UNIVERSIDADE DO ESTADO DO AMAZONAS – UEA
ESCOLA SUPERIOR DE TECNOLOGIA – EST
BACHARELADO EM ENGENHARIA ELÉTRICA

EMILY CÉLIA RODRIGUES FOGAÇA

SISTEMA DE DETECÇÃO DE CRIANÇAS EM SITUAÇÕES DE PERIGO EM PISCINAS USANDO DEEP LEARNING E IOT

MANAUS - AM
2022

EMILY CÉLIA RODRIGUES FOGAÇA

SISTEMA DE DETECÇÃO DE CRIANÇAS EM SITUAÇÕES DE PERIGO EM PISCINAS USANDO DEEP LEARNING E IOT

Projeto de Pesquisa desenvolvido durante a disciplina de Trabalho de Conclusão de Curso II e apresentada à banca avaliadora do Curso de Engenharia Elétrica da Escola Superior de Tecnologia da Universidade do Estado do Amazonas, como pré-requisito para a obtenção do título de Engenheira Eletricista.

Orientador: Prof. Dr. Almir Kimura Junior

MANAUS - AM

2022

Universidade do Estado do Amazonas – UEA
Escola Superior de Tecnologia - EST

Reitor:
André Luiz Nunes Zodahib
Vice-Reitor:
Kátia do Nascimento Coureiro
Diretor da Escola Superior de Tecnologia:
Ingrid Sammyne Gadelha Figueiredo
Coordenador do Curso de Engenharia Elétrica:
Israel Gondres Torné

Banca Avaliadora composta por: *Data da defesa: 20/05/2022.*
Prof. Almir Kimura Junior, Dr. *(Orientador)*
Prof. Bruno da Gama Monteiro, Me.
Prof. Jozias Parente de Oliveira, Dr.

CIP – Catalogação na Publicação

Fogaça, Emily Célia Rodrigues

Sistema de detecção de crianças em situações de perigo em piscinas usando Deep Learning e IoT / Emily Célia Rodrigues Fogaça/; [orientada por] Almir Kimura Junior. – Manaus: 2022.

p.: il.

Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica). Universidade do Estado do Amazonas, 2022.

1.Afogamento. 2. Detecção de crianças e adultos. 3. *Deep Learning*. 4. *Raspberry Pi*. Kimura Junior, Almir.

EMILY CÉLIA RODRIGUES FOGAÇA

SISTEMA DE DETECÇÃO DE CRIANÇAS EM SITUAÇÕES DE PERIGO EM PISCINAS USANDO DEEP LEARNING E IOT

Pesquisa desenvolvida durante a disciplina de Trabalho de Conclusão de Curso II e apresentada à banca avaliadora do Curso de Engenharia Elétrica da Escola Superior de Tecnologia da Universidade do Estado do Amazonas, como pré-requisito para a obtenção do título de Engenheira Eletricista.

Nota obtida: 9,6 (nove vírgula seis)

Aprovada em 20/05/2022

Área de concentração: Sistemas Embarcados e Inteligência Artificial.

BANCA EXAMINADORA

Orientador: Almir Kimura Junior, Dr.

Avaliador: Bruno da Gama Monteiro, Me.

Avaliador: Jozias Parente de Oliveira, Dr.

Manaus 2022

DEDICATÓRIA

Dedico este trabalho aos pais, mães e familiares que perderam suas crianças para este acidente que ocorre de maneira muitas vezes silenciosa acometendo muitas famílias.

AGRADECIMENTOS

Agradeço a minha mãe, por todo o apoio e incentivo, conseguindo me proporcionar a melhor educação possível fazendo com que eu alcançasse a faculdade, meu eterno obrigada.

Agradeço ao professor Dr. Almir Kimura Junior por ter me acolhido quando mais precisei o que me permitiu concluir o TCC.

Agradeço a todos os colegas que fiz durante a faculdade que me ajudaram de forma direta e indireta ao longo desses anos.

Por fim, agradeço ao meu parceiro e namorado Lucas Pereira Reis por todo o apoio durante esses anos de faculdade, por sempre estar ao meu lado e por sempre me ajudar.

RESUMO

Este trabalho apresenta uma proposta para identificar crianças em situações de perigo, ou seja, sozinhas perto ou dentro de piscinas utilizando técnicas de *Deep Learning*. Atualmente a segunda maior causa de mortes de crianças no Brasil é devido ao afogamento, em piscinas públicas normalmente utiliza-se um salva-vidas para cuidar de muitas pessoas, e em piscinas residenciais é necessário que responsáveis estejam inteiramente vigiando crianças por perto. Assim, a justificativa deste trabalho é promover um sistema de baixo custo capaz de fornecer uma maior segurança em piscinas particulares, de modo a prevenir possíveis acidentes. Inicialmente consolidou-se uma base de dados representativa de crianças e adultos, para ser fornecida à modelos de classificação e detecção com o intuito de avaliá-los por métricas e escolher a melhor abordagem. Os resultados obtidos mostram que a melhor abordagem foi a detecção com a YOLOv4-tiny obtendo um mAP (*Mean Average Precision*) de 89,5%, onde foi embarcada em um *Raspberry Pi*. A solução foi implementada com uma câmera, um circuito de alarme e o envio de mensagens através do MQTT, onde o modelo conseguiu realizar as detecções através dos *frames* coletados e validar que a criança se encontrava em perigo, ou seja, sem a presença de um adulto, acionando assim o alarme e o envio de mensagens.

Palavras-chave: Afogamento, Detecção de crianças e adultos, *Deep Learning*, *Raspberry Pi*.

ABSTRACT

This work presents a proposal to identify children in dangerous situations, in other words, alone near or inside swimming pools using Deep Learning techniques. Currently, the second leading cause of death for children in Brazil is due to drowning, in public swimming pools a lifeguard is usually used to take care of many people, and in residential swimming pools it is necessary that responsibles are fully watching children nearby. Thus, the reason of this work is to create a low-cost system capable of providing greater safety in private pools, in order to prevent possible accidents. Initially, a representative database of children and adults was consolidated, to be feed to a classification and detection models in order to evaluate them by metrics and choose the best approach. The results obtained show that the best approach was the detection with YOLOv4-tiny obtaining a mAP of 89.5%, and it will be loaded on a Raspberry Pi. The solution was implemented with a camera, an alarm circuit and messaging sending through MQTT, where the model was able to do the detections through the collected frames and validate that the child was in danger, in other words, without the presence of a adult, thus triggering the alarm and sending messages.

Keywords: Drowning, Detection of children and adults, Deep Learning, Raspberry Pi.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CNN – *Convolutional Neural Networks* (em português, *Redes Neurais Convolucionais*)

DL – *Deep Learning*

FPS – *Frames Per Second*

GPIO – *General Purpose Input/Output*

GPU – *Graphics Processing Units*

IDE – *Integrated Development Environment*

IOT – *Internet of Things* (em português, *Internet das Coisas*)

mAP – *Mean Average Precision*

ML – *Machine Learning*

MQTT – *Message Queuing Telemetry Transport*

SOBRASA – *Sociedade Brasileira de Salvamento Aquático*

SUMÁRIO

INTRODUÇÃO

A piscina é um dos principais meios de recreação de crianças, principalmente no verão, onde proporciona estímulo físico e bem-estar, o que contribui para se sentirem melhor no restante do dia, favorecendo melhora no sono e na alimentação. De acordo com Bittencourt (2020) embora seja uma atividade prazerosa, não é recomendado deixar as crianças livres para brincar como bem entendem e sair para realizar outra atividade, causando um risco a segurança das crianças.

Segundo a SOBRASA (2020), Sociedade Brasileira de Salvamento Aquático, cerca de 59% das mortes de crianças na faixa de 1 a 9 anos ocorrem em piscinas e residências, conforme o boletim de 2020 e ainda crianças de 4 a 12 anos que já sabem nadar se afogam mais pela sucção da bomba em piscinas.

A negligência é o principal fator para o risco de afogamento, pois o afogamento ocorre de forma muito rápida e silenciosa. Segundo Salinet (2021), um simples momento de distração pode ser fatal para a segurança de uma criança, pois em dois minutos se perde a consciência, em quatro os danos cerebrais se tornam irreversíveis. Esse olhar atento é muito importante, pois o acidente pode ocorrer até mesmo com pessoas no ambiente, porém distraídas, já que cerca de 70% dos afogamentos acontecem ao lado de um adulto. Com isso se faz necessário o acompanhamento constante de crianças e uso de equipamentos certos, como colete de salva-vidas em ambientes com água para evitar possíveis acidentes.

Como a taxa de morte por afogamentos em piscinas de crianças apresentada pela SOBRASA (2020) é alta, é importante buscar soluções inteligentes para automatizar o monitoramento de piscinas, de modo a reduzir a ocorrência desses incidentes. Dessa forma, o objetivo dessa pesquisa foi criar um sistema de baixo custo baseado em Visão Computacional capaz de detectar uma criança sozinha perto ou dentro de uma piscina, prevenindo assim possíveis acidentes. Com isso, um sistema que utiliza técnicas de *Deep Learning* aplicado em um sistema embarcado, onde ao realizar a detecção de uma criança através de uma câmera, irá emitir um alarme. Com isso, buscou-se criar uma base de dados representativa de crianças em situações reais, desenvolver um modelo de detecção e classificação para compará-los, avaliar os resultados dos modelos através de métricas para escolher a melhor abordagem e integrar o melhor modelo em um sistema embarcado, com o circuito de alarme e envio de mensagens através do protocolo MQTT.

1 REFERENCIAL TEÓRICO

Com o problema e sua respectiva proposta de solução definidos, é importante conhecer as ferramentas e métodos necessários para a viabilidade deste trabalho. Tendo em vista que a hipótese se refere a elaboração de um sistema capaz de detectar em tempo real uma criança em situação de perigo perto de uma piscina, faz-se necessário discorrer sobre o principal acidente que pode ocorrer em uma piscina, o afogamento. Esse conhecimento é fundamental para o entendimento da justificativa da pesquisa, ao mostrar como a falta de uma supervisão mais detalhada poderia prevenir tal acidente.

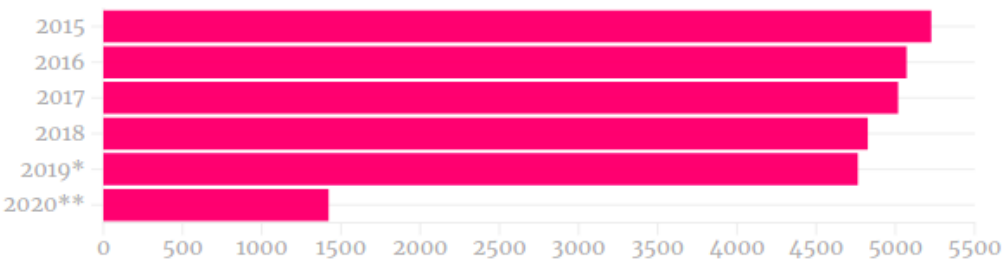
Além de entender a problemática deste trabalho, é preciso conhecer os componentes que formam a solução. Para a parte de *software* se faz necessário entender como funcionam as redes neurais convolucionais e visão computacional. Para a parte de *hardware* é importante entender a arquitetura de internet das coisas pois é a partir dessa arquitetura que será integrado o *hardware* e o *software*.

1.1 AFOGAMENTOS

Segundo Szpilman (2015) em 2002, a Organização Mundial de Saúde (OMS) definiu o afogamento como um acontecimento em que o líquido entra em contato com as vias aéreas da pessoa em imersão (água na face) ou por submersão (abaixo da superfície do líquido). Afogamento pode ser subdividido em afogamento fatal e não fatal, sendo o primeiro definido caso a pessoa morra e o segundo caso quando a pessoa é resgatada impossibilitando a morte.

A partir de dados do Ministério da Saúde, o (M)Dados, núcleo de jornalismo de dados do Metrópoles (2020), estima que desde 2005 até maio de 2020, 26.325 pessoas morreram afogadas no Brasil, o que equivale a 12 mortes a cada 24 horas. Em 2020, até maio, foi registrado 1.422 óbitos, conforme mostra na **Figura 1**.

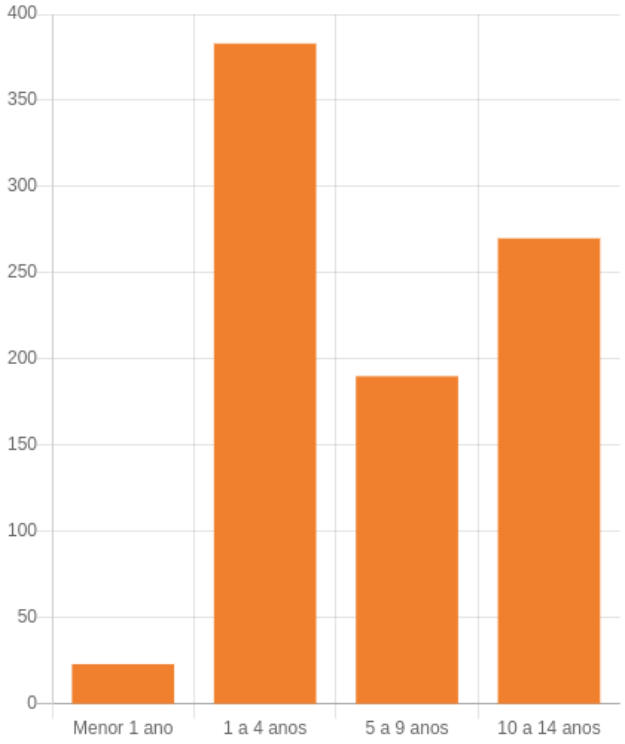
Figura 1 - Mortes por afogamento em 2020 (dados parciais até maio).



Fonte: *Dados Parciais ** Dados Parciais até maio (METRÓPOLES, 2020)

Dentre as vítimas, os mais afetados são jovens de 20 a 29 anos, seguidos pela segunda faixa etária mais atingida de 30 a 39 anos. Para as crianças, o cenário não difere, onde o afogamento é a segunda causa de morte entre crianças de até 14 anos.

Figura 2 - Mortes de crianças por afogamento em 2018.



Fonte: (SEGURA, 2018)

Na **Figura 2** podemos perceber que a faixa mais atingida é de 1 a 4 anos, com 383 mortes. É um número alto, visto que grande parte das mortes acontecem em piscinas residenciais, sob vigilância não rigorosa.

1.2 APRENDIZAGEM DE MÁQUINA

Segundo Marsland (2015), *Machine Learning* (ML) ou Aprendizado de Máquina é a forma como os computadores modificam ou adaptam suas ações, seja essas ações fazer previsões ou controlar um robô, para ficarem mais precisas, onde a precisão (acurácia) é mensurada por quão bem essas ações escolhidas refletem nas corretas.

Um exemplo comum de como funciona a forma como os algoritmos de *Machine Learning* funcionam é ensinar uma criança a diferença entre cães e gatos usando *flashcards*. Conforme a criança pratica fazendo uma escolha errada ou certa, o desempenho dela melhora. Após a criança se tornar proficiente nas imagens dos *flashcards*, ela será capaz de classificar cães e gatos no dia a dia. De acordo com Brink, Richards e Fetherolf (2016) essa capacidade de

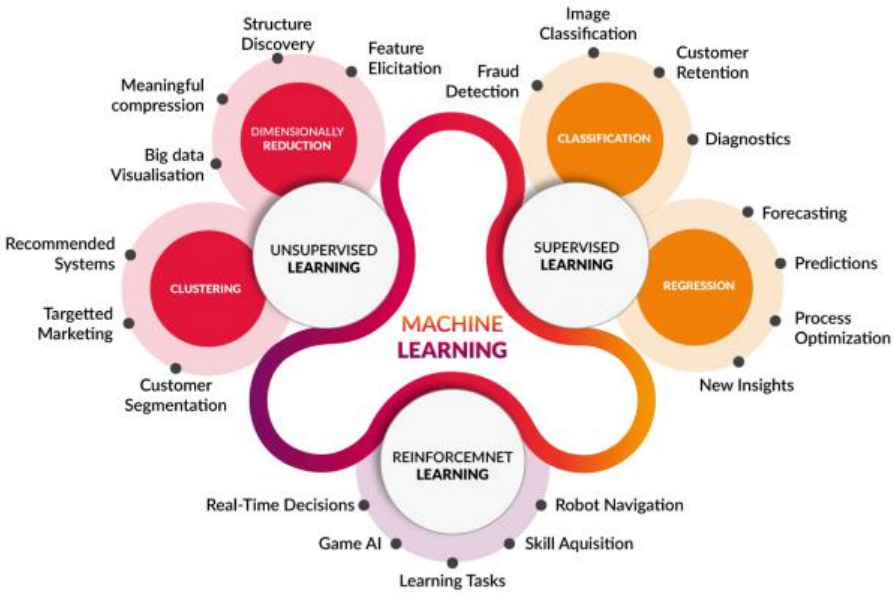
generalizar, de aplicar conhecimento adquirido através de treinamento a novos exemplos não vistos é a característica chave do aprendizado de máquina.

A utilização de *Machine Learning* resolve diversas tarefas e problemas, porém para entender como o algoritmo está acertando ou errando, é necessário fornecer diferentes respostas para esse algoritmo. De acordo com essas categorias de respostas dadas ao algoritmo quanto ao desempenho de suas saídas, Marsland (2015) as definiu da seguinte forma:

a) **aprendizagem supervisionada**: um conjunto de exemplos de treinamento com respostas corretas é fornecido e, com base neste conjunto, o algoritmo generaliza para responder corretamente a todas as entradas possíveis;
b) **aprendizagem não supervisionada**: respostas corretas não são fornecidas, mas sim o algoritmo deve inferir semelhanças entre as entradas que possuam algo em comum para serem categorizadas juntas;
c) **aprendizagem por reforço**: o algoritmo recebe a informação quando a resposta está errada, mas não é informado sobre como corrigir o erro. Ele busca explorar e experimentar diferentes possibilidades até obter a resposta correta.

