INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Sistemas de segurança Plug & Play para equipamentos de Grab and Go

SÉRGIO FILIPE CARREIRINHA
Outubro de 2023

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Sistemas de segurança Plug & Play para equipamentos de Grab and Go

Sérgio Filipe Carreirinha

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas Gráficos e Multimédia

Orientador: Paula Maria de Sá Oliveira Escudeiro

Júri:

Presidente:

Fernando Jorge Ferreira Duarte, Professor Adjunto, ISEP

Vogais:

António Manuel De Sousa Barros, Professor Adjunto, ISEP

Porto, 28 de dezembro de 2022

Declaração de Integridade

Declaro ter conduzido este trabalho académico com integridade.

Não plagiei ou apliquei qualquer forma de uso indevido de informações ou falsificação de resultados ao longo do processo que levou à sua elaboração.

Portanto, o trabalho apresentado neste documento é original e de minha autoria, não tendo sido utilizado anteriormente para nenhum outro fim.

Declaro ainda que tenho pleno conhecimento do Código de Conduta Ética do P. PORTO.

ISEP, Porto, 11 de outubro de 2023

Resumo

A indústria das máquinas de venda automática é um importante contribuinte para a economia global, sendo um setor com um crescimento notável devido à crescente procura por opções convenientes de autoatendimento e de avanços tecnológicos, como sistemas de pagamento sem dinheiro e máquinas de venda automática inteligentes. Apesar de serem projetadas com características antirroubo, o roubo destas máquinas é uma ocorrência frequente, deixando os proprietários com elevadas despesas.

Considerando a relevância da indústria de máquinas de venda automática e os desafios de segurança enfrentados, foi tomada a decisão de desenvolver um sistema capaz de identificar características físicas e emocionais de potenciais ladrões, com o objetivo de criar um detetor de atividades suspeitas. Este sistema visa fornecer uma camada adicional de proteção e mitigação de roubos nas máquinas de venda automática da *Reckon.ai*, mas pode ser utilizado em qualquer propriedade, reduzindo as despesas e prejuízos enfrentados pelos seus proprietários.

A solução proposta divide-se em três blocos fundamentais, cada um dedicado a uma tarefa específica. O primeiro consiste numa *neural network* dedicada à identificação de expressões faciais, o segundo foca-se no reconhecimento de objetos, e o terceiro aborda a deteção de ações agressivas.

A aplicação consiste em captar imagens em tempo real a partir da webcam, verificar se existe movimento e rostos no *frame*, extrair o rosto do utilizador, classificar qual a expressão facial que este manifesta, identificar objetos que representem perigo e classificar a ação. Por fim, o sistema analisa as respostas dos três classificadores, identificando se o indivíduo presente em cena é ou não suspeito.

Foram empregues diversos métodos e conduzidas várias experiências com o intuito de otimizar o desempenho dos modelos. Estas abordagens incluíram o ajuste dos hiperparâmetros, a análise dos resultados de várias vertentes da arquitetura e a exploração de diferentes arquiteturas. Também é feita uma comparação entre as experiências desenvolvidas com outros trabalhos que tenham o mesmo objetivo.

O sistema final foi testado com um *dataset* de 200 vídeos e teve como resultado 127 avaliações corretas num total de 150 para suspeitos e 39 avaliações corretas num total de 50 para não suspeitos, representando isto uma *accuracy* de 0,83.

Palavras-chave: Sistema de Deteção de Suspeitos, Classificação de Expressões Faciais, Classificação de Objetos, Classificação de ações, Rede Neuronal Convolucional, Aprendizagem Profunda

Abstract

The vending machine industry is a significant contributor to the global economy, representing a sector with remarkable growth due to the increasing demand for convenient self-service options and technological advancements such as cashless payment systems and smart vending machines. Despite being designed with anti-theft features, the theft of these machines remains a frequent occurrence, leaving owners with high expenses.

Considering the relevance of the vending machine industry and the security challenges faced, the decision was made to develop a system capable of identifying physical and emotional characteristics of potential thieves with the goal of creating a suspicious activity detector. This system aims to provide an additional layer of protection and mitigation of thefts in Reckon.ai vending machines but can be utilized in any property, reducing expenses and losses for its owners.

The proposed solution is divided into three fundamental blocks, each dedicated to a specific task. The first block consists of a neural network dedicated to facial expression identification, the second focuses on object recognition, and the third addresses the detection of aggressive actions.

