**. [*S.l.*: *s.n.*], 2021. p. 16–20.

HERSHEY, S. *et al.* Cnn architectures for large-scale audio classification. In: **2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**. [*S.l.*: *s.n.*], 2017. p. 131–135.

IEEE. **IEEE 802.15 WPAN™ Task Group 4 (TG4)**. 2010. https://www.ieee802.org/15/pub/TG4.html. Acesso em: 14 jan. 2022.

JOST, D. **What is an Ultrasonic Sensor?** 2019. https://www.fierceelectronics.com/sensors/what-ultrasonic-sensor. Acesso em: 13 jan. 2022.

JOURNAL, M. **DIGITAL VS. ANALOG MEMS MICROPHONES**. 2012. https://www.memsjournal.com/2012/03/digital-vs-analog-mems-microphones.html. Acesso em: 13 jan. 2022.

KAUR, K. **Microphone – Sound Sensor**. 2013. https://www.azosensors.com/article.aspx?ArticleID=229. Acesso em: 13 jan. 2022.

KEIL. **SAI Interface**. 2022. https://www.keil.com/pack/doc/CMSIS/Driver/html/group__sai_ _interface__gr.html. Acesso em: 14 jan. 2022.

KITE, T. Understanding pdm digital audio. http://users.ece.utexas.edu/~bevans/courses/rtdsp/lectures/10_Data_Conversion/AP_Understanding_PDM_Digital_Audio.pdf, 2012. Acesso em: 13 jan. 2022.

LI, D. *et al.* **A Classification of general audio data for content-based retrieval**. 2000.

LI, Y. **Introduction to SPI Interface**. 2016. https://yannik520.github.io/sdio.html. Acesso em: 14 jan. 2022.

LIU, L. *et al.* Deep learning for generic object detection: A survey. **International Journal of Computer Vision**, v. 128, 02 2020.

M, M. R. **BLE- An overview**. 2019. https://medium.com/@muhamed.riyas/ble-an-overview-d524ceb73c94. Acesso em: 02 fev. 2022.

MAHER, R. Acoustical characterization of gunshots. In: . [*S.l.*: *s.n.*], 2007. p. 1 – 5. ISBN 1-4244-1226-9.

MAKLIN, C. **Dropout Neural Network Layer In Keras Explained**. 2019. https://towardsdatascience.com/machine-learning-part-20-dropout-keras-layers-explained-8c9f6dc4c9ab. Acesso em: 12 abr. 2022.

MBED. **Mbed**. 2022. https://os.mbed.com/. Acesso em: 14 jan. 2022.

MCFEE, B. *et al.* librosa: Audio and music signal analysis in python. In: . [*S.l.*: *s.n.*], 2015. p. 18–24.

MOREHEAD, A. *et al.* Low cost gunshot detection using deep learning on the raspberry pi. In: **2019 IEEE International Conference on Big Data (Big Data)**. [*S.l.*: *s.n.*], 2019. p. 3038–3044.

MORIN, S. *et al.* **Onde mora a impunidade?** 2017. http://soudapaz.org/wp-content/uploads/2019/11/index_isdp_web.pdf. Acesso em: 15 set. 2020.

NODE-RED. **Node-RED: Low-code programming for event-driven applications**. 2022. https://nodered.org/. Acesso em: 14 jan. 2022.

NXP. **FRDM-K64F: Freedom Development Platform for Kinetis® K64, K63, and K24 MCUs**. 2022. https://www.nxp.com/design/development-boards/freedom-development-boards/mcu-boards/freedom-development-platform-for-kinetis-k64-k63-and-k24-mcus:FRDM-K64F. Acesso em: 15 jan. 2022.

OSPINA, A. **Dataset-AOB: urban sounds events classification**. Zenodo, 2020. Disponível em: https://doi.org/10.5281/zenodo.4319802.

PARK, D. S. *et al.* SpecAugment: A simple data augmentation method for automatic speech recognition. In: **Interspeech 2019**. ISCA, 2019. Disponível em: https://doi.org/10.21437%2Finterspeech.2019-2680.

PELION. **nanostack-border-router**. 2022. https://github.com/PelionIoT/nanostack-border-router. Acesso em: 15 jan. 2022.

PEŇA, E.; LEGASPI, M. G. **UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter**. 2020. https://www.analog.com/en/

analog-dialogue/articles/uart-a-hardware-communication-protocol.html. Acesso em: 14 jan. 2022.