A categoria de aprendizagem que será abordado neste trabalho é a aprendizagem supervisionada, visando solucionar um problema de classificação. A classificação consiste em analisar entradas e decidir de qual das *N* classes a entrada pertence.

Figura 3 - Tipos de algoritmos de *Machine Learning*.



Fonte: (IRONHACK, 2019)

1.3 APRENDIZAGEM DE MÁQUINA PROFUNDA

Deep Learning (DL) ou Aprendizado Profundo é uma subárea da aprendizagem de máquina que utiliza redes neurais profundas com uma quantidade grande de camadas ocultas na sua essência e pode gerar conteúdos baseado no aprendizado a partir de assimilação. Os algoritmos de DL são capazes de analisar dados não-estruturados sem pré-processamento ou supervisão (GOODFELLOW, BENGIO e COURVILLE, 2016).

Assim como ML, as técnicas de DL são divididas em aprendizagem supervisionada, não supervisionada e por reforço, o modelo computacional mais comum para a aprendizagem supervisionada são as redes neurais convolucionais.

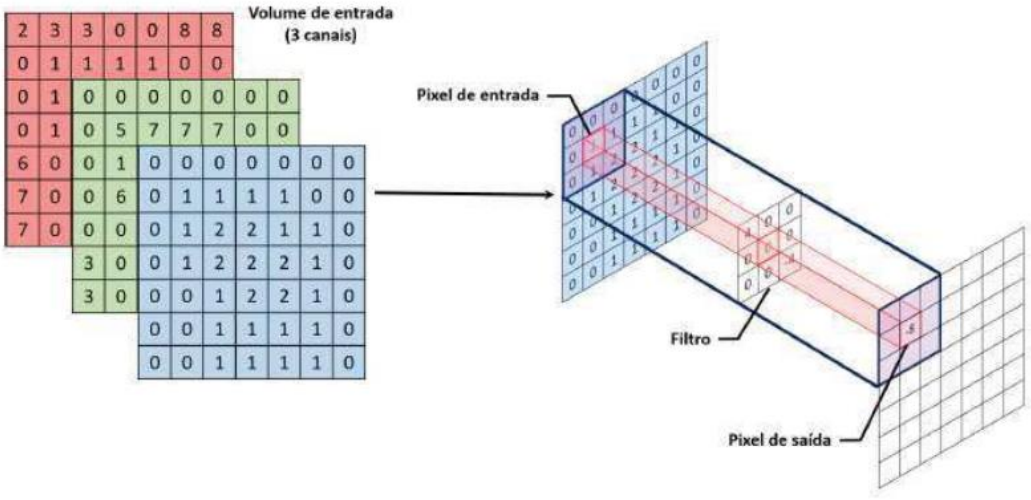
1.3.1 REDES NEURAIS CONVOLUCIONAIS

As Redes Neurais Convolucionais (CNNs, do inglês *Convolutional Neural Networks*) são um tipo de rede neural capaz de reconhecer padrões através da convolução de imagens, ou seja, partindo de uma imagem obtém-se outra imagem sobre um operador linear, chamado de kernel (GOODFELLOW, BENGIO e COURVILLE, 2016). Podem ser aplicadas em tarefas de classificação, regressão, detecção e outras, onde se destaca no reconhecimento de dados de alta dimensionalidade, como imagens e vídeos (KHAN et al., 2018).

O nome “convolucional” indica que a rede utiliza uma operação matemática chamada convolução, ou seja, as redes convolucionais são redes neurais que utilizam convoluções no lugar de multiplicação de matrizes em pelo menos uma de suas camadas. (GOODFELLOW, BENGIO e COURVILLE, 2016).

A convolução é parte do processo de reconhecimento da CNN, onde é uma multiplicação ponto a ponto entre duas matrizes, sendo uma matriz mantida fixa e a outra percorre através de um deslocamento realizando multiplicações. O reconhecimento da CNN ocorre em três etapas: a convolução de cada camada de entrada, onde essas camadas realizam diversas convoluções em paralelo para produzir um conjunto de ativações lineares; a aplicação de uma função de ativação para mapear os resultados obtidos e a subamostragem (*pooling*) para condensar os resultados Goodfellow, Bengio e Courville (2016), esse processo se repete camada por camada iterando seus pesos. A **Figura 4** exemplifica esse processo.

Figura 4 - Exemplo da convolução entre um filtro 3x3 e o volume de entrada.

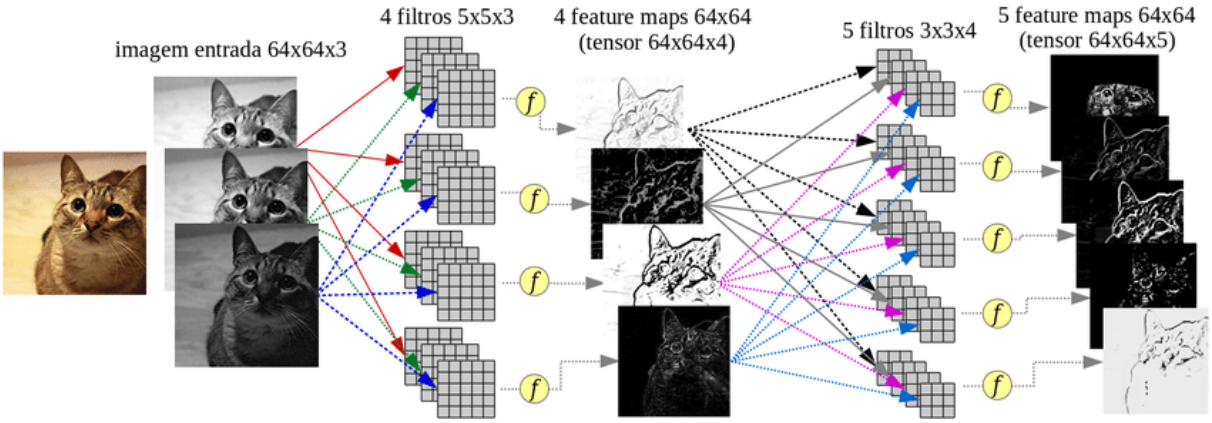


Fonte: (ARAÚJO *et al.*, 2017)

Sendo a convolução que possui o papel central nas CNNs, segundo Goodfellow, Bengio e Courville (2016) caracteriza-se pela primeira matriz de imagem (input) onde é um *array* multidimensional de dados e a segunda matriz (kernel) sendo um *array* multidimensional de pesos onde passa o núcleo de parâmetros que são adaptados pelo algoritmo de aprendizagem. O resultado desse processamento é o denominado mapa de características (*feature maps*), o qual permitirá o reconhecimento de padrões, inclusive outros padrões de rede.

Os *feature maps* resultantes da convolução, compreendem a noção de filtros, são responsáveis por capturarem características relativas à entrada, como contornos, linhas, texturas e outros. Quando combinados de maneira sequencial, essas características capturadas, vão se tornando mais complexas à medida que aumenta a profundidade da rede. Assim, um primeiro *feature map* de uma camada convolucional captura um simples contorno, enquanto um *feature map* em uma camada mais profunda captura uma forma, rosto ou objeto (BUDUMA e LOCASCIO, 2017). A **Figura 5** ilustra essa noção apresentada.

Figura 5 - Exemplo de detecção dos *features maps*.



Fonte: (PONTI e COSTA, 2017)

A estrutura de uma rede de CNN é comumente composta por blocos de operações, nomeadas de *CNN Layers* (ou camadas de CNN), essas camadas implementam ações gerais como normalização, *pooling*, convolução e *fully connected*. A camada convolucional é o componente mais importante da CNN, pois realiza a convolução de fato, sendo acompanhado de um conjunto de filtros (também intitulados de *kernels* convolucionais), sendo uma matriz de números discretos com seus valores inicializados de maneira aleatória e tunados a cada iteração (KHAN et al., 2018).

Para reduzir a dimensão dos mapas de características e refinar as características encontradas, é comum inserir uma camada de *pooling* após a camada convolucional. Esta camada visa condensar os *feature maps*, diminuindo a dimensão da matriz por um fator de 2, por um filtro de valor máximo (*max pooling*) ou valor médio (*average pooling*) reduzindo-o para torná-lo invariante a pequenas mudanças na entrada (GOODFELLOW, BENGIO e COURVILLE, 2016).

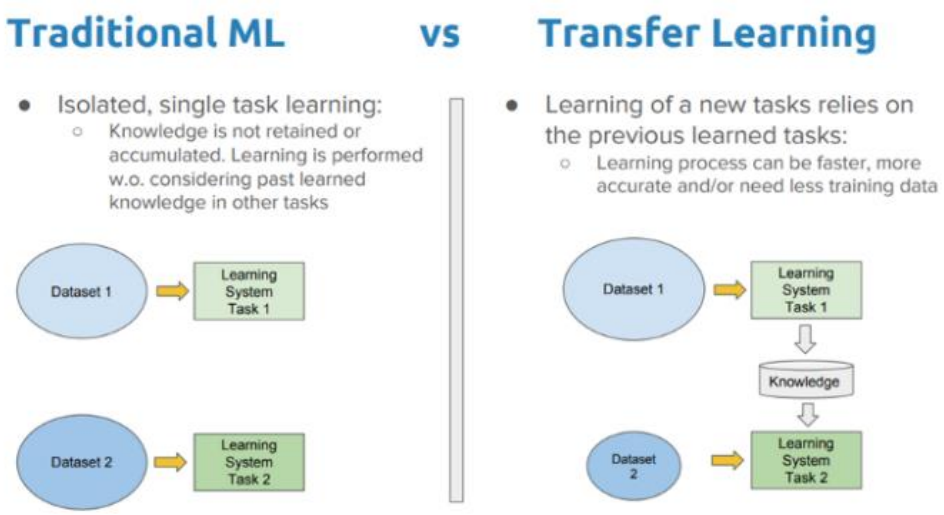
As últimas camadas de uma CNN são as *fully connected*, que correspondem à blocos convolucionais com filtros de tamanho 1x1, onde cada unidade é densamente conectada com todas as unidades da camada anterior (KHAN et al., 2018). Esta camada é seguida por uma função de ativação para obter a probabilidade de a imagem pertencer à cada classe, normalmente utiliza-se a *softmax (*WOOD, 2018) para este fim.

1.3.2 TRANSFER LEARNING

Segundo Sarkar (2018) e Brownlee (2019) a aprendizagem por transferência (*Transfer Learning*) é um método que visa superar o paradigma da aprendizagem isolada e utilizar o conhecimento adquirido para uma tarefa para resolver outra tarefa relacionada. Essa abordagem utilizada em *Deep Learning*, utiliza modelos pré-treinados como ponto de partida em tarefas de visão computacional, dado os vastos recursos computacionais e tempo necessário para desenvolver o modelo, a partir desse conhecimento como recursos, pesos e entre outros, utiliza-se para treinar modelos mais novos ou resolver problemas que tenham menos dados.

No caso de problemas no domínio da visão computacional, recursos de baixo nível, como bordas, formas, cantos e intensidade, podem ser compartilhados, permitindo a transferência de conhecimento entre tarefas. Assim, conforme mostra a **Figura 6**, a utilização dessa abordagem para o treinamento de novos modelos é simplificada, finalizando em tempo menor, mas garantindo um bom desempenho mesmo com um conjunto de dados pequenos.

Figura 6 – Aprendizagem Tradicional *versus* Aprendizagem por Transferência.



Fonte: (SARKAR, 2018)

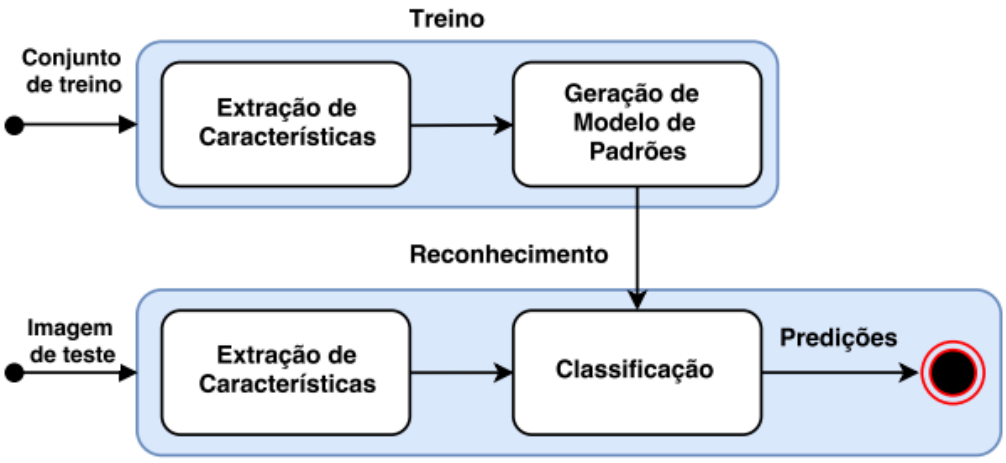
1.4 VISÃO COMPUTACIONAL

A Visão Computacional, segundo García *et al*. (2015), é o processo de extrair informações a partir de operações ou transformações feitas em dados tais como imagens, vídeos ou uma estrutura de dados multidimensional do mundo real. Esse resultado pode levar a uma tomada de decisão ou uma representação destes, de forma a descrever o que vemos a nossa volta, reconstruindo assim propriedades tais como, forma, iluminação e distribuição de cores (SZELISKI, 2010).

Aplicações de visão computacional incluem identificação de doenças médicas, anúncios de imagens e principalmente reconhecimento e detecção de objetos em imagens que é o tema proposto deste trabalho.

O sistema de reconhecimento de objetos conforme Jain, Kasturi e Schunck (1995) encontra objetos reais em uma imagem, utilizando modelos de objetos que são conhecidos. Esse problema pode ser definido como um problema de classificação baseado em objetos conhecidos. Com isso, dada uma imagem que contém um ou mais objetos conhecidos e um conjunto de *labels* que correspondem ao conjunto de modelos de objetos conhecidos pelo sistema, o algoritmo atribui classificações para regiões ou conjunto de regiões da imagem.

Figura 7 - Exemplo de esquema geral para algoritmo de reconhecimento de objetos.



Fonte: (LUCENA, VELOSO e LOPES, 2016)

A **Figura 7** exibe um diagrama para reconhecimento de objetos, na fase de treino, é onde o algoritmo busca padrões para representar objetos de interesse, primeiramente a imagem é processada por um detector de características, onde o algoritmo busca identificar formas da imagem como bordas, cor, contornos, entre outros. Em seguida esse conjunto de característica

são submetidas a um algoritmo que utiliza funções matemáticas, probabilidades e heurísticas, como *Machine Learning* para buscar separar a presença de objetos com as características extraídas na etapa anterior. Para reconhecer um objeto em uma imagem, é realizado uma combinação dos padrões que correlacionam o objeto de interesse com uma classe específica. Com isso, é realizado a classificação, através de uma medida de similaridade entre as imagens analisadas (LUCENA, VELOSO e LOPES, 2016).

1.4.1 DETECÇÃO DE OBJETOS VIA YOLO

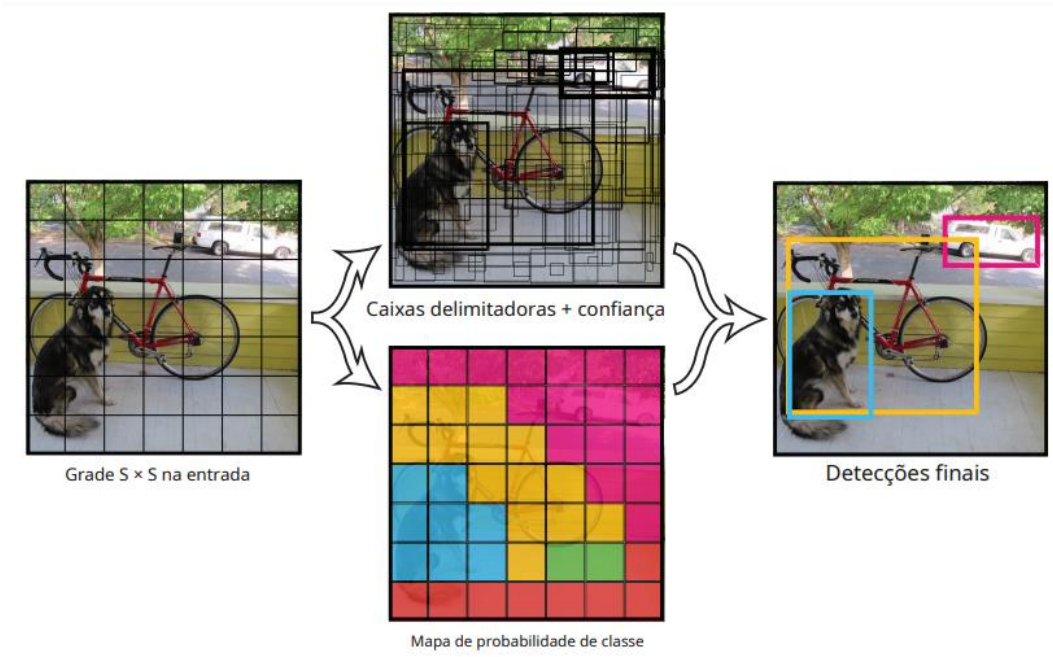
As CNNs realizaram uma revolução em visão computacional, atualmente estas redes são a abordagem dominante em quase todas as tarefas de reconhecimento e detecção ultrapassando desempenho humano em alguns casos (LECUN, BENGIO e HINTON, 2015).

Para solucionar estas tarefas, utilizam-se abordagens baseadas na combinação de métodos de reconhecimento de padrões para geração das áreas onde os objetos estão dispostos na imagem. As primeiras versões de arquitetura, utilizavam as CNNS como arquitetura base para a extração de características e uma rede auxiliar, inserida para localizar e rastrear os objetos na imagem (GIRSHICK et al, 2014).

A YOLO (*You Only Look Once*) é uma rede neural convolucional com foco para a velocidade de detecção em tempo real. Esta arquitetura apresenta uma abordagem mais moderna, transformando o problema de detecção de objetos em um problema de regressão, ou seja, não há necessidade de um pipeline complexo, apenas lê os *pixels* da imagem e retorna suas regiões de coordenadas e probabilidades de classes. A rede mais básica roda 45 quadros por segundo sem processamento em lote em uma GPU, isso significa ser possível processar streaming de vídeo em tempo real com menos de 25 milissegundos de latência (REDMON et al., 2016).