The application involves capturing real-time images from the webcam, checking for movement and faces in the frame, extracting the user's face, classifying their displayed facial expression, identifying potentially dangerous objects, and classifying the action. Finally, the system analyzes the responses from the three classifiers to determine whether the individual present in the scene is suspicious or not.

Various methods were employed, and numerous experiments were conducted to optimize the performance of the models. These approaches included fine-tuning hyperparameters, analyzing results from different architectural perspectives, and exploring various architectures. Additionally, a comparison was made between the experiments carried out and other works with similar objectives.

The final system was tested with a dataset of 200 videos, resulting in 127 correct assessments out of 150 for suspects and 39 correct assessments out of 50 for non-suspects, achieving an accuracy of 0.83.

Keywords: Suspect Detection System, Facial Expression Classification, Object Classification, Action Classification, Convolutional Neural Network, Deep Learning.

Agradecimentos

Em primeiro lugar, gostaria de expressar o meu agradecimento à minha entidade paternal, *Reckon.ai*, por me proporcionar a oportunidade de trabalhar neste projeto e por me disponibilizar todo o material necessário para que a sua conclusão fosse um sucesso.

Um segundo agradecimento vai para a Eng. Rita Carvalho e ao Eng. Sérgio Pinto, por toda a disponibilidade e senso crítico, a fim de tomar as melhores e mais eficientes decisões contribuindo para o melhor desenvolvimento deste projeto.

Em seguida, um especial agradecimento à minha família e namorada, por me acompanharem durante toda esta jornada, pelo apoio constante, pela motivação, por sempre acreditarem nas minhas habilidades e por nunca me deixarem desistir dos meus objetivos.

Quero também agradecer a todos os meus amigos que me acompanharam neste ciclo académico, em especial ao César Ferreira e ao José Cunha, por partilharem comigo as mesmas preocupações e conquistas.

Não posso esquecer de agradecer ao Instituto Superior de Engenharia do Porto, bem como a todos os meus professores, porque graças à sua dedicação e empenho, tive a oportunidade de aprender e crescer para uma vida repleta de oportunidades na área de engenharia.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos

Lista de Acrónimos

CNN	*Convolucional Neural Network*
NN	*Neural Network*
ANN	*Artificial Neural Networks*
MTCNN	Multi-Task Cascaded Convolutional Neural Networks
R-CNN	Region Based Convolutional Neural Networks
RNN	Recurrent neural network
HDMI	High-Definition Multimedia Interface
USB	Universal Serial Bus
LSTM	Long short-term memory
SMTP	Simple Mail Transfer Protocol
YOLO	You Only Look Once

1 Introdução

A indústria das máquinas de venda automática contribui significativamente para a economia mundial e gera biliões de dólares em receitas em curtos prazos. Esta indústria tem registado elevado crescimento nos últimos anos devido à crescente procura por opções convenientes de auto atendimento e pelos avanços tecnológicos que se tem verificado, como sistemas de pagamento sem dinheiro e máquinas de venda automática inteligentes. As receitas geradas por estas máquinas podem oscilar dependendo de fatores como localização, tipo de produtos vendidos e nível de concorrência e estão também sujeitas a alterações ao longo do tempo devido às condições econômicas e de mercado. No entanto, estas máquinas têm sido uma fonte significativa de renda para empresas e empreendedores e, espera-se, que o mercado global das máquinas de venda automática continue a crescer no futuro (Parlevel Systems, 2014).

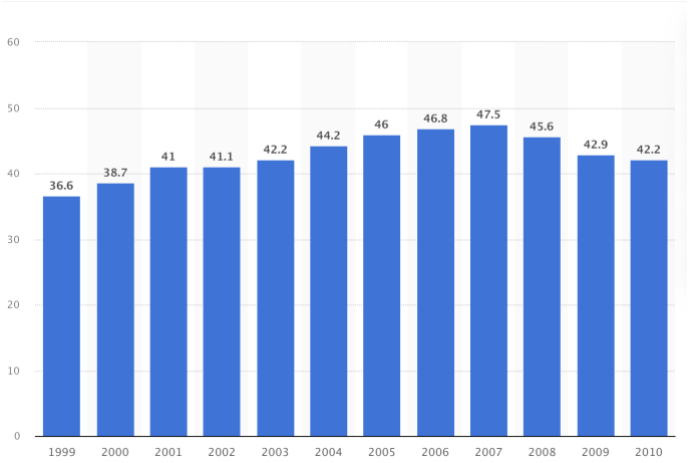


Figura 1 - Volume total de produtos vendidos em máquinas de venda automática nos EUA de 1999 a 2010 (Statista, 2012)