RAHMAN, S. U. *et al.* Hybrid system for automatic detection of gunshots in indoor environment. **Multimedia Tools and Applications**, v. 80, n. 3, p. 4143–4153, Jan 2021. ISSN 1573-7721. Disponível em: https://doi.org/10.1007/s11042-020-09936-w.

RANGER, S. **What is the IoT? Everything you need to know about the Internet of Things right now**. 2020. https://www.zdnet.com/article/what-is-the-internet-of-things-everything-you-need-to-know-about-the-iot-right-now/. Acesso em: 13 jan. 2022.

RIBEIRO, L.; LIMA, F. M. **Será que vai virar processo? Determinantes da elucidação dos homicídios dolosos em uma cidade brasileira**. 2018. https://doi.org/10.1590/1807-0191202026166. Acesso em: 15 set. 2020.

SAHA, S. **A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way**. 2018. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. Acesso em: 14 jan. 2022.

SALAMON, J.; JACOBY, C.; BELLO, J. P. A dataset and taxonomy for urban sound research. In: **22nd ACM International Conference on Multimedia (ACM-MM'14)**. Orlando, FL, USA: [*s.n.*], 2014. p. 1041–1044.

SHARMA, P. **Keras Dense Layer Explained for Beginners**. 2020. https://machinelearningknowledge.ai/keras-dense-layer-explained-for-beginners/. Acesso em: 12 abr. 2022.

SHEA, S.; BURNS, E. **What is a Smart City? Definition from WhatIs.com**. 2020. https://internetofthingsagenda.techtarget.com/definition/smart-city. Acesso em: 13 jan. 2022.

SHELBY, Z.; HARTKE, K.; BORMANN, C. **The Constrained Application Protocol (CoAP)**. 2014. https://datatracker.ietf.org/doc/html/rfc7252. Acesso em: 14 jan. 2022.

SHOTSPOTTER. **Gunshot Detection - ShotSpotter**. 2022. https://www.shotspotter.com/law-enforcement/gunshot-detection/. Acesso em: 28 fev. 2022.

SINGH, M. K. **Difference between Interrupt and Polling**. 2019. https://www.geeksforgeeks.org/difference-between-interrupt-and-polling/. Acesso em: 14 jan. 2022.

SPADINI, T. **Sound Events for Surveillance Applications**. Zenodo, 2019. Disponível em: https://doi.org/10.5281/zenodo.3519845.

ST. **STM32CubeIDE - Integrated Development Environment for STM32**. 2022. https://www.st.com/en/development-tools/stm32cubeide.html. Acesso em: 15 jan. 2022.

ST. **STM32H747I-DISCO - Discovery kit with STM32H747XI MCU**. 2022. https://www.st.com/en/evaluation-tools/stm32h747i-disco.html. Acesso em: 15 jan. 2022.

ST. **X-NUCLEO-S2868A2 - Sub-1 GHz 868 MHz RF expansion board based on S2-LP radio for STM32 Nucleo**. 2022. https://www.st.com/en/ecosystems/x-nucleo-s2868a2.html. Acesso em: 31 mai. 2022.

STARK, A. **Gist - An Audio Analysis Library**. [*S.l.*]: GitHub, 2021. https://github.com/adamstark/Gist.

STARK, A.; PLUMBLEY, M. Real-time chord recognition for live performance. In: . [*S.l.*: *s.n.*], 2009.

T, N. **Direct Memory Access (DMA)**. 2019. https://binaryterms.com/direct-memory-access-dma.html. Acesso em: 14 jan. 2022.

TENSORFLOW. **TensorFlow Lite: ML for Mobile and Edge Devices**. 2022. https://www.tensorflow.org/lite/. Acesso em: 15 jan. 2022.

UPADHYAY, R. K. **Real Time Operating System (RTOS)**. 2021. https://www.geeksforgeeks.org/real-time-operating-system-rtos/. Acesso em: 14 jan. 2022.

WANG, Q. *et al.* A model ensemble approach for sound event localization and detection. In: **2021 12th International Symposium on Chinese Spoken Language Processing (ISCSLP)**. [*S.l.*: *s.n.*], 2021. p. 1–5.

WEIGERT, J. **xxd(1) - Linux man page**. 1999. https://linux.die.net/man/1/xxd. Acesso em: 15 jan. 2022.

WILLIAMS, L. **TCP vs UDP: Key Difference between TCP and UDP Protocol**. 2021. https://www.guru99.com/tcp-vs-udp-understanding-the-difference.html. Acesso em: 14 jan. 2022.

WINTER, E. T. *et al.* **RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks**. 2012. https://datatracker.ietf.org/doc/html/rfc6550. Acesso em: 14 jan. 2022.