A arquitetura da YOLO usa recursos de toda a imagem para prever cada caixa delimitadora, e prevê todas as caixas delimitadoras em todas as classes para uma imagem simultaneamente, isso significa que a rede raciona globalmente sobre a imagem completa e todos os objetos contidos na imagem. Em termos técnicos, a arquitetura divide a imagem de entrada em um S x S *grid*, se o centro do objeto cair em uma célula do *grid*, essa célula é responsável por detectar esse objeto. Cada célula do *grid* prevê caixas delimitadoras (*bounding boxes*) e a confiança da predição para cada região, essas pontuações de confiança refletem o quão o modelo está confiante que um determinado objeto está presente naquela região e o quão preciso é a classe prevista (REDMON et al., 2016). A **Figura 8** exemplifica este processo.

Figura 8 – Exemplo de divisão de imagens em *grid* feita pela YOLO.



Fonte: (REDMON et al., 2016)

Este mesmo processo também se repete ao detectar novas imagens em um modelo já treinado. A imagem é inicialmente dividida em um *grid*, e cada célula prediz as *bounding boxes* e o nível de confiança, que diz o quão confidente o modelo está que a célula contém um objeto e o quão preciso é esta predição, obtendo-se assim cinco valores, onde 4 correspondem as coordenadas da imagem. Ao final deste processo, o modelo realiza múltiplas predições na imagem, e para descartar as que possuem baixa confiança, é definido um valor limite, comumente sendo 0,5 para manter somente as predições que o modelo está mais preciso.

A versão da YOLO que será abordada neste trabalho é a YOLOv4, sua arquitetura é desenvolvida em cima da *CSPDarknet53 (Cross Stage Partial)*, que utiliza 137 camadas convolucionais. Essa arquitetura possui uma velocidade duas vezes mais rápida o *EfficientDet*, além de ter sua precisão (acurácia) e FPS (*Frames Per Second*) melhores em comparação a YOLOv3 certa de 10% e 12% respectivamente. Esta versão realiza a detecção em três escalas, melhora a acurácia para objetos pequenos e aumenta o número de coordenadas, onde cada classe é prevista por regressão logística e um limiar é usado para definir quais classes serão utilizadas (BOCHKOVSKIY, WANG e LIAO, 2020).

1.5 RASPBERRY PI 3

A fundação do *Raspberry Pi* é de uma instituição de caridade educacional com sede no Reino Unido. O *Raspberry Pi*, conforme **Figura 9** é um pequeno dispositivo capaz de realizar inúmeras atividades, além de ser acessível para todas as idades explorarem.

Suas principais características técnicas são, de acordo com Souza (2016):

- Dimensões Totais: 85.6mm x 56mm x 21mm;
- Processador Broadcom BCM2837 64bit Quad Core de 1,2GHz e 1GB RAM;
- Bluetooth 4.1 e Wi-Fi integrado;
- 1 Slot para cartão MicroSD;
- Micro USB para entrada de energia. Fonte de alimentação comutada que pode lidar com até 2,5 Amperes;
- 1 Suporte para câmera;
- 1 Saída HDMI;
- 4 portas USB 2.0;
- 40 pinos externos GPIO.

Figura 9 - *Raspberry Pi* 3 modelo B.

Fonte: (FOUNDATION, 2011)

Segundo De Andrade (2016) esse hardware flexível pode realizar desde atividades mais simples como colocar dados em uma planilha até aplicações industriais, como um controlador de robô ou soluções que envolvem automação.

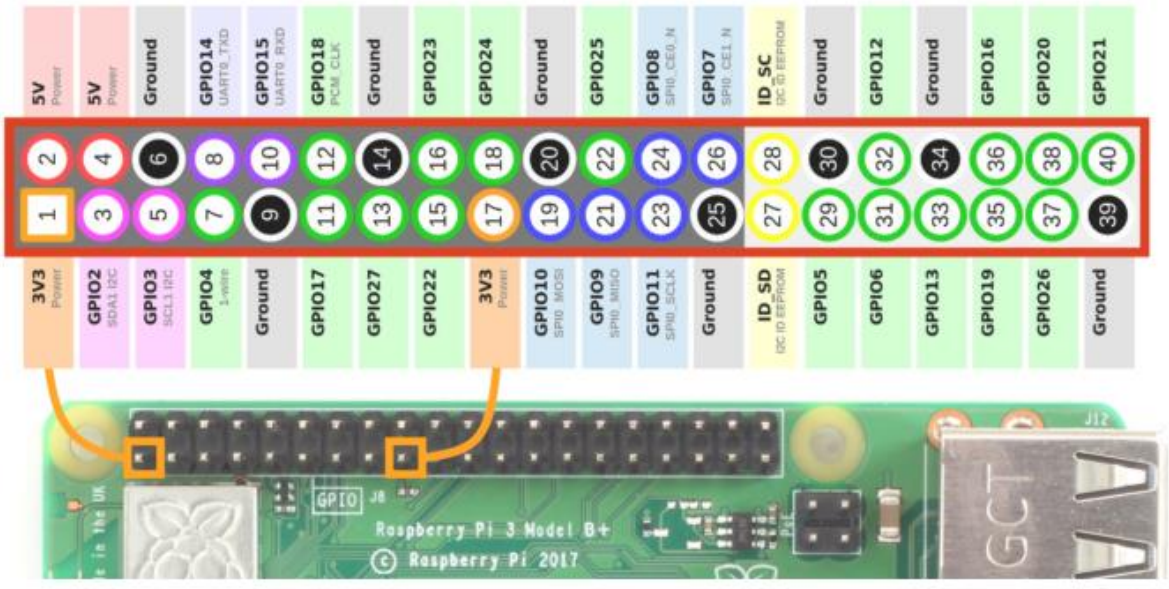
Ainda de acordo com De Andrade (2016), esse hardware apresenta vantagens como:

a) **Consumo de energia:** possui baixo nível de consumo de energia;
b) **Não possui partes móveis:** todo o sistema está integrado em uma única placa, havendo apenas a necessidade de um cartão SD para armazenamento;
c) **Silencioso:** depende da necessidade de ventoinha;
d) **Tamanho reduzido:** seu tamanho permite que seja inserido dentro de outros dispositivos;
e) **Capacidade de expansão:** há diversos dispositivos disponíveis para o uso;
f) **Gráficos:** capaz de alcançar boas resoluções para vídeos e imagens;
g) **Acessível:** possui baixo custo tanto para uso pessoal quanto comercial.

Como é uma ferramenta capaz de realizar muitas atividades, o *Raspberry Pi* possui um sistema operacional oficial baseado no kernel do Linux, cujo nome é *Raspbian* (FOUNDATION, 2011). Além desse sistema, possui suporte para outros sistemas operacionais como: Arco, Ubuntu e o Windows 10 para IoT CORE. Neste trabalho será usado o sistema padrão do *Raspberry Pi* que é instalado no cartão SD.

Além disso o *Raspberry Pi* possui um conjunto de portas de entrada e saída de uso geral (GPIO), que podem ser ilustradas na **Figura 10**.

Figura 10 – Portas GPIO Raspberry Pi.



Fonte: (MATT, 2012)

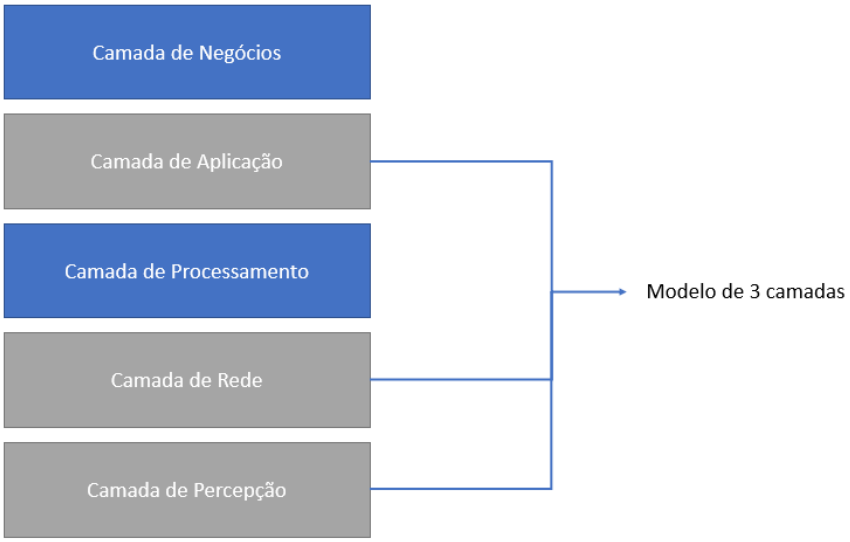
1.6 IoT – INTERNET DAS COISAS

A Internet das Coisas segundo Sethi e Sarangi (2017) refere-se a um mundo onde quase todos os dispositivos e aparelhos que usamos estão conectados a uma rede. Pode ser usado em colaboração com dispositivos para realizar tarefas complexas que exigem muita inteligência. Para essa inteligência e interconexão, os dispositivos IoT são equipados com sensores, atuadores, processadores e transceptor, com isso IoT não é uma tecnologia única, mas sim um conjunto de várias tecnologias que trabalham juntas. Suas aplicações incluem sistemas na área da saúde, *fitness*, educação, entretenimento, conservação de energia, automação residencial e outros.

A Arquitetura de cinco camadas tem como base a arquitetura de três camadas, que define a ideia principal da Internet das Coisas, mas que precisou ser atualizada devido ao rápido crescimento e desenvolvimento da área.

Assim, segundo Silva (2017), o modelo de arquitetura em cinco camadas tem-se conforme a **Figura 11**:

Figura 11 - Arquitetura de cinco camadas.



Fonte: Elaborado pela autora

a) **Camada de Percepção:** é a camada física, que possui sensores e atuadores que visam coletar e processar informações como temperatura, peso, movimento, vibração, aceleração e outros;
b) **Camada de Rede ou Transporte:** transfere os dados produzidos pela camada de percepção para a camada de processamento através de tecnologias como RFID, 3G, GSM, UMTS, *Wi-fi*, *Bluetooth* e outros;

c) **Camada de Processamento:** conhecida também como *middleware*, armazena, analisa e processa dados em abundância que vem da camada de transporte, além de gerenciar e fornece um conjunto de serviços para as camadas inferiores. Emprega tecnologias como banco de dados, computação em nuvem e módulos de processamento de big data;
d) **Camada de Aplicação:** fornece os serviços solicitados pelos usuários, como casas inteligentes, medições de temperatura, umidade do ar e outros;
e) **Camada de Negócio:** gerencia as atividade e serviços globais do sistema IoT, incluindo aplicativos, modelos de negócios e privacidade dos usuários.

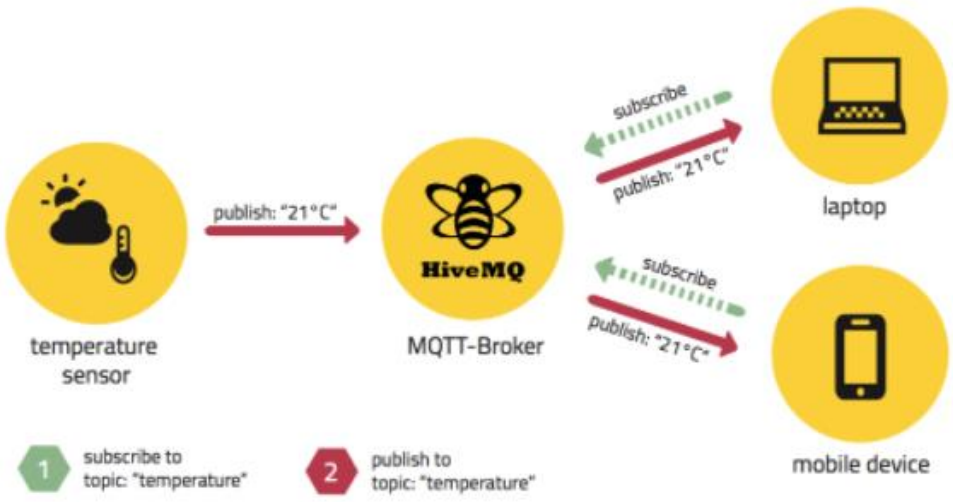
1.6.1 PROTOCOLO MQTT

Segundo Zolett e Ramirez (2020) há diversos protocolos de comunicação que são usados para comunicar a máquina e a *internet* no cenário da IoT, como o MQTT (*Message Queuing Telemetry Transport*), CoAP (*Constrained Application Protocol*) e o AMQP (*Advanced Message Queuing Protocol*). Onde esses protocolos foram desenvolvidos para serem rápidos, leves e escaláveis.

Segundo Ibm (2022) o protocolo MQTT foi criado em 1999 pelo Dr. Andy Stanford-Clark, da IBM, e Arlen Nipper, da Arcom (agora Eurotech). É um protocolo de camada de aplicação que opera sobre o protocolo TCP/IP através da rede *Wi-fi* na camada de transporte, além de ser desenvolvido para ser aplicado em dispositivos móveis.

De acordo com Jaffey (2014) o funcionamento desse protocolo é baseado na arquitetura *publish*/*subscribe.* Nessa arquitetura existem dois elementos principais chamados *broker*, que pode ser desacoplado da aplicação podendo ser hospedado em uma máquina local ou servidor *online*, que controla e envia o fluxo de mensagens, e o cliente que são os responsáveis finais por gerar e consumir os dados (GRGIĆ, ŠPEH e HEĐI, 2016). Como nessa arquitetura apenas o endereço do *broker* é conhecido, é possível conter mais de uma categoria de comunicação tais como: comunicação um para um (*one-to-one*), um para muitos (*one-to-many*) ou muitos para muitos (*many-to-many*), (TORRES, ROCHA E DE SOUZA, 2016). A **Figura 12** ilustra um exemplo dessa arquitetura.

Figura 12 – Exemplo de arquitetura *publish*/*subscribe* do protocolo MQTT.



Fonte: (BASÍLIO, 2018)

O *publish* e *subscribe* irão se comunicar com o *broker* a partir de uma assinatura, intitulada como tópico. Segundo Steve (2022) os tópicos são uma forma de endereçamento no formato *string* UTF-8 estruturados em uma hierarquia que se assemelha a estrutura de pastas e arquivos usando a barra (/) como delimitador. Na **Figura 13** se encontra um exemplo de um tópico MQTT.

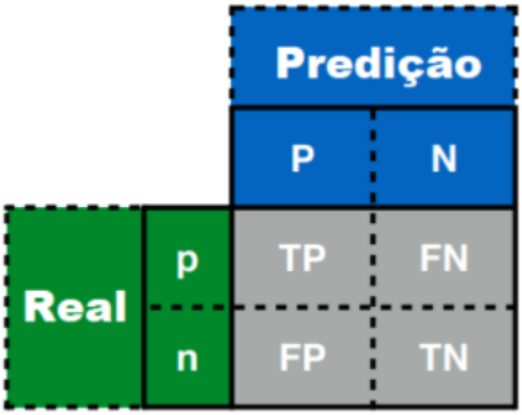
Figura 13 – Exemplo de um tópico MQTT.

Fonte: (HIVEMQ, 2019)

1.7 MÉTRICAS DE AVALIAÇÃO DE DESEMPENHO

Para avaliar o desempenho dos modelos de classificação e detecção, são utilizadas algumas métricas como precisão, revocação e *F1-score*, e para a tarefa de detecção a métrica que mais se adequa a este problema é o mAP (*Mean Average Precision*). Para entender melhor cada métrica, se faz necessário explicar a matriz de confusão e seus termos.

Figura 14 - Matriz de Confusão.



Fonte: (SILVA, 2020)

A matriz de confusão é uma matriz N x N usada para avaliar o desempenho do modelo de classificação, onde N é o número de classes do atributo alvo. A matriz compara o valor real com o valor previsto pelo modelo (ANIRUDDHA, 2020). Cada célula da matriz representa uma informação de erro ou acerto no modelo, para uma matriz 2 x 2, conforme a **Figura 14**, temos os seguintes termos:

a) **verdadeiro positivo (TP):** indica a quantidade de imagens classificadas de maneira correta para a classe positiva.
b) **verdadeiro negativo (TN):** indica a quantidade de imagens classificadas de maneira correta para a classe negativa.
c) **falso positivo (FP):** indica a quantidade de imagens classificadas de maneira incorreta para a classe negativa, ou seja, o modelo classificou como positivo o que era para ser negativo.
d) **falso negativo (FN):** indica a quantidade de imagens classificadas de maneira incorreta para a classe positiva, ou seja, o modelo classificou como negativo o que era para ser positivo.

A partir dessas informações é possível calcular as métricas apresentadas para a tarefa de classificação.

A precisão indica a relação entre a quantidade de verdadeiros positivos sobre o número total de classificações como positivos pelo modelo conforme descrito na Equação 1. Essa métrica tem como principal função minimizar os falsos positivos do modelo, ou seja, as detecções feitas de maneira errada, então quanto mais perto de 100% melhor (CASTRO e BRAGA, 2011).

$$precisão = \frac{TP}{TP + FP} \tag{1}$$

A revocação, ou *recall*, avalia a capacidade do modelo de detectar com sucesso os resultados classificados como positivos conforme descrito na Equação 2. Quanto maior a revocação, mais amostras positivas foram detectadas (GAD, 2020).