Muitas pessoas que se querem aventurar no negócio das máquinas de venda automática começam por se questionar se estas são facilmente danificadas ou roubadas. Do ponto de vista de um fabricante profissional, todos consideram que não, dada a própria natureza das máquinas de venda automática serem projetadas com características contra o roubo, serem maioritariamente representadas pelos seus materiais fortificados, como o vidro de camada dupla e a chapa de metal galvanizada forte. Porém, o roubo destas máquinas é uma prática frequentemente visualizada em vários jornais. Para fornecedores que lidam com baixas margens de lucro, “experimentar” o roubo destas máquinas pode significar a perda de uma parte significativa da receita obtida num dia ou semana, pois a máquina fica inutilizável no período da sua reconstrução e o valor da sua receita é afetado.

O teste de integridade das máquinas vai da consciência das pessoas que a usam, mas o facto destas máquinas conterem algum dinheiro e mercadoria expostos como se fosse um "cofre natural" e não tipicamente com vigilância 24/7, dá uma grande tentação aos ladrões. Empiricamente existem dois tipos de roubo ligados a esta indústria. Os ladrões externos, que normalmente usam meios violentos para demolir, danificar as máquinas e roubar propriedades e o infiltrado, vulgarmente conhecido como "toupeira", que estão associados por norma aos funcionários das próprias máquinas. Estes estão familiarizados com as lacunas, como o bloqueio do interruptor não estar gravado, a chave poder ser copiada, etc., para que possam facilmente lucrar com cada máquina.

1.1 Contextualização do problema

A *Reckon.ai* é uma empresa de tecnologia com sede em Portugal que se especializa na produção de *software* de inteligência artificial e visão computacional. Tem como principal objetivo oferecer soluções personalizadas para empresas em diferentes setores, incluindo retalho, saúde, entre outros. Uma das soluções da *Reckon* é a produção e/ou renovação de máquinas comuns para torná-las em máquinas de venda totalmente autónomas com um conjunto de novas funcionalidades – plataforma de *backoffice*, controlo de utilizador e análise do stock em tempo real. Estas *microstores* inteligentes tornam o processo de compra de produtos mais simples, autónoma e eficiente e para isso incluem a análise de dados, a visão computacional para processamento de imagens e a inteligência artificial para tomada de decisão.

Figura 2 - Microstore com a tecnologia BuyBye da Reckon.ai

As máquinas de venda baseadas em inteligência artificial e visão computacional usam estas ferramentas para melhorar a eficiência e a conveniência da experiência de compra. Estas áreas tecnologias combinam a analise de dados das balanças colocadas nas prateleiras que sustentam os produtos e das camaras incorporadas no corpo da máquina para que seja possível identificar o produto selecionado pelo cliente, atribuir automaticamente o preço, gerar uma fatura e gerir o stock daquela *microstore*. A ideia é que as combinações destas ferramentas tornem a máquina de vendas mais eficiente e conveniente para os clientes, proporcionando uma experiência de compra mais simples, dividida em 3 simples passos:

1. Tocar com o cartão ou telemóvel no sensor da *microstore*.
2. Interagir com o armário e tirar os produtos que deseja.
3. Fechar a porta.

Este fluxo é controlado por um **Rasberry PI** que proporciona à *microstore* todos os recursos para o modulo funcional da máquina.

O *RaspBerry Pi (RPi)* é um computador pequeno integrado numa única placa. Trata-se de uma peça única que possui conectores para diversos dispositivos sendo possível adicionar monitores, sensores e outros periféricos. É um computador de baixo custo, de tamanho pequeno e alto desempenho, que pode ser usado para uma ampla gama de projetos, incluindo a construção de *embedded systems*. No mundo do retalho, este equipamento é vendido como uma placa de desenvolvimento para projetos DIY (*do it yourself*) e é amplamente utilizado para criar soluções personalizadas para diversas finalidades devido à sua versatilidade (Raspberry Pi Trading Ltd., 2021).

Existem vários modelos de *Raspberry Pi* disponíveis, com diferentes especificações de hardware, mas todos executam o sistema operacional Linux e são capazes de executar uma ampla gama de aplicações e projetos, desde simples tarefas de computação até projetos de robótica e Internet das coisas (Iot). Um exemplo, é o modelo do *Raspberry* 4 Model B mostrado na figura abaixo.