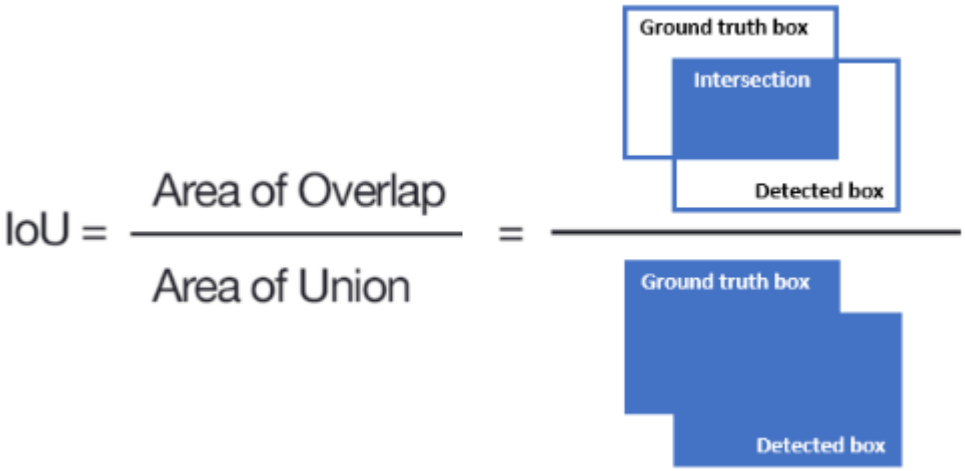
$$revocação = \frac{TP}{TP + FN} \tag{2}$$

O *F1-Score* é a média harmônica da precisão e a revocação, essa métrica funciona muito bem em dados não equilibrados (KORSTANJE, 2021). Por ser uma métrica que une outras duas, um possível valor baixo, é um indicativo de que a precisão ou o recall estejam baixo. essa média é descrita conforme a Equação 3.

$$F1 - Score = 2 * \frac{precisão * recall}{precisão + recall} \tag{3}$$

Para problemas de detecção de objetos, a métrica mais utilizada é a *Mean Average Precision*, ou mAP. Como em uma tarefa de detecção é necessário detectar e classificar o objeto encontrado na imagem, é importante avaliar se a detecção ocorreu da maneira correta e para isso temos a métrica IoU (*Intersection Over Union*) que significa uma razão entre a interseção e união das caixas delimitadoras previstas e reais, na **Figura 15** é ilustrado como ocorre:

Figura 15 - Como funciona o cálculo da IoU.



Fonte: (BAELDUNG, 2018)

Os valores da IoU são fornecidos em forma de porcentagem, então quanto maior for esse valor, mais precisa será a detecção de um objeto, porém pode ocasionar a restrição do algoritmo. Já valores baixos para esta métrica implica em um algoritmo com margem de aceitação alta, porém não muito preciso, podendo resultar em detecções equivocadas.

Com isso é calculado os valores de precisão e revocação para detecção de objetos, onde os valores de verdadeiros positivos e falsos positivos são coletados da IoU. Assim usando esses valores para um nível de aceitação > 0,5 para uma detecção ser considerada verdadeira, é possível calcular a precisão como sendo o número de detecções corretas para cada classe presente na imagem e a revocação como o valor da previsão correta (NOGUEIRA et al, 2018).

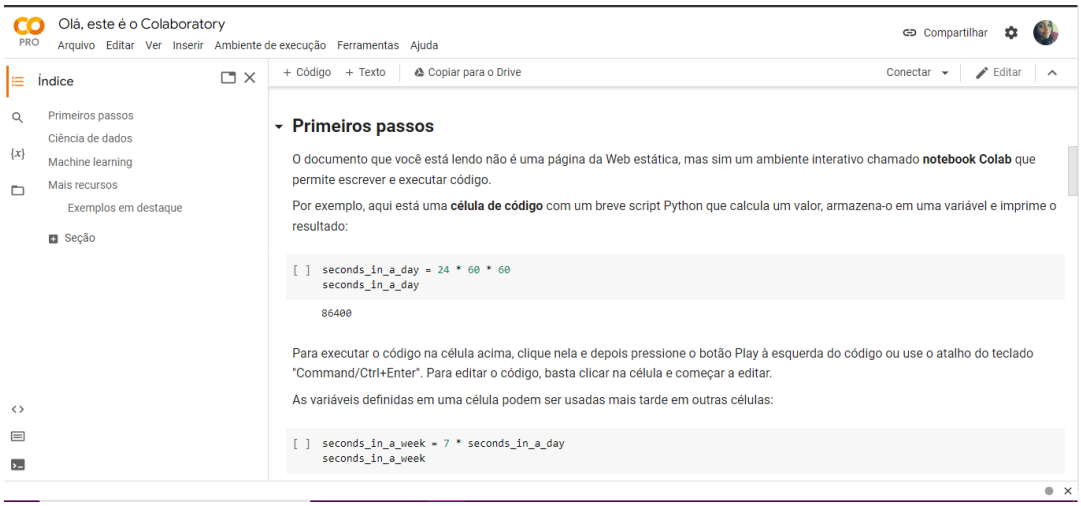
Segundo Hui (2018) o mAP corresponde à média calculada no final do treinamento do *Average Precision* (AP), onde é o resultado da interpolação dos pontos da curva de precisão e revocação coletados durante o treinamento do algoritmo, com isso, essa métrica informa o desempenho do modelo de detecção.

1.8 FERRAMENTAS PARA DESENVOLVIMENTO

O *Google Colaboratoy* ou *Google Colab* é um serviço na nuvem gratuito e hospedado pelo *Google*, usado para tarefas de *Machine Learning* e *Deep Learning* que executa a linguagem Python, com suas bibliotecas. Além de fornecer suporte a instâncias de GPU (*Graphics Processing Units*) gratuitamente e ainda possibilitar a integração com o serviço Google Drive (FRANZ et al, 2021). Além de ser construído com base no Jupyter Notebook, possui recursos de colaboração entre desenvolvedores e a possibilidade de exportar o código construído em extensões **.py**.

Segundo Da Silva (2020), o *Google Colab* possui um mecanismo de divisão de blocos de código no mesmo arquivo, o que facilita comentários de código, compilação de código de forma separada e a possibilidade de elaborar relatórios para análises de dados por exemplo. Na **Figura 16** exibe a interface principal da plataforma**.**

Figura 16 – Interface do *Google Colab*.

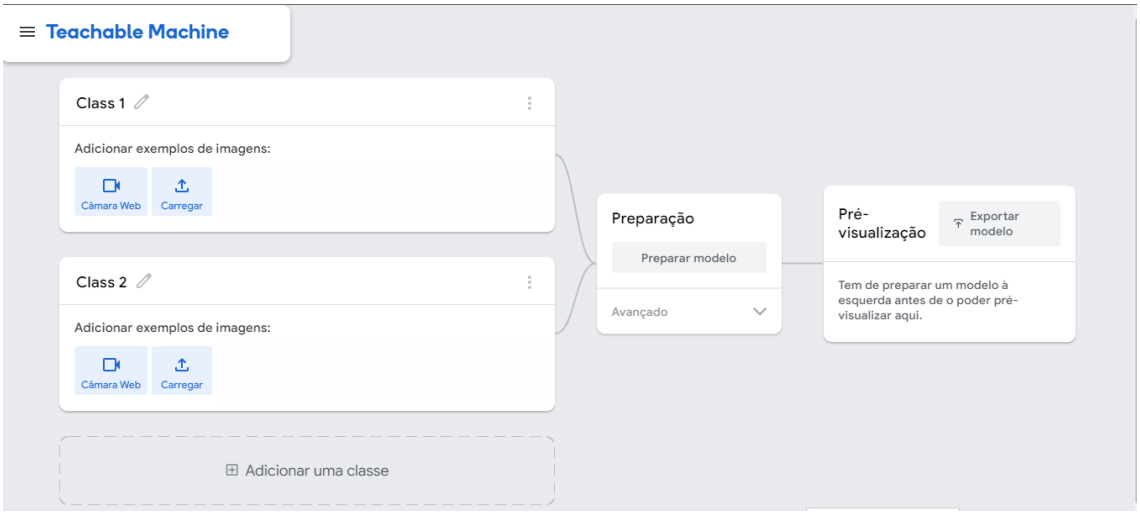


Fonte: Elaborado pela autora

A plataforma *Teachable Machine* é uma ferramenta gratuita baseada na web para criar modelos personalizados de *Machine Learning*, onde utiliza o *framework* TensorFlow.js para treinar e rodar os modelos de forma online pelo navegador (CHEN, 2020).

Segundo Oliveira et al. (2022) essa ferramenta possui uma interface intuitiva baseada em um fluxo de trabalho que permite o *upload* de um conjunto de dados para treinamento do modelo, resultando em uma predição das classes que pode ser testada utilizando a própria câmera do computador ou uma imagem para teste, além de possibilitar a exportação do modelo formato **.h5** para ser carregado em um código em linguagem Python. A **Figura 17** mostra como é a tela inicial de treinamento na ferramenta.

Figura 17 – Tela inicial de treinamento de um modelo na plataforma.



Fonte: (GOOGLE, 2017)

2 METODOLOGIA

O trabalho em questão é uma pesquisa de natureza aplicada e visa realizar uma pesquisa exploratória sobre o material bibliográfico e de laboratório. Serão utilizados os procedimentos técnicos de pesquisa bibliográfica e experimental. Serão utilizados o método de abordagem hipotético-dedutivo e o método de procedimento em sua elaboração. Para coleta de dados é utilizada a observação direta e documentação indireta e a análise e interpretação de seus dados qualitativos.

Inicialmente, serão realizadas pesquisas bibliográficas na área de *Deep Learning*, detecção de objetos, visão computacional e internet das coisas. Para o desenvolvimento prático deste trabalho utilizou-se um *notebook acer* aspire F5, um *Raspberry Pi* 3 modelo B e uma câmera *logitech* c922. Utilizou-se, também, as ferramentas de software *Teachable Machine* e o *Google Collaboratory*.

Após a etapa de planejamento do projeto, consolidou-se uma base de dados representativa de crianças em situações reais imersas em um contexto, coletadas através de frames representativos de vídeos, cujo acesso foi permitido pelos próprios responsáveis pelo vídeo. Para tarefa de detecção de objetos é necessário para cada imagem delimitar a região onde a criança está imersa em uma situação através de ferramentas externas.

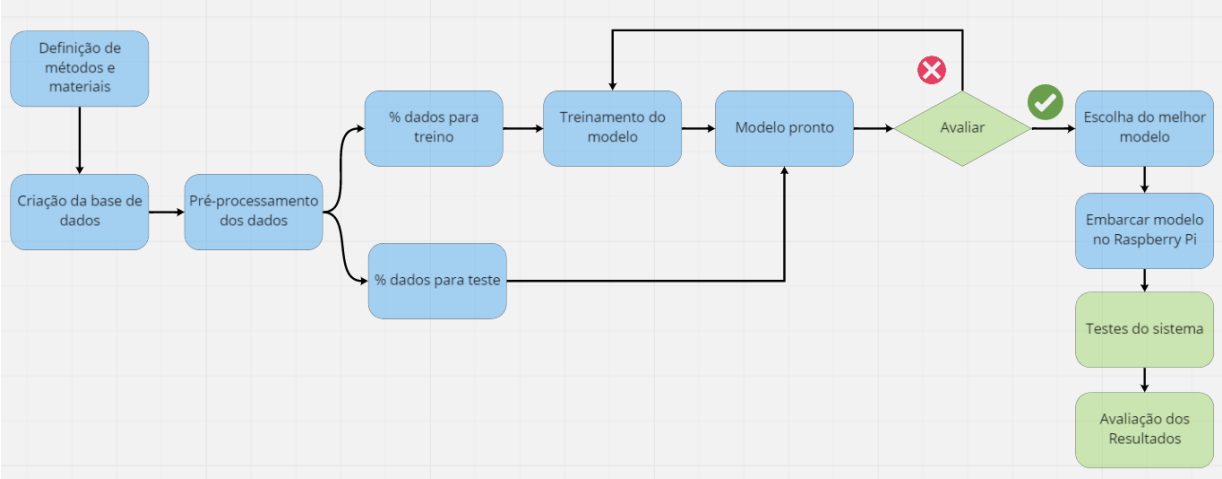
A etapa seguinte será de implementação dos modelos, onde as bases de dados coletadas serão distribuídas em dados de treinamento, validação e teste que serão processadas por modelos de classificação e detecção, onde essas duas abordagens serão avaliadas isoladamente e comparadas a fim de encontrar a melhor solução para o problema proposto. Para a tarefa de classificação, empregou-se o programa *Teachable Machine* para criação do modelo e para a tarefa de detecção de objetos, a arquitetura de rede neural YOLO.

Para cada abordagem, serão utilizados os dados de teste para avaliar o desempenho dos modelos treinados e analisar seus resultados, para encontrar as circunstâncias mais favoráveis e identificar os pontos de melhoria de modo a retreinar o modelo se necessário.

Por fim, o modelo que apresentou melhor desempenho, é embarcado no R*aspberry pi* para realizar testes com o auxílio de uma câmera em direção ao ambiente e analisar se atingiu o objetivo proposto ao acionar um alarme logo após detectar a situação de possível perigo, ou seja, a criança sozinha.

A metodologia descrita pode ser resumida de acordo com o diagrama que se encontra na **Figura 18**.

Figura 18 – Metodologia a ser adotada no projeto.



Fonte: Elaborado pela autora

Para a realização das atividades propostas na metodologia faz-se necessário definir os recursos que serão utilizados para o desenvolvimento e execução do projeto, conforme dispostos na **Tabela 1**.

Tabela 1 – Recursos utilizados para o desenvolvimento do projeto.

Tipo	Quantidade	Descrição	Valor (R$)
Hardware	1	Notebook Acer Aspire F5	4.250,00
	1	Raspberry Pi 3 modelo B	400,00
	1	Câmera Raspberry Pi Ver. 1.3 5 Mp	44,90
Software	-	Teachable Machine	0,00
	-	Google Colab	0,00

Fonte: Elaborado pela autora

É importante ressaltar que o *notebook* descrito está presente apenas para realizar a simulação em ambiente de teste, considerando um ambiente real este item não é incluso no orçamento da implementação.

A seguir serão detalhados os aspectos relacionados a aquisição da base de dados, as tecnologias utilizadas para a elaboração do projeto, a arquitetura escolhida para desenvolvimento do modelo de detecção, as métricas utilizadas e as etapas necessárias para embarcar o modelo no *Raspberry Pi*.

2.1 BASE DE DADOS

Para o problema apresentado neste trabalho, não foi encontrado nenhuma base de dados pronta contendo imagens específicas para o problema. Por se tratar de imagens de crianças imersas em determinado contexto foi necessário a elaboração desse conjunto de imagens.

A tarefa de coleta de imagens foi realizada através da *internet*, utilizou-se de plataformas de Rede Sociais como o *Instagram* e o *YouTube* para obter os vídeos representativos. A separação de *frames* de imagens foi realizada com a ferramenta *VLC Media Player* que é um reprodutor de mídia gratuito e de código aberto para diversos sistemas operacionais (KOVACS, 2021). Onde foi necessário configurar a ferramenta para extrair por padrão 1 *frame* a cada 50 segundos, necessitando adaptar este parâmetro conforme o tamanho do vídeo, pois em algumas situações as imagens extraídas se encontravam repetidas devido ao baixo espaçamento de tempo coletado. Por fim, realizou-se uma verificação manual para remover qualquer indício de duplicidade nas imagens coletadas.

O *dataset* (conjunto de dados) utilizado contém um total de 700 imagens coletadas com contextos diferentes, sendo 350 imagens para crianças em situações de perigo, ou seja, sozinhas perto ou dentro de piscinas e 350 para crianças em situações de não perigo com os seus responsáveis presentes na imagem. Após alcançar este número total de imagens, observou-se que não estava sendo encontrado mais imagens representativas para compor o *dataset*, assim optou-se por permanecer em 700, pois a quantidade de imagens para cada classe já se encontrava balanceada.

O conjunto de dados foi dividido utilizando a proporção indicada em Russel e Norvig (2013), que para este trabalho corresponde em 490 imagens para treinamento correspondendo a 70%, 140 imagens para validação sendo 20% e 70 imagens para teste representando 10%. A **Tabela 2** exemplifica como foi essa divisão para cada tarefa proposta, onde a seleção das imagens se deu por escolha aleatória.

Tabela 2 – Informações sobre a distribuição do *dataset* para cada tarefa proposta.

Tarefa	Quantidade Treino	Quantidade Validação	Quantidade Teste	Total
Detecção	490	140	70	700
Classificação	490	140	70	700

Fonte: Elaborado pela autora

2.2 TECNOLOGIAS UTILIZADAS

As plataformas utilizadas para a realização deste trabalho para auxiliar no desenvolvimento dos algoritmos de detecção e classificação foram o *Google Colaboratory* ou *Google Colab* e o *Teachable Machine*, onde os mesmos se encontram no referencial teórico.

O *Google Colab* foi utilizado para treinar e testar o modelo de detecção através do *framework Darknet*, cuja execução é por comandos de terminal sem a necessidade de uma programação para trabalhar com modelos.

O *Teachable Machine* foi utilizado para treinar o modelo de classificação, onde as imagens foram adicionadas para treinamento e após isso, realizado a exportação do modelo para extrair métricas de desempenho com base nos dados de validação e teste, onde a geração dessas métricas foi feita no *Google Colab*.

Neste trabalho a principal linguagem de programação utilizada foi o Python 3, uma das mais utilizadas nos campos de *Machine Learning* e *Deep Learning*, além de possuir diversas bibliotecas de apoio a estas tarefas. As bibliotecas utilizadas neste trabalho foram *OpenCV*, *Scikit-Learn*, GPIO e *Paho* MQTT.

A biblioteca *OpenCV* (*Open Source Computer Vision Library*), foi utilizada para carregar o modelo treinado, processar as imagens e habilitar a câmera do *Raspberry Pi* para testar o modelo. Segundo Silva, Chaves e Aquino (2012), é uma biblioteca *open-source* (código aberto) usada no campo da visão computacional e processamento de imagens e vídeos, além de ter sido desenvolvida em linguagem C e C++ dando suporte a linguagem Python.

A biblioteca *Scikit-Learn* foi utilizada para gerar as métricas do modelo de classificação treinado na *Teachable Machine*, de modo a validar o desempenho do mesmo. De acordo com Weiand (2018), essa biblioteca também implementa diversos algoritmos de aprendizagem de máquina, como algoritmos de classificações, regressão e outros.

A biblioteca GPIO do Python é usada para controlar as portas de entrada e saída GPIO do *Raspberry Pi* além de ser utilizada para ativar e desativar o circuito de alarme do projeto.

A biblioteca *Paho* MQTT fornece uma classe de cliente para que os aplicativos se conectem a um broker MQTT para publicar mensagens, assinar tópicos e receber as mensagens publicadas, que serão utilizados neste trabalho (PAHO, 2014).

2.3 ARQUITETURAS CONSIDERADAS

Embora a YOLOv4 apresente mais velocidade e melhor capacidade de detecção que a YOLOv3, a arquitetura escolhida para este trabalho foi a YOLOv4-tiny.

Esta arquitetura é uma versão compactada da YOLOv4, onde segundo Wang, Bochkovskiy e Liao (2021) se propõe a ter uma estrutura mais simples, reduzindo parâmetros para se tornar viável o desenvolvimento de aplicações móveis e embarcados. Por possuir uma estrutura menor, a YOLOv4-tiny foi treinada a partir de 29 camadas convolucionais pré-treinadas em comparação com a YOLOv4 que utiliza 137 camadas convolucionais pré-treinadas, com isso possuindo menos camadas para a técnica de *Transfer Learning*. Além de ser cerca de oito vezes mais rápida em FPS (*Frames Per Second*), porém em precisão e desempenho apresentou cerca de 2/3 da YOLOv4 quando testado com o conjunto de dados MS COCO (WANG, BOCHKOVSKIY e LIAO, 2021).

Para este trabalho, julga-se ser mais relevante a velocidade da detecção em tempo real do que a acurácia, pois tem-se como objetivo utilizar o modelo para detectar crianças perto ou dentro de piscinas de forma rápida para evitar acidentes.

2.4 MÉTRICAS DE DESEMPENHO

Após os treinamentos do modelo de classificação e detecção, se faz necessário a utilização das métricas de desempenho abordadas no referencial teórico para avaliar se os modelos estão apresentando os resultados esperados.

Para a tarefa de detecção foi utilizado como métrica principal o mAP, que avalia a precisão das caixas delimitadoras detectadas no algoritmo para o conjunto de teste e de validação, onde esses valores foram fornecidos através do próprio *framework* utilizado para treinamento. Para fins de comparação com a tarefa de classificação, foi utilizado também as métricas de precisão, revocação e F1-score, onde para classificação foi utilizado a biblioteca *Scikit-Learn* da linguagem de programação Python para se calcular estas métricas.

2.5 EMBARCAR O MODELO NO RASPBERRY PI

Por fim, o melhor modelo da abordagem escolhida foi embarcado em um *Raspberry Pi* para realizar a predição em tempo real e avaliar seus resultados. Para isto, foi utilizado

novamente a linguagem Python para carregar o modelo e receber imagens (*frames*), obtidas de uma câmera conectada ao sistema embarcado, para realizar a predição e retornar se existe alguma criança em situação de perigo.

Além da informação do modelo, foi implementado também um retorno visual, para ser possível realizar alguma ação de imediato. Para este trabalho, foi desenvolvido um circuito de alarme para acionar quando ocorrer a predição, e uma arquitetura de *Publisher* e *Subscriber* através do protocolo MQTT, para enviar a mensagem no mesmo instante.

Para estes testes, foi simulado que está se utilizando uma câmera de vigilância em uma piscina, onde a câmera conectada ao *Raspberry Pi* será ajustada para capturar um vídeo em outro computador, e assim seguir o fluxo normal da aplicação.

Esta etapa busca consolidar toda a implementação feita durante este trabalho, para simular a possibilidade de uma câmera detectar que a criança se encontra sozinha em uma piscina, ou seja, em situação de perigo.

3 IMPLEMENTAÇÃO

Nesta seção será demonstrado os processos descritos na metodologia para a construção do projeto.

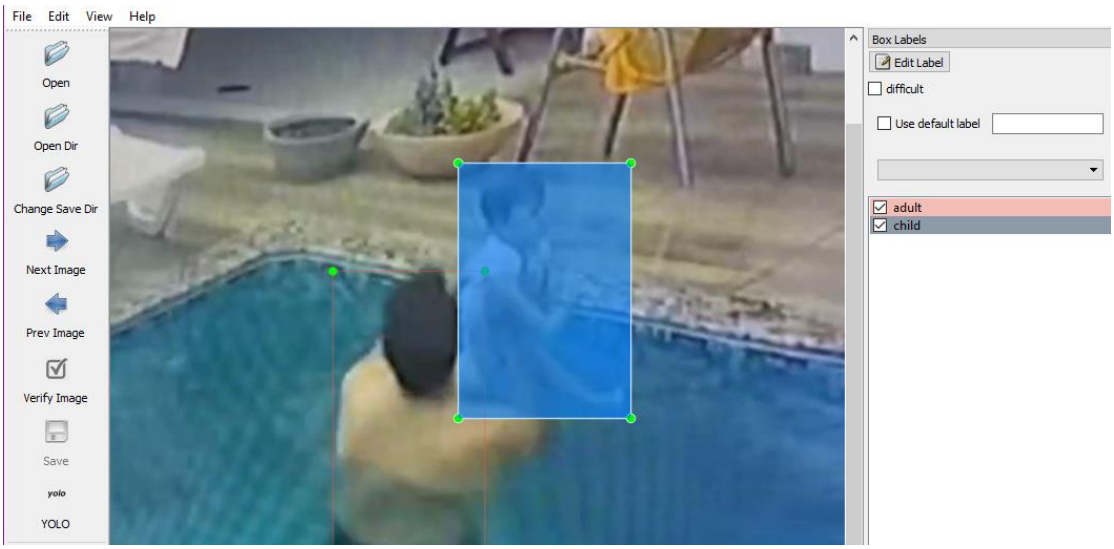
3.1 MÉTODO DE DETECÇÃO DE OBJETOS EM IMAGENS

3.1.1 PRÉ-PROCESSAMENTO DOS DADOS

A primeira etapa para este método consiste em realizar as anotações das imagens coletadas em formatos adequados para que a CNN escolhida possa realizar a tarefa de aprendizado. Para isso, a ferramenta *open-source LabelImg* (LIN, 2019) escrita em linguagem Python que funciona como uma ferramenta de anotações gráficas, foi utilizada para informar as caixas delimitadoras (*Bounding Boxes*) dos objetos nas imagens, onde essas anotações foram exportadas no formato **.txt** compatível com a YOLO, especificamente a YOLOv4-tiny.

As imagens foram separadas por duas classes chamadas *Child* e *Adult*, onde foram inseridas em dois contextos distintos chamados *Danger* e *Safe*. O primeiro só continha imagens de crianças sozinhas perto ou dentro de uma piscina, evidenciadas através dos *bounding boxes* e no segundo imagens onde as crianças se encontravam na presença dos responsáveis, evidenciando a segurança da criança ao detectar a presença de um adulto por perto. Esta análise de contexto será realizada por código ao detectar a presença de um adulto, onde será entendido se a criança está segura ou não. A **Figura 19** ilustra a execução das anotações nas imagens.

Figura 19 – Seleção onde se localiza a criança e o adulto na imagem utilizando a ferramenta *LabelImg*.



Fonte: Elaborado pela autora

Após a etapa de divisão das imagens com suas devidas anotações em imagens de treino, validação e teste de forma randômica, é necessário preparar o ambiente onde será treinado o modelo. Neste trabalho será utilizado o *Google Colaboratory*, para executar os comandos necessários para o treinamento e teste do modelo.

3.1.2 CRIAÇÃO DO MODELO

Após a consolidação da base de dados e escolha da arquitetura a ser utilizada neste trabalho, a primeira etapa é treinar o modelo da YOLOv4-tiny e para isso será utilizado o *framework Darknet*, que é usado como base para desenvolver o modelo de detecção.

Com isso, alteraram-se as configurações de execução do *notebook* do *Google Colaboratory* para utilizar a GPU, e assim ser possível treinar o modelo com as placas de vídeo dedicadas do *Google Colaboratory*, tornando mais rápido o treinamento. Assim, bastou-se clonar o repositório da *Darknet* e instalar as dependências necessárias para executar o treinamento.

Para realizar o treinamento é necessário baixar os pesos da última camada pré-treinada da YOLOv4-tiny para realizar a operação de *Transfer Learning* e configurar dois arquivos principais: um arquivo **.data** que contém o número de classes total e o diretório dos arquivos de treinamento e teste, e um arquivo **.cfg** onde se encontra as configurações da arquitetura a ser utilizada. Para este trabalho optou-se por definir os seguintes hiper parâmetros de treinamento para a YOLOv4-tiny encontrados na **Tabela 3**.

Tabela 3 - Informações sobre os parâmetros passados para a YOLOv4-tiny.

Batch	64
Subdivision	2
Altura da imagem	416
Largura da imagem	416
Max Batches	4000

Fonte: Elaborado pela autora

Para os valores do tamanho de *batch*, largura e altura da imagem, optou-se por deixar os valores padrões da YOLOv4-tiny, faltando ajustar os seguintes parâmetros *subdivision*, *max batches*. Com isso *subdivision* é em quantos minilotes o tamanho do *batch* é dividido, no caso

deste trabalho como o *batch* é de 64 imagens por iteração e a *subdivision* é de 2 minilotes, isso significa que serão 2 minilotes com 32 imagens cada que será enviado para a GPU para processamento. Assim quanto menor o tamanho da *subdivision* mais memória será consumida da GPU, porém será carregado mais imagens por cada iteração.

Max batches é o número máximo de épocas que o modelo irá realizar o treinamento e segundo Bochkovskiy, Wang e Liao (2020) o modo de calcular esse número máximo é através da seguinte fórmula:

$$MaxBatches = 2000 * número\ de\ classes \quad (4)$$

Para este projeto como o número de classes abordadas são 2 (adulto e criança) foram necessárias no máximo 4000 iterações.

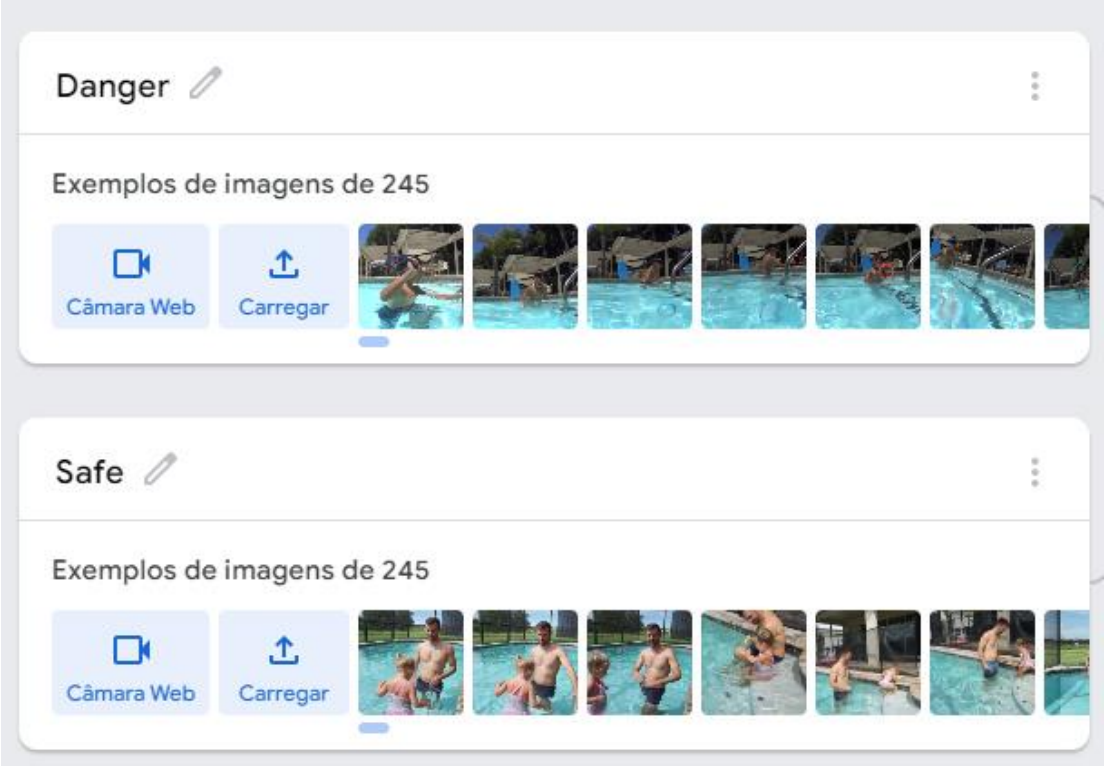
Com isso por padrão, a *Darknet* executa o modelo e salva os pesos a cada 100 épocas e em seguida a cada 1000 épocas, tornando possível a reexecução a partir de uma determinada época salva.

3.2 MÉTODO DE CLASSIFICAÇÃO EM IMAGENS

A primeira etapa para este método consiste em separar as imagens em duas classes chamadas *danger* e s*afe*, onde a primeira contém imagens de crianças sozinhas e a segunda contém imagens de crianças com algum responsável por perto. Esta nova proposta foi feita, pois, a tarefa de classificação visa detectar somente uma das classes nas imagens, caso seja escolhido por manter as classes criança e adulto, existirão imagens onde será necessário que o modelo detecte duas classes fugindo do tema proposto da tarefa. Assim, a classe *safe* representa uma junção da existência de criança e adulto na mesma imagem, já a de *danger* representa somente a criança, mas seu nome foi mudado para um contexto mais apropriado.

Com isso o conjunto de dados para treino foi adicionado na plataforma *Teachable Machine*, uma ferramenta baseada na *web* para criar modelos de classificação de *Machine Learning* sem precisar de conhecimentos técnicos especializados (CHEN, 2020). Essa ferramenta recebe um conjunto de imagens separadas em suas devidas classes conforme a **Figura 20**, redimensionando-as para o tamanho 224 × 224 de acordo sua configuração padrão de treinamento.

Figura 20 – Separação das classes de imagens na ferramenta *Teachable Machine*.



Fonte: Elaborado pela autora

Para realizar o treinamento do modelo na ferramenta é necessário fornecer alguns parâmetros já pré-definidos na *Teachable Machine* como a quantidade de épocas e o tamanho do lote. A quantidade de épocas se refere a quantidade de vezes que o modelo iterou no conjunto de treinamento e para fins de comparação com o método de detecção, optou-se por colocar 4000 épocas no treinamento. O tamanho do lote é o conjunto de imagens que será utilizado em cada iteração de treinamento, optou-se por colocar o mesmo tamanho no treinamento da YOLOv4-tiny, que foi de 64 imagens.

Como na ferramenta não há uma forma de testar via lotes de imagens, foi necessário fazer a exportação do modelo no formato **.h5** para realizar via código experimentos com o conjunto de validação e teste, e calcular métricas para o modelo.

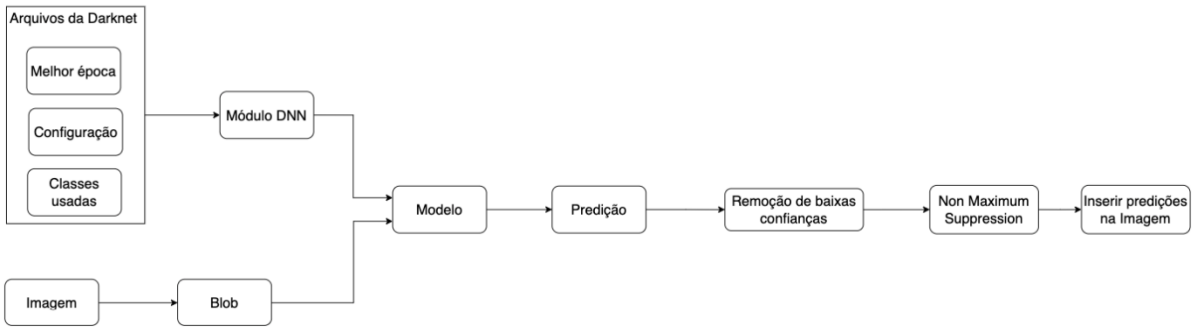
3.3 DETECÇÃO EM TEMPO REAL NO RASPEBERRY PI

Nesta seção será descrito todas as etapas necessárias para possibilitar o uso da YOLOv4-tiny para realizar a detecção em tempo real em um sistema embarcado, neste caso o *Raspberry Pi* 3.

3.3.1 INTEGRAÇÃO DA YOLO COM OPENCV

Para ser possível utilizar o modelo criado na seção 3.1.2 em uma aplicação em linguagem *Python*, é necessário primeiramente carregá-lo utilizando uma ferramenta externa e então realizar a predição, para este fim utilizou-se a biblioteca *OpenCV* (OPENCV, 2019). A **Figura 21** apresenta todo o fluxo necessário, desde carregar o modelo até de fato ter a predição, suas etapas serão descritas abaixo.

Figura 21 – Sequência de etapas necessárias para obter a predição em uma imagem.



Fonte: Elaborado pela autora

Para carregar o modelo são necessários três arquivos: a configuração da rede **.cfg**, a melhor época treinada (neste caso, de 4000) e o último com as classes utilizadas. Estes três arquivos serão passados para a função presente no módulo de *Deep Learning* do *OpenCV* para criar uma variável que representasse o modelo e permitisse realizar predições.

Em cada imagem recebida, é necessário um pequeno pré-processamento para adequá-la ao formato reconhecido pelo modelo carregado pelo *OpenCV*, para isto utiliza-se uma função da própria biblioteca para obter um *blob* da imagem, isto é, um vetor de 4 dimensões contendo a própria imagem, após normalizar seus *pixels* para entre 0 e 1 e redimensioná-la para 416 x 416, e suas propriedades como número de canais e altura e largura (ROSEBROCK, 2017).

Este *blob* é passado para o modelo realizar a predição, realizando as etapas já mencionadas neste trabalho, e a saída consiste em todas as detecções realizadas, retornando uma lista de coordenadas e níveis de confiança, sendo então necessário manualmente filtrar somente as predições com maior valor de confiança.