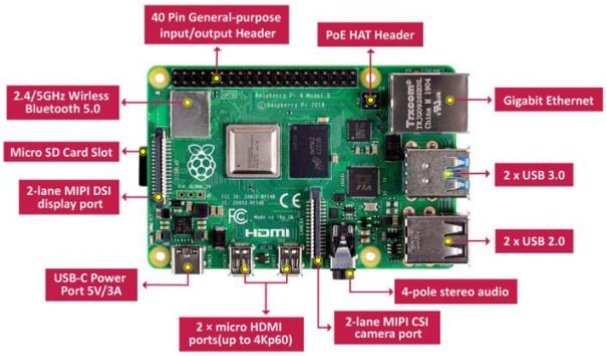


Figura 3 - Raspberry Pi 4 Model B Hardware

Grande parte dos componentes de *hardware* evidenciados acima têm um papel fundamental no funcionamento das *microstores*, sendo os mais relevantes os seguintes:

- HDMI (Interface Multimédia de Alta-Definição): É usado para transmitir dados de vídeo e áudio com o Monitor do Computador. Normalmente é utilizado antes de ser introduzido na máquina, facilitando o processo de configuração do RPI e a instalação do software necessário.
- USB (porta de série universal): É uma interface *plug & play* que permite a comunicação entre o computador e outros dispositivos. É através deste componente que o *software* instalado no RPI consegue controlar e comunicar com outros componentes do sistema da *microstore* (camaras e balanças).

Algumas das aplicações do *RaspBerry* utilizadas nas *microstores* incluem o controlo das balanças de todas as prateleiras e a monitorização e gravação das camaras que através do *software* instalado permitem identificar e quantificar os produtos retirados. O equipamento encontra-se normalmente nos módulos laterais criados pela *Reckon* facilitando o seu acesso, gestão e controlo das ligações com os restantes componentes. Foi adotada a escolha deste dispositivo para o controlo das *microstores* numa fase inicial devido à vasta documentação existente na internet, contudo, o aumento dos preços do mercado exigiu o estudo e configuração de outras soluções mais económicas, porém com menos documentação, como é o caso do Orange PI.

1.2 Motivação

A *Reckon.ai* foi fundada em 2017, com o principal objetivo de transformar o retalho através da combinação de inteligência artificial e visão computacional, tendo como principal princípio a experiência do utilizador. Levando isso em consideração, criaram MicroLojas que podem operar sozinhas 24/7, sem intervenção humana, proporcionando uma experiência de compra sem qualquer tipo de atrito tanto para os consumidores finais quanto para os operadores de comércio.

Atualmente os equipamentos *Grab and Go* necessitam de estar junto de sistemas de segurança externos para evitar furtos/danos no equipamento. A problemática da falta de segurança das máquinas de venda em ambientes propícios ao furto tem sido amplamente discutida nos últimos anos, e tem sido identificada como um fator crítico para o sucesso e satisfação dos clientes. Com o objetivo de compreender melhor esta questão, foi proposto a realização, numa primeira fase, de uma revisão da literatura no sentido de identificar e estudar várias técnicas de segurança e, posteriormente, ser apresentado um sistema *plug & play* que aplique essas mesmas técnicas resolvendo então o problema em questão.

A escolha deste tema é motivada pela crescente importância da proteção de bens e propriedades de empresas e até mesmo pessoais. Além disso, este estudo pode fornecer *insights* valiosos para as empresas, ajudando-as a desenvolver estratégias mais eficazes para

aumentar a segurança dos seus pertences. O crime é uma das maiores preocupações da sociedade moderna e o furto é um dos crimes mais comuns. A criação deste *software* anti-furto é uma forma de ajudar a combater esse problema e proteger as pessoas e os seus bens. Além disso, o desafio de desenvolver esta solução tecnológica com o objetivo de contribuir para a redução da criminalidade e para a proteção da sociedade é uma motivação constante.

1.3 Objetivo

O principal objetivo desta dissertação é, numa primeira fase, fazer uma revisão da literatura recorrendo a artigos científicos, teses e trabalhos desta área, no sentido de identificar e estudar as várias técnicas de segurança e, de seguida, proceder à realização de uma avaliação experimental de uma ou várias técnicas existentes. Como conclusão do projeto deve ser apresentado um sistema *plug & play* intuitivo para proteger o patrimônio dos utilizadores, fornecendo recursos avançados de deteção e prevenção de roubos e furtos, bem como funcionalidades alarmísticas de possíveis perigos, para garantir a segurança do património do proprietário e a sua tranquilidade.

1.4 Contribuições

Esta dissertação apresenta as seguintes contribuições:

- Desenvolvimento de um algoritmo inovador de deteção de roubos baseado em inteligência artificial e visão computacional, que permite uma maior precisão na identificação de atividades suspeitas e uma resposta mais rápida em caso de tentativa de roubo.
- Análise crítica dos *softwares* anti-roubo existentes no mercado e revisão da literatura, permitindo identificar as lacunas no mercado e desenvolver soluções mais eficazes.
- Realização de testes aos modelos de *deep learning* utilizados, garantindo a eficácia e a confiabilidade do sistema final.
- Desenvolvimento de novos conjuntos de dados.

1.5 Estrutura

A estrutura deste documento está dividida nas seguintes partes: Introdução, Estado da Arte, Metodologia, Desenho da Solução, Implementação, Experiências e Resultados e Conclusão. A Introdução é composta pelo problema, metodologia, objetivos do trabalho, juntamente com os principais requisitos da solução, contribuições, contextualização do problema e a análise de valor. O Estado da Arte é composto por conceitos técnicos de inteligência artificial, redes, extração de recursos de imagem, metodologias de avaliação de modelos, tecnologia e abordagens científicas relacionadas ao problema. A Metodologia apresenta os métodos

usados para a análise e reavaliação dos *datasets* utilizados, e para a avaliação de impacto. O Desenho da Solução apresenta alternativas da arquitetura da solução e a escolha da linguagem de programação e da *framework* de *deep learning*. O capítulo 5 apresenta o processo de desenvolvimento dos conjuntos de dados utilizado pelos modelos, os modelos utilizados na solução e sistema como um todo. O capítulo das Experiências e Resultados apresenta os resultados de cada um dos módulos da metodologia. E, finalmente, no capítulo 6, alguns insights finais, escrevendo recomendações sobre o trabalho feito e trabalhos futuros que podem ser explorados.

1.6 Análise de Valor

1.6.1 SWOT

A Análise SWOT é uma ferramenta de diagnóstico/análise estratégica que tem como objetivo identificar os fatores internos e externos representados por Pontos Fortes *(Strenghts)*, Pontos Fracos (*Weaknesses*), Oportunidades (*Opportunities)* e Ameaças *(Threats)* para que, posteriormente, a organização possa tirar partido de determinadas oportunidades, possa minimizar as ameaças, potenciar os pontos fortes e externalizar os pontos fracos.

Tabela 1 - Análise SWOT

Pontos Fortes	**Pontes Fracos**
Inovação Próprio conceito do novo *Software* (anti-roubo) Acrescenta valor às *microstores* Tecnologias própria e difícil de ser copiada	Custos elevados com o *Software* Custos de instalação Custos de manutenção *Software* novo ainda desconhecido
Novidade de *Software* Reduzida Concorrência Aumento de segurança social e pessoal Contribuição para a diminuição de roubos Elevada procura por soluções anti-roubo	Concorrência de outros sistemas anti-roubo mais comuns (alarmes)
Oportunidades	**Ameaças**

Com a implementação deste novo *software* anti-roubo a Reckon.ai poderá aumentar a sua cota de mercado dada a elevada demanda neste tipo de soluções e a reduzida concorrência. Para tal, torna-se também necessário que as soluções sejam as mais corretas e precisas possíveis de forma a fazer face às ameaças do mercado (outros sistemas existentes de anti-roubo). Embora o desconhecimento deste novo *software* possa atrasar o crescimento

da cota de mercado, com a elaboração de um bom plano de marketing e apresentação de bons resultados o projeto irá conseguir-se destacar.

1.6.2 *Business Model Canvas*

O *Business Model Canvas* é uma ferramenta de planeamento estratégico que permite a visualização do negócio de uma forma fácil em apenas uma página, revelando toda a interligação estratégica existente entre as diferentes áreas, ou seja, permite esboçar ou desenvolver modelos de negócio novos ou pré-existentes.

O principal objetivo é estruturar um modelo inovador do plano de negócios de forma prática e dinâmica na análise interna das organizações e demonstrar a forma como a Reckon.ai gera valor e o entrega aos seus clientes, permitindo reconhecer a viabilidade financeira do projeto. Trata-se, portanto, de um mapa visual elaborado para perceber se cada pilar do negócio está a ter a devida atenção.