A confiança padrão escolhida será de 0,5 para este código, assim, todas as predições que apresentarem um valor abaixo, serão consideradas incorretas e descartadas na visualização final. Para as detecções remanescentes, aplicou-se o filtro do *OpenCV* de *Non Maximum Supression* (PRAKASH, 2021), para remover possíveis predições que estão se sobrepondo e assim ter uma imagem mais limpa.

Após estas etapas para remover predições indesejadas, é necessário apenas percorrer as remanescentes e desenhar um retângulo a partir das coordenadas na imagem, com o *OpenCV*, com o texto informando a classe e a confiança obtida.

Para uma captura em tempo real, o processo é extremamente semelhante, necessário apenas utilizar novamente o *OpenCV* e a função *Video Stream* da biblioteca *Imutils* da linguagem Python que usa a biblioteca *Picamera* (PICAMERA, 2016) para a extração de *frames* da câmera do *Raspberry Pi* para trabalhar com vídeos. Dessa forma, cada um é enviado para o fluxo descrito anteriormente para o modelo detectar as classes presentes, este processo se repete até o fim do vídeo.

3.3.2 ARQUITETURA DE PUBLISH/SUBSCRIBE PARA MQTT

Além de realizar a detecção, é interessante também externalizar a informação obtida, para ser possível realizar alguma ação. Para isto, será utilizado o protocolo de mensageria MQTT para enviar alerta para celulares ou notebooks, via rede *Wi-Fi*, para informar algum problema.

Seguindo a arquitetura de PubSub (*Publisher* & *Subscriber*), será utilizado um *broker* online e gratuito disponibilizado pela empresa EMQX (EMQX, 2013) para facilitar o desenvolvimento e focar na lógica do problema. O endereço e porta utilizado está descrito na **Tabela 4**, com o tópico escolhido para o problema.

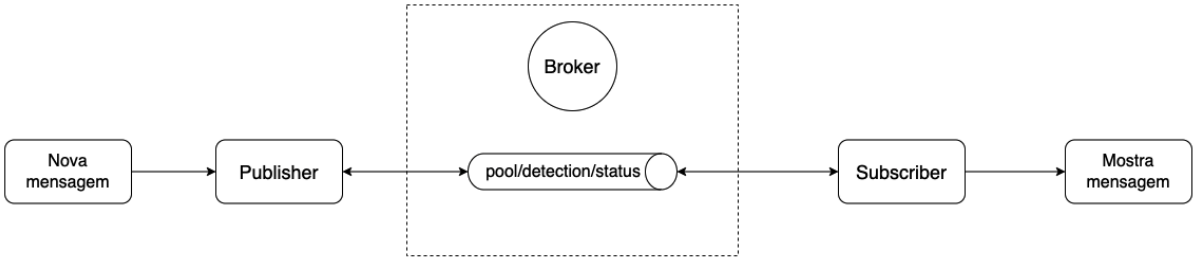
Tabela 4 - Dados de conexão do *broker* MQTT.

Host	broker.emqx.io
Porta	1883
Tópico	pool/detection/status

Fonte: Elaborado pela autora

A implementação do MQTT será feita na linguagem Python, através da biblioteca *Paho* MQTT (PAHO, 2014), que contém toda a lógica necessária para construir o padrão PubSub. Para este trabalho, o *Subscriber* basicamente irá se conectar no *broker*, se inscrever no tópico descrito acima e aguardar as mensagens enviadas. O *Publisher* irá de fato enviar/publicar a mensagem gerada, ou seja, informar quando uma criança está sozinha perto de uma piscina, ou seja, em uma situação de perigo, que será enviado para o mesmo tópico. A **Figura 22** ilustra a arquitetura descrita para a troca de mensagem.

Figura 22 – Fluxo de dados adotado do MQTT PubSub.



Fonte: Elaborado pela autora

O *Publisher* precisa estar executando no mesmo ambiente onde está sendo feito a predição, neste caso o *Raspberry Pi*, para ser possível ele receber o resultado e publicar em um tópico. O *Subscriber* pode estar sendo executado em qualquer máquina que possui acesso a rede *Wi-fi*, permitindo assim exibir a informação obtida em diversas categorias de aparelhos eletrônicos.

3.3.3 ARQUITETURA DA YOLO COM MQTT

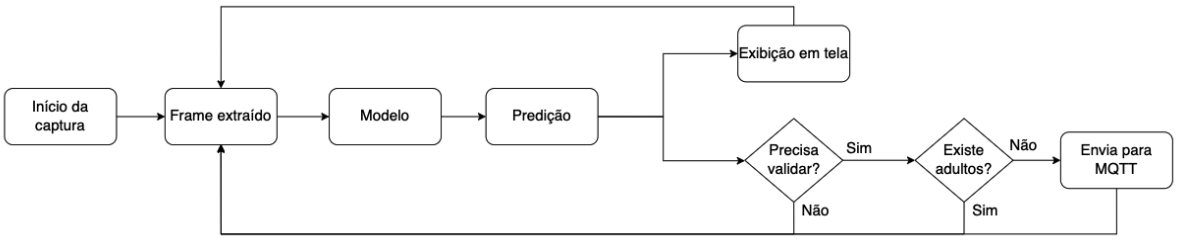
Após a criação da arquitetura MQTT, resta apenas integrar com a detecção da YOLO e o *OpenCV* e publicar uma mensagem caso não encontre um adulto, ou seja, caso a criança esteja em situação de perigo na piscina.

Em cada *frame* processado, é retornado também uma lista de classes identificadas para poder analisar se existe algum adulto na imagem. Durante o decorrer dos testes, observou-se que ao utilizá-los em um sistema embarcado, como o *Raspberry Pi*, exige um grande processamento do hardware, e caso demore muito para analisar os *frames*, poderia perder informações importantes.

Para evitar este problema, escolheu-se o menor número possível para informar em que momento deve-se validar a condição de envio da mensagem, alguns testes demonstraram que o número um pode ser utilizado sem muitos problemas, assim, a cada um frame será validado se existe a classe adulto dentro do conjunto de classes previstas neste intervalo de tempo. Caso a validação seja negativa, o MQTT publica no tópico o texto “criança em perigo”, caso seja positivo, não será feito nenhum tratamento, ao final da validação o conjunto é reiniciado e o processo se repete até finalizar a captura.

A **Figura 23** exemplifica o fluxo da predição com a condição para a mensagem MQTT, percebe-se ao final da sequência o diagrama retorna para a etapa de extração de *frame*, representando que será executado continuamente até o final da captura.

Figura 23 – Fluxo utilizado para avaliar se precisa enviar a mensagem para o MQTT ou não.



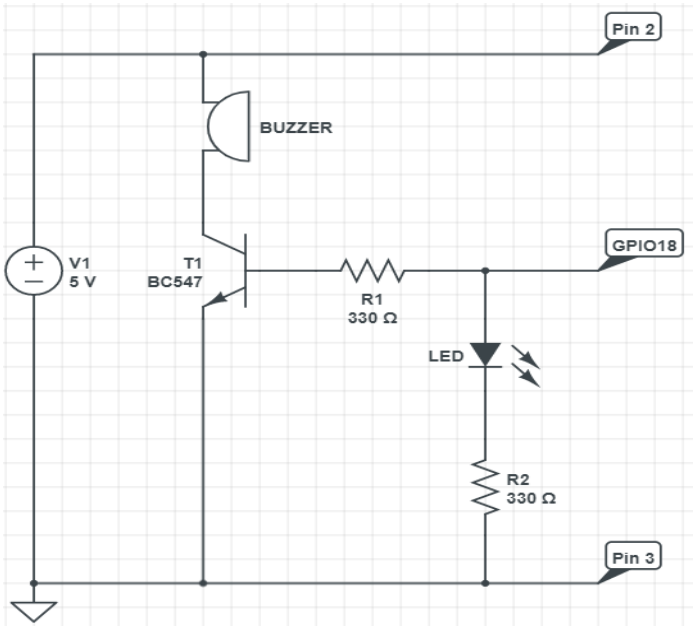
Fonte: Elaborado pela autora

3.3.4 IMPLEMENTAÇÃO DO CIRCUITO DE ALARME

Além de enviar o resultado da detecção para dispositivos externos através do protocolo MQTT, implementou-se também um retorno visual e sonoro para saber em que momento não foi detectado um adulto e consequentemente ocasionando uma situação de perigo.

Este circuito tem como propósito ser ativado e desativado pelo próprio *Raspberry Pi*, sem interferência externa. Para este fim utilizou-se um LED com um resistor de 330 Ω para limitar sua corrente, um *Buzzer* com tensão de trabalho de 3,5 até 5,5V ativo para emitir sinal sonoro com uma frequência pré-determinada, um resistor de 330 Ω o acompanha para regular seu sinal sonoro à um nível audível e um transistor NPN BC547 para assegurar que o *Buzzer* está recebendo a tensão correta de operação. Seu esquemático foi criado na ferramenta CircuitLab (CIRCUITLAB, 2016) e pode ser constatado na **Figura 24**.

Figura 24 – Esquemático do circuito de alarme.



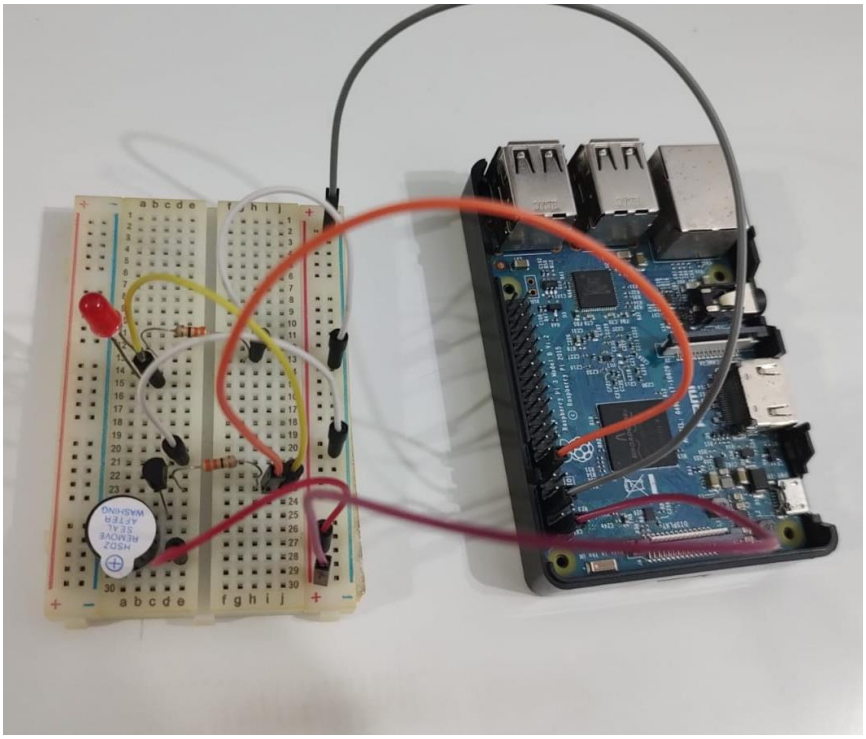
Fonte: Elaborado pela autora

Serão utilizados três pinos para o circuito: o pino 6 que representa o GND do circuito, o pino 2 que representa a fonte DC de 5 *volts* para energizar o *Buzzer,* e o pino 18 para acionar tanto o LED quanto o *Buzzer* quando necessário, além da pinagem destinada à câmera do *Raspberry Pi*.

Para manipular a pinagem na linguagem Python, será utilizado a biblioteca GPIO (GPIO, 2013) para ativar ou desativar o pino 18. Suas condições serão simples: no momento que o modelo detectar que não existe um adulto em um determinado conjunto de *frames*, será enviado uma mensagem para o MQTT e, em simultâneo, será ativado o pino 18 para acender o LED e ligar o *Buzzer,* e após três segundos será desligado e o processo se repetirá até o fim da captura.

Este circuito foi implementado fisicamente para ser possível realizar os testes necessários, seguindo o esquemático *online* apresentado anteriormente, o resultado pode ser visto na **Figura 25**.

Figura 25 – Circuito de alarme.



Fonte: Elaborado pela autora

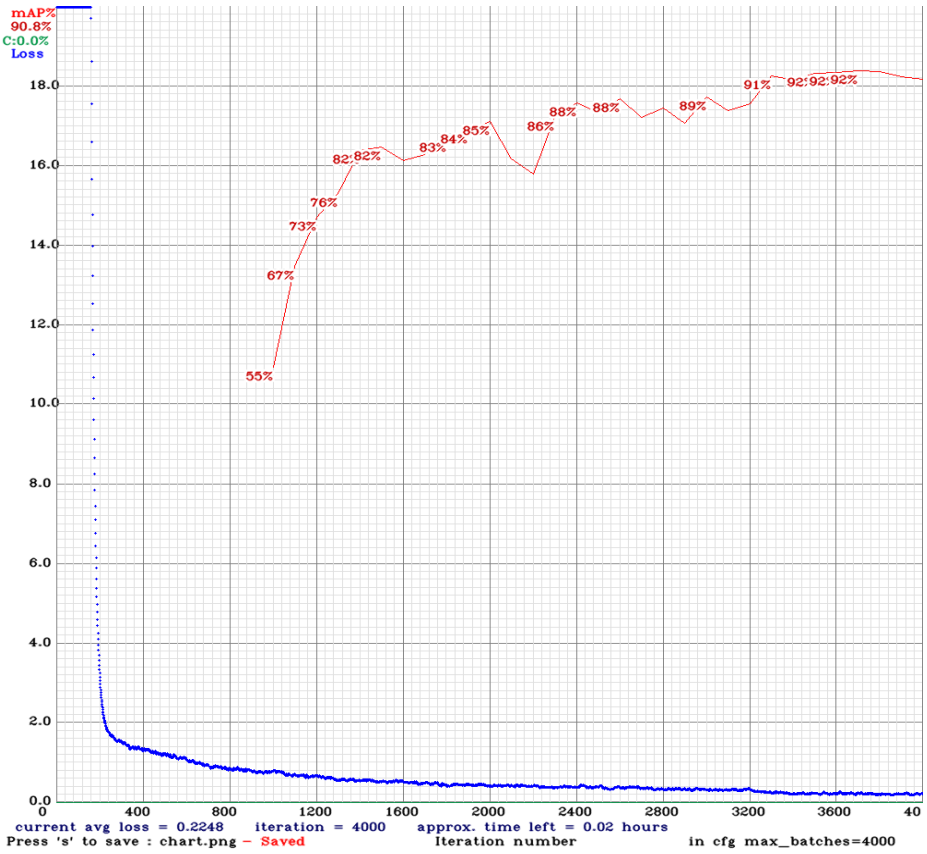
4 RESULTADOS

Conforme descrito na metodologia e na implementação, realizou-se o treinamento dos modelos para cada abordagem com o *dataset* personalizado para cada um. Para abordagem de detecção foi utilizado a plataforma *Google Colab* e arquitetura da Yolov4-tiny e para classificação utilizou-se a plataforma *Teachable Machine*. Os resultados obtidos em cada um serão descritos abaixo.

4.1 DETECÇÃO

Para esta abordagem será utilizado como principal métrica o mAP (M*ean Average Precision*), empregado para avaliar problemas de detecção de objetos, mas também será levantado a precisão, revocação e *F1-score*, para avaliar os resultados obtidos a partir do conjunto de validação e teste, estas métricas são calculadas através do próprio framework. A **Figura 26** apresenta o comportamento do modelo no decorrer do treinamento com o conjunto de validação.

Figura 26 – Curva *loss* e mAP da YOLOv4-tiny.



Fonte: Elaborado pela autora

É possível notar que durante o treinamento a cada aumento de épocas o valor do mAP aumentou, chegando ao máximo de 92%, à medida que o *average loss* diminuiu chegando a 0.2248, mostrando que para o conjunto de dados de validação o modelo obteve um bom desempenho.

O melhor modelo obtido foi alcançado em 4000 épocas com duração de 2 horas de treinamento, a **Tabela 5** sumariza as métricas obtidas para este nos conjuntos de validação e de teste.

Tabela 5 - Resultado das métricas obtidas para o modelo de detecção.

Dataset	**Precisão**	**Revocação**	**F1-Score**	**mAP**
Validação	0,84	0,87	0,86	0,908
Teste	0,88	0,90	0,89	0,895

Fonte: Elaborado pela autora

Esses valores apresentados foram retirados do próprio resultado do *framework Darknet* para os conjuntos de validação e teste, onde nas **Figuras 27** e **28** é identificado as porcentagens de acertos para a detecção de uma criança e um adulto, sendo 93,88% de acerto para o conjunto de validação e 92,19% para o conjunto de teste na detecção de uma criança. Esses resultados mostram que o modelo criado tem uma maior chance de detectar uma criança, facilitando na hora de classificar se ela está em perigo (sozinha) ou não.

Figura 27 – Resultado das métricas para a YOLOv4-tiny no conjunto de validação.

```
 calculation mAP (mean average precision)...
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
140
 detections_count = 522, unique_truth_count = 212
class_id = 0, name = child, ap = 93.88%          (TP = 127, FP = 17)
class_id = 1, name = adult, ap = 87.77%          (TP = 58, FP = 17)

 for conf_thresh = 0.25, precision = 0.84, recall = 0.87, F1-score = 0.86
 for conf_thresh = 0.25, TP = 185, FP = 34, FN = 27, average IoU = 65.66 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.908231, or 90.82 %
Total Detection Time: 4 Seconds
```

Fonte: Elaborado pela autora

Figura 28 - Resultado das métricas para a YOLOv4-tiny no conjunto de teste.

```
 calculation mAP (mean average precision)...
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
72
 detections_count = 248, unique_truth_count = 110
class_id = 0, name = child, ap = 92.19%          (TP = 65, FP = 5)
class_id = 1, name = adult, ap = 86.89%          (TP = 34, FP = 8)

 for conf_thresh = 0.25, precision = 0.88, recall = 0.90, F1-score = 0.89
 for conf_thresh = 0.25, TP = 99, FP = 13, FN = 11, average IoU = 68.02 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.895393, or 89.54 %
Total Detection Time: 3 Seconds
```

Fonte: Elaborado pela autora

4.2 CLASSIFICAÇÃO

Para esta abordagem será utilizado como principal métrica o *F1-score*, empregado para avaliar problemas de aprendizado de máquina, além de ser levantando a precisão e revocação para avaliar os resultados obtidos para o conjunto de validação e testes. Estas métricas foram calculadas usando a biblioteca *Scikit Learn* do Python usada para gerar métricas de algoritmos (PEDREGOSA et al, 2011). A **Tabela 5** sumariza as métricas obtidas.

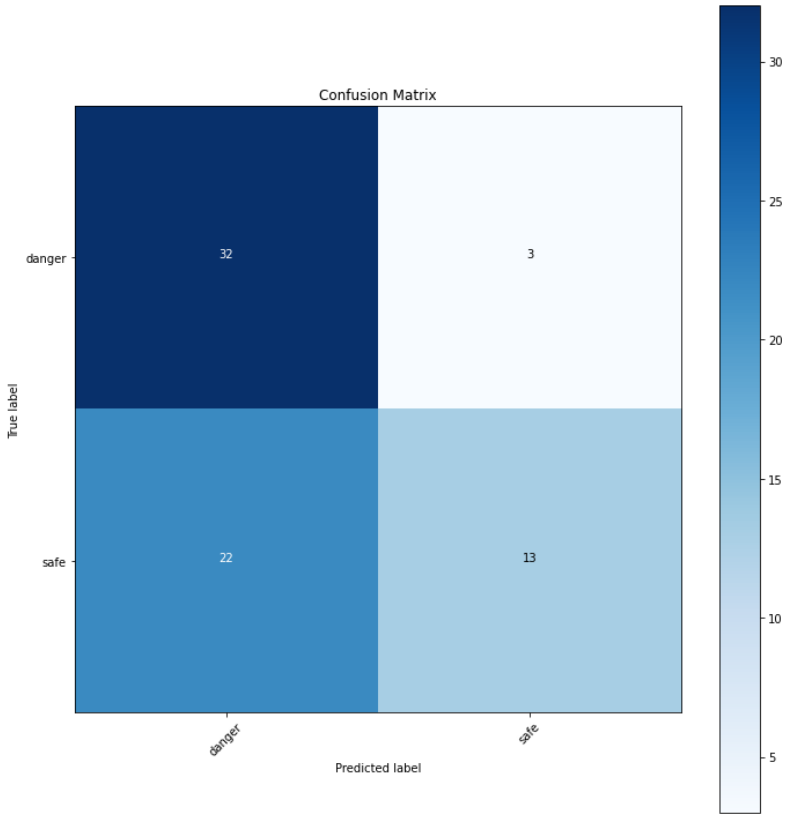
Tabela 6 - Resultado das métricas obtidas para o modelo de classificação.

Dataset	**Precisão**	**Revocação**	**F1-Score**
Validação	0,69	0,60	0,56
Teste	0,70	0,64	0,61

Fonte: Elaborado pela autora

É possível notar que o modelo teve um desempenho um pouco acima da média de 0,5, ou seja, não obteve um resultado satisfatório alcançando apenas um *F1-score* de 0,61 para o conjunto de teste. A matriz de confusão representada na **Figura 29** desse modelo foi obtida para observar o comportamento gerado no modelo.

Figura 29 – Matriz de confusão do modelo de classificação.



Fonte: Elaborado pela autora

Podemos observar que o modelo gerou muitos falsos positivos, ou seja, em 22 imagens o modelo entendeu que uma situação de *safe* era de *danger*, mas em apenas 3 imagens ocorreu ao contrário. Isso significa que o modelo na maioria das vezes irá entender que a criança estará em estado de perigo mesmo estando acompanhada de um responsável.

4.3 DETECÇÃO COM YOLO E OPENCV

Após analisar as métricas obtidas nas seções 4.1 e 4.2, percebe-se que o modelo de detecção apresentou melhores resultados em F1-score, além de ter apresentado cerca 89% de acerto com os dados de teste em mAP, que é uma métrica importante em termos de detecção de objetos. Como o problema deste trabalho é detectar em tempo real e de acordo com as métricas apresentadas, decidiu-se adotar o modelo de detecção para realizar a integração da *Darknet* com o *OpenCV* através da linguagem Python, conforme descrito na seção 3.3.1, para observar o desempenho obtido.

Para os primeiros testes, utilizaram-se vídeos, coletados previamente, como argumento para o código desenvolvido para observar se de fato será detectado as classes de crianças e adultos. Como já era esperado, a YOLOv4-tiny conseguiu detectar rapidamente cada *frame* do vídeo e conseguiu detectar as classes informadas, a **Figura 30** apresenta uma imagem tirada de um vídeo que está sendo processado pelo algoritmo, onde o mesmo pode ser encontrado em (VÍDEO1, 2022).

Figura 30 – Captura feita de um vídeo disponível no Youtube.

Fonte: Elaborado pela autora

É importante ressaltar que o objetivo da YOLOv4-tiny é ter uma taxa de FPS alta para ser possível uma detecção em tempo real ao invés de ter uma acurácia alta, ou seja, em determinados momentos observa-se que em alguns *frames* o modelo pode não conseguir detectar com sucesso as classes, devido à posição da pessoa, se ela está de costa ou outros, mas logo em seguida o modelo volta a detectar novamente.

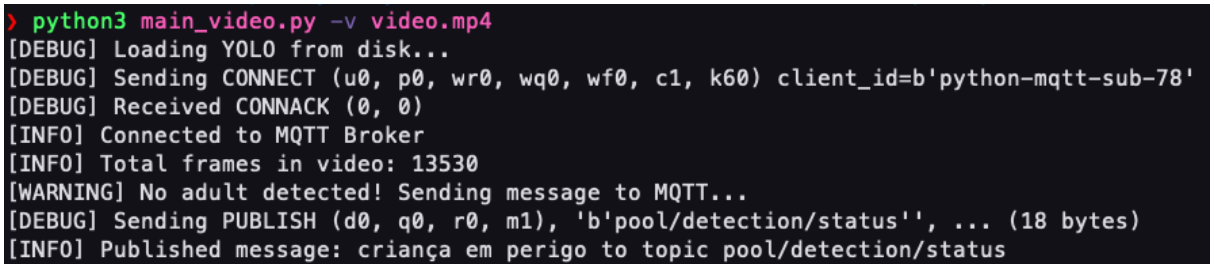
Outro ponto de atenção é a capacidade do *hardware* para detectar o *frame* e, em simultâneo, exibir o vídeo, caso esse teste seja feito em um dispositivo com muita baixa capacidade, é provável que a YOLOv4-tiny não apresente um desempenho satisfatório e assim ficando a impressão de que o modelo não tem um bom desempenho, quando, na verdade é um problema relacionado ao *hardware*. Para estes tipos de testes é importante então ao menos um *hardware* mediano para avaliarmos como está a detecção em vídeo.

4.4 ENVIO E RECEBIMENTO DE MENSAGEM COM MQTT

Para testar o uso do protocolo MQTT, pode ser feito tanto em um vídeo quanto em uma captura em tempo real, com a única regra de não apresentar adultos nos *frames*.

Em qualquer um dos casos, é necessário apenas executar o algoritmo desenvolvido que está integrado a YOLOv4-tiny, *OpenCV* e o *Paho* MQTT para publicar em um tópico, a **Figura 31** apresenta o *log* de informações geradas pelo código, comprovando a publicação do tópico.

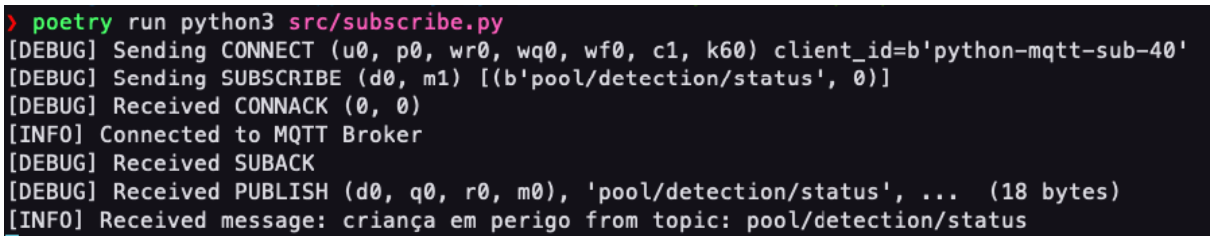
Figura 31 – *Logs* do algoritmo informando o processamento realizado.



Fonte: Elaborado pela autora

Podemos observar que está sendo informado o passo a passo do algoritmo: primeiramente carrega o modelo a partir dos arquivos, se conecta ao *broker* MQTT e depois inicia o processamento de cada *frame* do vídeo, que neste caso apresenta 13530. No momento que o algoritmo validou que não encontrou um adulto em um determinado conjunto de *frames* é preparado uma mensagem para ser publicada no tópico "*pool*/*detection*/*status*" com o texto "criança em perigo", conforme descrito na imagem. Esta mensagem será recebida por outro código que está inscrito neste mesmo tópico, conforme observado na **Figura 32**.

Figura 32 – *Logs* do algoritmo inscrito no tópico.



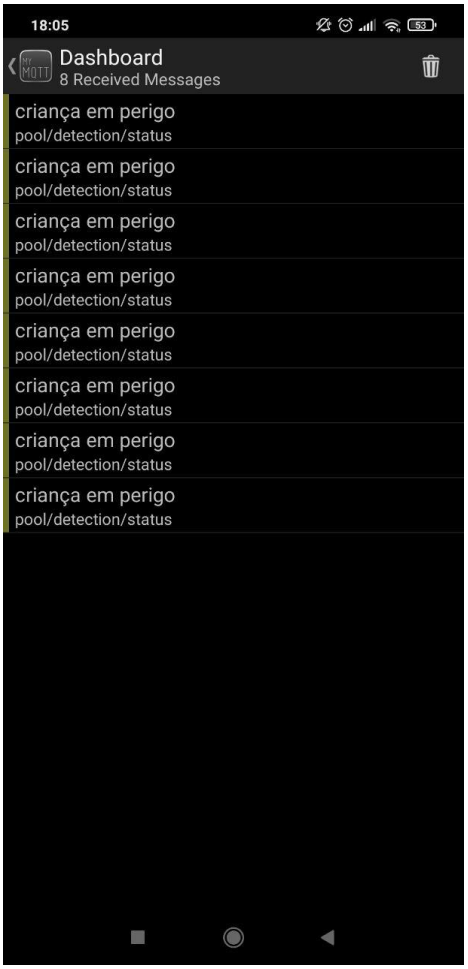
Fonte: Elaborado pela autora

Este algoritmo, conforme descrito na seção 4.3.3, é responsável por somente se inscrever no tópico esperar e mostrar a mensagem recebida, para demonstrar o funcionamento da aplicação. Podemos observar que ele também se conecta ao *broker* MQTT para depois iniciar sua execução, e, logo em seguida, recebe uma mensagem no tópico "*pool*/*detection*/status" e exibe o seu conteúdo, com o texto "criança em perigo".

Para ter um retorno visual que o MQTT estava de fato publicando tópicos simultaneamente a detecção, utilizou-se o aplicativo *Android MyMQTT*, onde foi informado o

broker e o tópico para se inscrever, assim em tempo real é possível receber a mensagem publicada pelo algoritmo, conforme pode ser observado na **Figura 33**, comprovando o funcionamento deste fluxo em um aplicativo de dispositivo móvel.

Figura 33 – *Logs* do algoritmo no aplicativo inscrito no tópico.



Fonte: Elaborado pela autora

4.5 ACIONAMENTO DO CIRCUITO DE ALARME

Os testes do circuito foram realizados em conjunto com o envio de mensagens do MQTT, pois sua ativação será no mesmo instante, no momento que o algoritmo não detectar um adulto em um determinado *frame*.

Foi possível ativá-lo e desativá-lo sem muitos problemas pois a biblioteca GPIO fornece funções para passar valores de “HIGH” ou “LOW” para determinado pino. O único ponto de atenção é em quanto tempo deve-se permanecer aceso o circuito para alertar que existe uma criança em perigo.

Durante os testes, observou-se que um acionamento de três segundos apresentou uma experiência satisfatória, visto que, caso precise, logo em seguida será ativado novamente caso não tenha um adulto presente. Assim, no momento que é detectado somente a criança no *frame*, é enviado a mensagem para o MQTT e simultaneamente aciona o circuito de alarme, desativando-o após três segundos, para depois repetir o ciclo de predição até finalizar a captura. A **Figura 34** exibe o circuito aceso e a mensagem enviada enquanto é executado os testes.

Figura 34 – Teste de acionamento do circuito e envio de mensagem para o MQTT.

Fonte: Elaborado pela autora

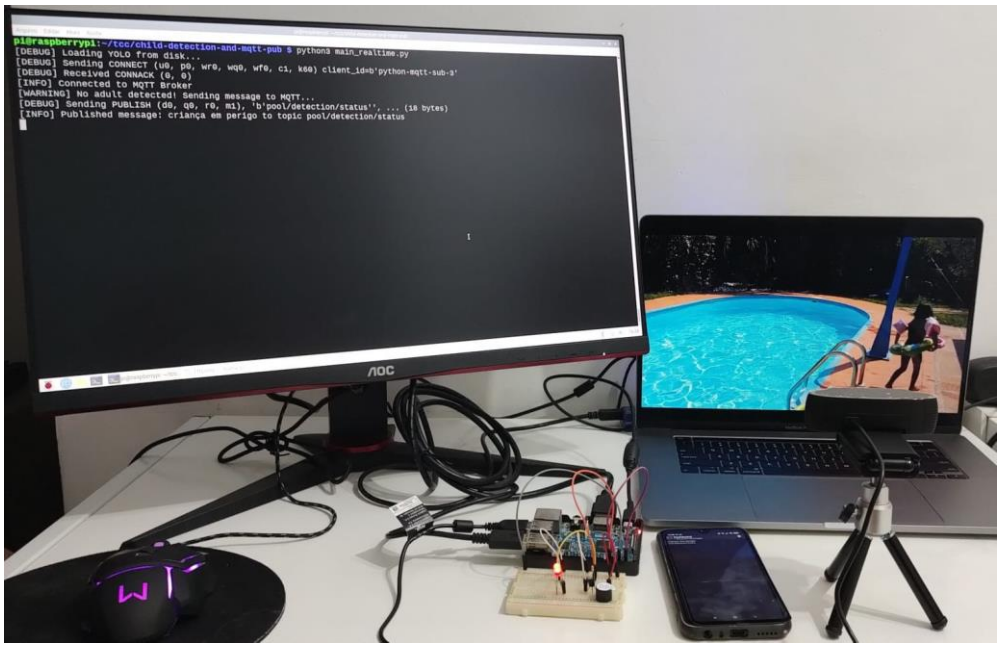
4.6 DETECÇÃO EM TEMPO REAL

Após garantir que todos os módulos, a detecção, o MQTT e o circuito de alarme, estão funcionando corretamente, os últimos resultados são de fato a detecção em tempo real através de uma simulação. Para isso, o algoritmo foi alterado para receber dados direto da câmera conectada ao *Raspberry Pi*, ao invés de receber um vídeo, para ser responsável de capturar os *frames*.

Para estes testes, a câmera foi apontada para um *notebook*, que estava exibindo um vídeo de uma criança sozinha em uma piscina, simulando uma situação real e a câmera capturou as informações e passou para o modelo. Para cada *frame* processado, o modelo realizou a predição em torno de um segundo com uma tolerância de 10%, e em seguida o algoritmo verificava se era necessário enviar informação para o MQTT e acionar o circuito, mantendo-se ativado por três segundos.

Os testes se sucederam bem, com as mensagens sendo enviadas constantemente, o circuito de alarme sendo acionado, e cada ciclo de predição sendo realizado sem haver muito tempo de espera. O *Raspberry Pi* se manteve com um desempenho bom, sem apresentar muito atraso na resposta do algoritmo e acionamento do circuito. A **Figura 35** apresenta como foi a construção deste ambiente de teste que pode ser visto em (VÍDEO2, 2022). Outro vídeo também foi utilizado, com a presença de um adulto, e, como esperado, sempre que era possível visualizá-lo, o algoritmo não realizava nenhuma ação, esse vídeo pode ser visto em (VÍDEO3, 2022).

Figura 35 – Ambiente de teste para simular a detecção em tempo real.



Fonte: Elaborado pela autora

CONCLUSÃO

O trabalho em questão teve o intuito de desenvolver um sistema de baixo custo capaz de detectar crianças em situações de perigo, ou seja, sozinhas perto ou dentro de piscinas utilizando técnicas de *Deep Learning* e Visão computacional implementadas em um sistema embarcado no *Raspberry Pi* 3.

Foi apresentado uma breve recapitulação de conceitos utilizados para o desenvolvimento da pesquisa e implementação do projeto, como: dados sobre afogamentos e suas fases; conceitos de aprendizagem de máquina e aprendizagem de máquina profunda, em específico as redes neurais, *transfer learning* e o porquê utilizá-lo, visão computacional e detecção de objetos com a rede neural YOLO, informações sobre a rede IoT com o Raspberry Pi, como funciona o protocolo MQTT e a arquitetura de *Publish* e *Subscriber* e por fim foi apresentado as métricas mais utilizadas para validar modelos, além de explicar sobre o *Google Colab* e o *Teachable Machine*.

O conjunto de dados utilizado nos treinamentos dos modelos de detecção e classificação abordados no trabalho foi criado a partir de imagens coletadas por redes sociais como o *Instagram* e o *Youtube* com crianças e/ou adultos imersos em determinado contexto. Para a tarefa de detecção cada imagem foi rotulada especificando a localização das classes desejadas e exportando-as em arquivos **.txt**.

O modelo de classificação foi treinado a partir da plataforma *Teachable Machine* e avaliado através das métricas de precisão, revocação e *F1-Score*, além da sua matriz de confusão. O modelo alcançou um desempenho de 61% no conjunto de teste na métrica *F1-Score*, este resultado mostra que o modelo possui um desempenho um pouco acima da média, porém abaixo do encontrado na tarefa de detecção.