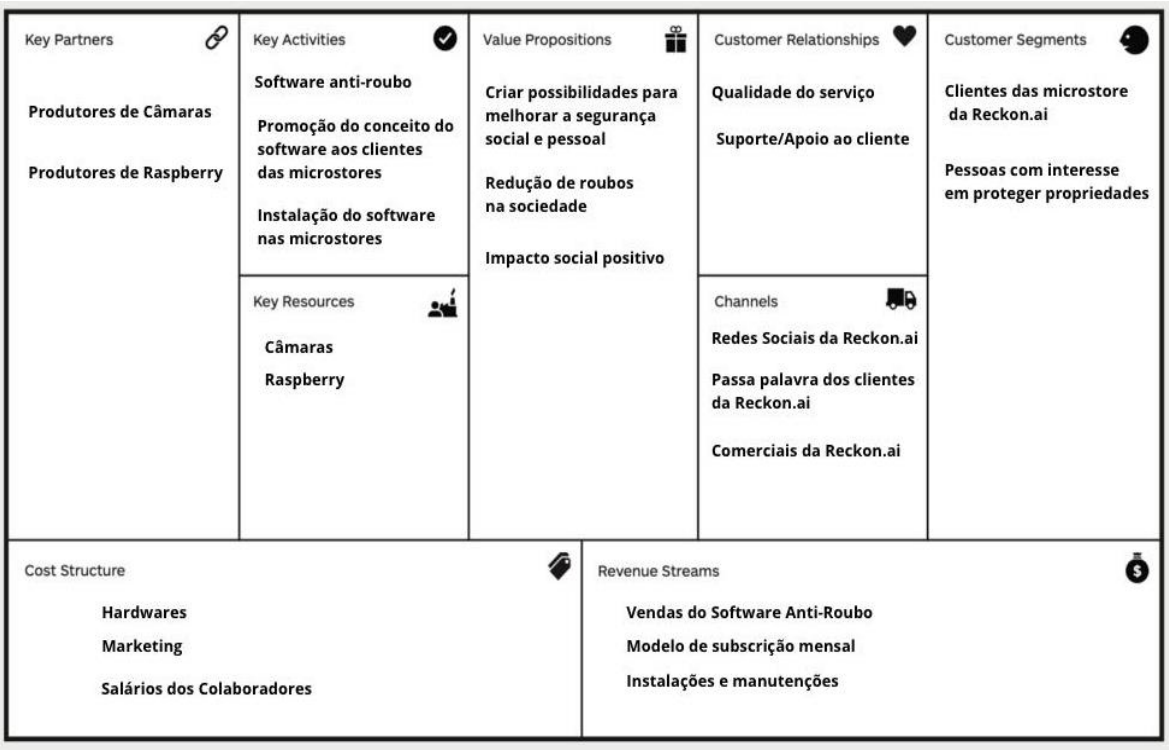


Figura 4 - Canva Business Model

1.7 Sumário

A *Reckon.ai* é uma empresa especializada em *software* que tem como objetivo principal revolucionar o retalho, mais concretamente, simplificar o processo de compra nas máquinas de venda automática. Combinam a inteligência artificial e a visão computacional para entregar ao cliente *microstores* inteligentes e autónomas, com a capacidade de operarem sozinhas e proporcionar uma experiência mais simples, comoda e inovadora ao utilizador. Estas máquinas são controladas por microcomputadores chamados *RaspBerry* que têm a capacidade de gerir o processo de aquisição de imagens (auxiliada por camaras) e de valores dos sensores de peso, combinando e identificando os produtos que o utilizador retirou da *microstore*. O *RaspBerry* PI será também o dispositivo onde irá ser instalado e testado a solução proposta para este projeto.

2 Revisão de literatura

De forma geral, um sistema anti-roubo baseado em *machine learning (ML)* consiste nas seguintes fases: deteção de movimento num determinado lugar de acordo com alguns requisitos, reconhecimento através de um modelo de ML de expressões faciais e identificação de pessoas que possam estar a usar mascaras e, por fim, detetar movimentos suspeitos no espaço envolvente por qualquer tipo de ameaça que o suspeito poderá ter. No presente capítulo, é dado uma contextualização do tipo de algoritmo utilizado nesta dissertação, a classificação.

Na seção 2.1 são explicados os fundamentos do *deep learning*: *neural networks* e o processo de aprendizagem na secção 2.1.1, *convolutional neural networks* na secção 2.1.2, algumas técnicas de treino usadas e como avaliar os seus resultados nas secções 2.1.3 e 2.1.4 respetivamente. Na secção 2.2 é apresentado o conjunto de dados empregado nesta dissertação, bem como algumas informações existentes sobre as características avaliadas nele. Na última secção é realizada uma pesquisa sobre o estado da arte dos métodos de classificação do problema em causa, cujos resultados são discutidos.