Para a tarefa de detecção, utilizou-se o *framework Darknet* e a arquitetura YOLOv4-tiny para realizar o treinamento do modelo através de *transfer learning*. Para avaliar este modelo as mesmas métricas foram utilizadas com a inclusão do mAP, a principal métrica para avaliar um modelo de detecção de objetos, obtendo-se um desempenho de 89% de *F1-Score* e 89,5% de mAP no conjunto de teste.

Com essas métricas obtidas, pode-se concluir que o modelo de detecção apresentou um desempenho maior que o de classificação na tarefa de detecção nas situações propostas no conjunto de dados. Após a definição da melhor abordagem, prosseguiu-se para embarcar esse modelo no *Raspberry Pi*.

O modelo de detecção foi integrado à biblioteca *OpenCV*, para realizar a predição de imagens na linguagem Python, onde o modelo irá receber imagens, especificamente *frames*, capturados a partir de uma câmera conectada ao *Raspberry Pi*. Assim, o modelo irá realizar a detecção da imagem retornando à informação se a criança está sozinha ou não, este dado é apresentado de forma visual através de um circuito de alarme e mensagens enviadas pelo MQTT.

Estes testes foram realizados através de uma simulação, onde a câmera está posicionada para capturar *frames* de um vídeo executado em outro computador, a fim de simular uma detecção real de uma criança em situação de perigo. Este ambiente de teste se mostrou eficaz, visto que, durante a execução do vídeo, a câmera conseguiu capturar as imagens, que foram passadas para o modelo, onde o mesmo detectou corretamente e consequentemente acionou o circuito de alarme e enviou a mensagem de "criança em perigo" para o MQTT.

Avaliando os resultados obtidos e comparando-os com objetivos definidos no início do projeto, pode-se considerá-los satisfatório, dado que foi possível:

- Criar uma base de dados representativa de criança em situações reais;
- Desenvolver um modelo de detecção e classificação para compará-los;
- Avaliar os resultados dos modelos através de métricas para escolher a melhor abordagem;
- Integrá-lo em um sistema embarcado, com o circuito de alarme e envio de mensagens através do MQTT.

Por fim, como sugestão de continuidade deste trabalho, sugere-se realizar testes com novas versões da família YOLO, como a YOLOv5 para avaliar se o desempenho sobrepõe o da YOLOv4-tiny, desenvolver uma aplicação final para receber e apresentar as mensagens publicadas pelo MQTT, incrementar o circuito de alarme desenvolvido para ser uma solução comercial e testar a aplicação com outros sistemas embarcados mais robustos, como a *Raspberry Pi 4* e *Nvidia Jetson Nano*.

REFERÊNCIAS

ARAÚJO, Flávio H. et al. Redes neurais convolucionais com tensorflow: Teoria e prática. **SOCIEDADE BRASILEIRA DE COMPUTAÇÂO. III Escola Regional de Informática do Piauí**. Livro Anais-Artigos e Minicursos, v. 1, p. 382-406, 2017.

ANIRUDDHA. **Everything you should know about confusion matrix for machine learning. Analytics Vidhya,** 2020. Disponível em: https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/. Acesso em: 06/04/2022.

BAELDUNG. **Intersection Over Union for Object Detection. Baeldung CS**, 2018. Disponível em: https://www.baeldung.com/cs/object-detection-intersection-vs-union. Acesso em: 30/04/2022.

BASÍLIO, Shirley. **O Protocolo MQTT Publish/Subscriber (parte-01). Blog MasterWalker**, 2018. Disponível em: https://blogmasterwalkershop.com.br/outros/protocolo-mqtt-publish-subscriber-parte-01. Acesso em: 21/03/2022.

BITTENCOURT, Gabriela. **Brincadeiras na piscina: dicas, benefícios e cuidados no verão. Globo Esporte**, 2020. Disponível em: https://globoesporte.globo.com/eu-atleta/saude/noticia/brincadeiras-na-piscina-dicas-beneficios-e-cuidados-no-verao.ghtml. Acesso em: 10/05/2021.

BOCHKOVSKIY, Alexey; WANG, Chien-Yao; LIAO, Hong-Yuan Mark. **Yolov4: Optimal speed and accuracy of object detection**. **arXiv preprint arXiv:2004.10934**, 2020.

BRINK, Henrik; RICHARDS, Joseph W.; FETHEROLF, Mark. **Real-world machine learning**. [S.l.]: Manning, 2016.

BROWNLEE, Jason. **A Gentle Introduction to Transfer Learning for Deep Learning**. **Machine Learning Mastery**, 2019. Disponível em: https://machinelearningmastery.com/transfer-learning-for-deep-learning/. Acesso em: 27/06/2021.

BUDUMA, Nikhil; LOCASCIO, Nicholas. **Fundamentals of deep learning**: **Designing next-generation machine intelligence algorithms**. [S.l.]: O'Reilly Media, Inc., 2017.

CASTRO, Cristiano Leite de; BRAGA Antônio Pádua. **Aprendizado supervisionado com conjuntos de dados desbalanceados.** Controle & Automação Sociedade Brasileira de Automática, v. 22, p. 441-466, 2011.

CHEN, Alex et al. Teachable Machine: Approachable Web-Based Tool for Exploring Machine Learning Classification. **Teachable Machine: Approachable Web-Based Tool for Exploring Machine Learning Classification**, 2020.

CIRCUITO.IO. **Design your circuit with circuito.io. Circuito.io**, 2016. Disponível em: https://www.circuito.io/. Acesso: 19/04/2022.

CIRCUITLAB. **Circuit simulation and schematics. CircuitLab,** 2016. Disponível em: https://www.circuitlab.com/. Acesso em 19/04/2022.

DA SILVA, Martony Demes. **Aplicação da Ferramenta Google Colaboratory para o Ensino da Linguagem Python**. In: **Anais da IV Escola Regional de Engenharia de Software**. SBC, 2020. p. 67-76.

DE ANDRADE, Alexandre Acácio. **Automação de baixo custo baseada no Raspberry Pi**. 2016.

EQMX. **An Open-Source, clould-native, distributed MQTT broker for IoT. EMQX.io**, 2013. Disponível em: https://www.emqx.io/. Acesso em: 11/04/2022.

FOUNDATION, RASPBERRY PI. **Rasberry Pi**, 2011. Disponível em: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. Acesso em: 26/06/2021.

FRANZ, Augusto César Medeiros et al. **Desenvolvimento de uma ferramenta visual de classificação de imagens para o ensino de machine learning no ensino médio**. 2021.

GARCÍA, Gloria Bueno et al. **Learning Image Processing with OpenCV**. [S.l.]: Packt Publishing Ltd, 2015.

GAD, Ahmed Fawzy. Evaluating Deep Learning Models: The Confusion Matrix, Accuracy, Precision and Recall. PaperspaceBlog, 2020. Disponível em: https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy/#:~:text=The%20recall%20is%20calculated%20as,ability%20to%20detect%20Positive%20samples. Acesso: 07/04/2022.

GIRSHICK, Ross et al. Rich feature hierarchies for accurate object detection and semantic segmentation. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. 2014. p. 580-587.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep Learning**. [S.l.]: MIT press, 2016. http://www.deeplearningbook.org.

GOOGLE. **Olá, este é o Colaboratory**, 2019. Disponível em: https://colab.research.google.com/notebooks/intro.ipynb#scrollTo=lSrWNr3MuFUS. Acesso em: 22/09/2021.

GOOGLE. **Teachable Machine. Techable Machine**, 2017. Disponível em: https://teachablemachine.withgoogle.com/. Acesso em: 15/03/2022.

GPIO. **A Python module to control the GPIO on a Raspberry Pi. SourceForge**, 2013. Disponível em: https://sourceforge.net/projects/raspberry-gpio-python/. Acesso em: 07/04/2022.

GRGIĆ, Krešimir; ŠPEH, Ivan; HEĐI, Ivan. **A web-based IoT solution for monitoring data using MQTT protocol**. **2016 international conference on smart systems and technologies (SST)**. IEEE, 2016. p. 249-253.

HIVEMQ. **MQTT Topics & Best Practices – MQTT Essentials: Part 5. HIVEMQ**, 2019. Disponível em: https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/. Acesso em: 24/03/2022.

HUI, Jonathan. **mAP (mean Average Precision) for Object Detection. Medium**, 2018. Disponível em: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173. Acesso em: 05/04/2022.

IBM. **MQTT**. **IBM Support,** 2022. Disponível em: https://www.ibm.com/support/pages/mqtt. Acesso em: 01/04/2022.

IRONHACK. **O que é machine learning? IronHack**, 2019 Disponível em: https://www.ironhack.com/br/data-analytics/o-que-e-machine-learning. Acesso em: 15/06/2021.

JAIN, Ramesh; KASTURI, Rangachar; SCHUNCK, Brian G. **Machine Vision**. New York: McGraw-Hill, 1995.

JAFFEY, Toby. **MQTT and CoAP, IoT protocols**. **Eclipe Foundation**, 2014. Disponível em: https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php. Acesso em: 03/04/2022.

KHAN, Salman et al. A Guide to Convolutional Neural Networks for Computer Vision. **Synthesis Lectures on Computer Vision**, v.8, n. 1, p. 1-207, 2018.

KORSTANJE, Joos. **The F1 score**. **Towards Data Science**, 2021. Disponível em: https://towardsdatascience.com/the-f1-score-bec2bbc38aa6. Acesso em: 30/03/2022.

KOVACS, Leandro. **O que é VLC? (VideoLAN). Tecnoblog**, 2021. Disponível em: https://tecnoblog.net/responde/o-que-e-vlc-videolan/. Acesso em: 12/04/2022.

LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. Deep learning. **nature**, v. 521, n. 7553, p. 436-444, 2015.

LIN, Tzuta. **LabelImg, Github**, 2019. Disponível em: https://github.com/tzutalin/labelImg. Acesso em: 15/09/2021.

LUCENA, Oeslle AS; VELOSO, Luciana R.; LOPES, Waslon TA. **Implementação de um Sistema de Reconhecimento de Objetos em Imagens**. **Revista de Tecnologia da Informação e Comunicação**, v. 6, n. 2, p. 34-42, 2016.

MARSLAND, Stephen. **Machine learning: an algorithmic perspective**. 2ª. ed. [S.l.]: CRC press, 2015.

MATT. **Simple Guide to the Raspberry Pi GPIO Header. Raspberry-spy**, 2012. Disponível em: https://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/. Acesso em: 07/02/2022.

METRÓPOLES. **Desde 2005, Brasil registrou, em média, 12 mortes por afogamento a cada dia**. **Metrópoles**, 2020. Disponível em: https://www.metropoles.com/brasil/desde-2005-brasil-registrou-em-media-12-mortes-por-afogamento-a-cada-dia. Acesso em: 08/06/2021.

NOGUEIRA, Angelo Baruffi et al. **Análise de viabilidade no uso de Deep Learning para contagem de pessoas com câmeras de segurança**. 2018.

OLIVEIRA, Fabiano Pereira de et al. **TMIC-Uma extensão do App Inventor para a implantação de modelos de ML voltados a classificação de imagens treinados no Teachable Machine**. 2022.

OPENCV. **Deep neural network module**. **OpenCV**, 2019. Disponível em: https: https://opencv.org/. Acesso em: 03/03/2022.

PAHO. **Eclipse Paho. Eclipse Foundation**, 2014. Disponível em: https://www.eclipse.org/paho/. Acesso em: 12/04/2022.

PICAMERA. **Docs Picamera. Picamera**, 2016. Disponível em: https://picamera.readthedocs.io/en/release-1.13/. Acesso em: 05/04/2022.

PRAKASH, Jatin. **Non Maximun Suppression: Theory and Implementation in Pytorch. LearnOpenCV,** 2021. Disponível em: https://learnopencv.com/tag/non-maximum-suppression/. Acesso em: 31/03/2022.

PONTI, Moacir Antonelli; COSTA, Gabriel B. Como funciona o Deep Learning. **arXiv preprint arXiv:1806.07908**, 2018.

REDMON, Joseph; FARHADI, Ali. YOLOv3: An Incremental Improvement. **arXiv preprint arXiv:1804.02767**, 2018.

REDMON, Joseph. **Darknet: Open Source Neural Networks in C**, 2016 Disponível em: https://pjreddie.com/darknet/. Acesso em: 12/10/2021.

REDMON, Joseph et al. You only look once: Unified, real-time object detection. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**, 2016. p. 779-788.

ROY, Ajil; SRINIVASAN, K. A novel drowning detection method for safety of swimmers. In: IEEE. **2018 20th National Power Systems Conference (NPSC)**, Tiruchirappalli, 14 Dezembro 2018. 1-6.

ROSEBROCK, Adrian. **Deep Learning: How OpenCV's blobFromImage works. Pyimagesearch,** 2017. Disponível em: https://pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/. Acesso em: 05/04/2022.

RUSSEL, Stuart; NORVIG, Peter**. Inteligência Artificial**. 3. ed. Rio de Janeiro: Campus, 2013. 1016p.

SALINET, Maria Fernanda. **Natação para crianças: quais os benefícios**. **ndmais**, 2021. Disponível em: https://ndmais.com.br/seguranca/bombeiros/veja-como-evitar-afogamento-de-criancas-em-piscinas-no-verao/amp/. Acesso em: 06/05/2021.

SARKAR, Dipanjan. **A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning**. **Towards Data Science**, 2018. Disponível em:

https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a. Acesso em: 27/06/2021.

SEGURA, Criança. **Entenda os Acidentes**. **Criança Segura**, 2018. Disponível em: https://criancasegura.org.br/entenda-os-acidentes/. Acesso em: 22/06/2021.

SETHI, Pallavi; SARANGI, Smruti R. **Internet of Things: Architectures, Protocols, and Applications**. **Journal of Electrical and Computer Engineering**, v. 2017, 2017.

SHAH, Tarang. Measuring Object Detection Models-MAP-What is mean average precision?. Towards Data Science, 2018. Disponível em: https://towardsdatascience.com/what-is-map-understanding-the-statistic-of-choice-for-comparing-object-detection-models-1ea4f67a9dbd. Acesso em: 03/04/2022.

SILVA, Adilane Ribeiro da. **Uma visão geral sobre Machine Learning – Classificação. Operdata,** 2020. Disponível em: https://operdata.com.br/blog/uma-visao-geral-sobre-machine-learning/. Acesso em: 06/04/2022.

SILVA, Marcos Aurélio Medeiros; CHAVES, A. P. A.; AQUINO, F. J. A. **Projeto e desenvolvimento de uma ferramenta educativa para ensino de processamento de imagens baseado na biblioteca OpenCV**. In: XL Congresso Brasileiro de Educação em Engenharia, Belém. 2012.

SILVA, Wilson Alves da. **Uma arquitetura para orquestração da distribuição de água no semiárido brasileiro baseada em internet das coisas e computação em nuvem**. UFPE. Pernambuco. 2017.

SOBRASA. **AFOGAMENTOS O que está acontecendo? Sobrasa**, 2020. Disponível em: https://www.sobrasa.org/new_sobrasa/arquivos/baixar/AFOGAMENTOS_Boletim_Brasil_2020.pdf. Acesso em: 04/05/2021.

SOUZA, Fábio. O Hardware da Raspberry Pi 3. **Embarcados**, 2016. Disponível em: https://www.embarcados.com.br/hardware-da-raspberry-pi-3/. Acesso em: 25/06/2021.

STEVE. **Understanding MQTT Topics. Steve's Internet Guide**, 2022. Disponível em: http://www.steves-internet-guide.com/understanding-mqtt-topics/#:~:text=MQTT%20topics%20are%20a%20form,(%20%2F%20)as%20a%20delimiter. Acesso em: 23/03/2022.

SZELISKI, Richard. **Computer vision: algorithms and applications**. [S.l.]: Springer Science & Business Media, 2010.

SZPILMAN, David. **AFOGAMENTO**. **Szpilman**, 2015. Disponível em: http://www.szpilman.com/new_szpilman/szpilman/ARTIGOS/Cap\%203\%20-\%20Afogamento\%20\%2005-11-15_szpilman.pdf. Acesso em: 20/06/2021.

TORRES, Andrei BB; ROCHA, Atslands R.; DE SOUZA, José Neuman. **Análise de desempenho de brokers mqtt em sistema de baixo custo**. In: **Anais do XV Workshop em Desempenho de Sistemas Computacionais e de Comunicação**. SBC, 2016. p. 2804-2815.

VASCONCELOS, Yuri. **O que acontece no corpo durante um afogamento? Abril**, 2019. Disponível em: https://super.abril.com.br/mundo-estranho/o-que-acontece-no-corpo-durante-um-afogamento/. Acesso em: 08/06/2021.

VÍDEO1, Emily. **Yolo Detection Adult Child 2. Youtube**, 2022. Disponível em: https://youtu.be/J6NcQVkn6fo.

VÍDEO2, Emily. **Test Environment 1. Youtube**, 2022. Disponível em: https://youtu.be/o79XHTjM9h4.

VÍDEO3. Emily. **Test Environment 2. Youtube**, 2022. Disponível em: https://youtu.be/DN7NUMy1Few.

WANG, Chien-Yao; BOCHKOVSKIY, Alexey; LIAO, Hong-Yuan Mark. **Scaled-yolov4: Scaling cross stage partial network**. **Proceedings of the IEEE/cvf conference on computer vision and pattern recognition**, p. 13029-13038, 2021.

WEIAND, Augusto. **Análise de sentimentos do Twitter com Naïve Bayes e NLTK. Trajetória Multicursos**, v. 7, n. 2, p. 3-18, 2018.

WOOD, Thomas. **Wha tis the Softmax Function?. DeepAi**, 2018. Disponível em: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer. Acesso em: 21/03/2022.

ZOLETT, Daniel; RAMIREZ, Alejandro Rafael Garcia. **Desenvolvimento de uma Interface de Monitoração Remota para o Sistema Robótico ROBIX, Integrando o Protocolo MQTT e oROS**. **Anais do Computer on the Beach**, v. 11, n. 1, p. 405-412, 2020.