2.1 Deep Learning

O *deep learning* é uma técnica de *machine learning* que ensina os computadores a realizar o que acontece naturalmente na lógica dos humanos: aprendizagem por exemplo. A maior parte dos métodos de *deep learning* usam *Neural Networks*, conhecidas também como *Artificial Neural Networks* (ANNs). Esta ferramenta de aprendizagem depende de dados de treino para aprender e melhorar a sua precisão ao longo do tempo. Os dados originais são

transformados em alto nível, mais abstratos por modelos de transformação simples e não-lineares e de seguida, os dados extraídos são treinados e testados no classificador.

No entanto, uma vez que estes algoritmos de aprendizagem são ajustados para ter grandes níveis de precisão, trata-se de ferramentas poderosas na ciência da computação e na inteligência artificial, permitindo classificar e agrupar dados rapidamente (IBM, 2021).

2.1.1 Neural Networks

A arquitetura das *Neural Networks* é modelada usando camadas de *artificial neurons* capazes de receber um *input*. Um exemplo de um artificial *neuron* pode ser representado pela figura seguinte:

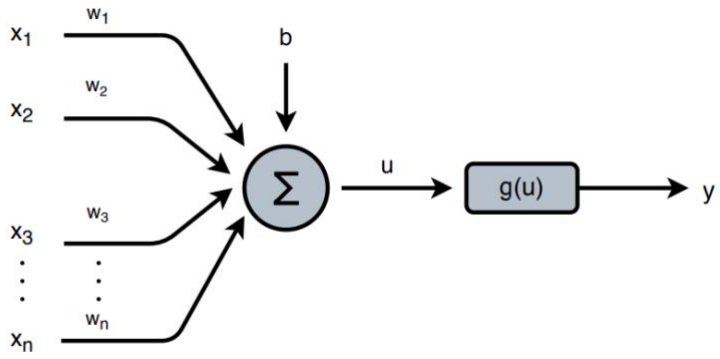


Figura 5 - Artificial Neuron

x_1, x_2, x_3, ..., x_n representam os **Inputs**, que descrevem o conjunto de características que são inseridas no modelo durante o processo de aprendizagem. Por exemplo, os Inputs no reconhecimento de objetos pode ser uma matriz de valores de pixéis relacionados a uma imagem. O w_1, w_2, w_3, ..., w_n são os **Pesos** onde a sua principal funcionalidade é dar importância aos Inputs que contribuem mais para o processo da aprendizagem, o **Bias** representado pela letra "b" é um valor constante idêntico ao de uma função linear, o "u" é o **valor de ativação,** que é o valor obtido antes de aplicar a **função de ativação** escolhida (representada pela letra g), obtendo "y" como **Output** do *g(u).*

Em termos matemáticos, o output do *artificial neuron* pode ser descrito pela seguinte fórmula:

$$y = g(u) \qquad (1)$$

$$u = \sum_{i=1}^{n} x_n * w_n + b \quad (2)$$

$$y = g(\sum_{i=1}^{n} x_n * w_n + b) \quad (3)$$

As funções de ativação permitem que pequenas mudanças nos Pesos e Bias causem apenas pequenas alterações no output. Estas são um elemento extremamente importante das *neural networks* decidindo se um *neuron* deve ser ativado ou não. Ou seja, se a informação que o *neuron* recebe é relevante ou deve ser ignorada. É importante selecionar uma função apropriada para criar uma *network* efetiva. Existem várias funções de ativação que englobam dois grupos: lineares e não lineares. A função linear, frequentemente chamada de função de ativação de identidade, é demonstrada na

Figura 6 – Função Linear e dada pela .

.

$$g(u) = u \quad (4)$$

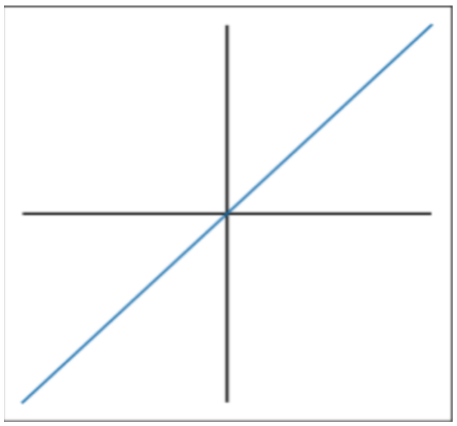


Figura 6 – Função Linear

A função linear pode ser utilizada em camadas intermediárias para fornecer uma saída linear dos dados. No entanto, não representa elevada utilidade na camada de saída de um *neuron* por não ser capaz de representar relações não lineares entre dados, o que é uma das principais vantagens das funções de ativação.

As funções de ativação não lineares são amplamente utilizadas por permitirem que as *neural networks* aprendam relações complexas e não lineares entre os dados de entrada e saída. Consequentemente, ajudam o modelo a generalizar ou adaptar à variedade de dados e diferenciam os outputs. Algumas das funções de ativação não lineares mais comuns em *neural networks* incluem a *sigmoid, hyperbolic tangent* e a função ReLU.

A função de ativação *sigmoid* é uma das funções de ativação mais antigas usadas em *neural networks*. É uma função matemática que transforma os inputs num intervalo de

outputs entre 0 e 1. A forma da função *sigmoid* é semelhante a um "S" invertido, e é definida pela Equação 5 e representada pela Figura 7:

$$g(u) = \frac{1}{(1+e^{-u})} \quad (5)$$

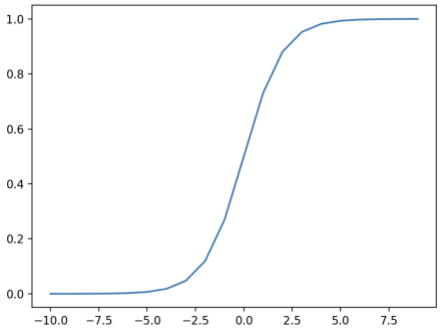


Figura 7 - Função sigmoid

Embora tenha sido amplamente utilizada no passado, tem algumas desvantagens em relação a outras funções de ativação mais recentes, como a ReLU. Por exemplo, o facto desta rede poder travar a *neural network* no tempo de treino, devido ao valor do gradiente, que é igual a zero na maioria da função. Porém, quando se deseja uma probabilidade como saída, é a função adequada, sendo frequentemente usadas em problemas de classificação binária, onde a saída da rede deve ser uma probabilidade entre 0 e 1.

A função tangente hiperbólica, é semelhante à função *sigmoid* variando apenas no seu alcance que pode ser de -1 a 1. A grande vantagem para a função anterior é que valores de *inputs* negativos são mapeados com -1 em vez de 0. Quanto maior o *input* (mais positivo), mais próximo estará o *output* do valor 1, quanto menor for o *input* (mais negativo, mais próximo estará o *output* do valor-1. Esta função de ativação é calculada pela seguinte equação e dada pela Figura 8.

$$g(u) = \frac{(e^{2u}-1)}{(e^{2u}+1)} \quad (6)$$

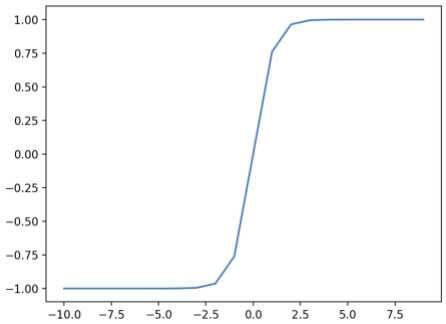


Figura 8 - Função tangente hiperbolica

A função de ativação ReLU é uma abreviação para *rectified linear unit*, ou unidade linear retificada. Produz resultados no intervalo [0, ∞[como é observado na figura seguinte.

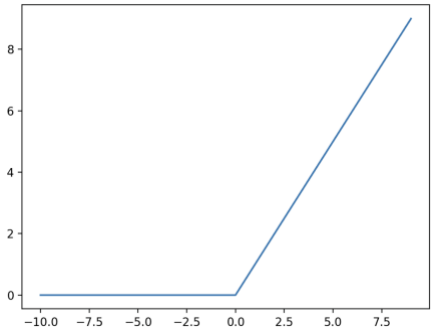


Figura 9 - Função ReLU

A função ReLU retorna 0 para todos os valores negativos e o próprio valor de input para valores positivos. É comumente utilizada por ser simples de implementar e eficaz para superar as limitações de outras funções de ativação populares, como é o caso da *SIgmoid* e da tangente hiperbólica. Como dito anteriormente o resultado está compreendido entre zero e infinito: se u < 0, g(u) = 0 e se u >= 0, g(u) = u. É definida pela seguinte equação.

$$g(u) = \max(0, u) \quad (7)$$

A função de ativação *softmax* transforma o vetor de entrada num novo vetor onde cada elemento é uma probabilidade que varia de 0 a 1 e a soma de todos os elementos é igual a 1. É frequentemente usada na camada de saída de uma *neural network* quando a tarefa é uma classificação multi-classe, em que a rede deve prever a probabilidade de pertencer a cada uma das possíveis classes de output.

$$g(u) = \frac{e^u}{\sum_{i=1}^{n} e^u} \quad (8)$$

A **Multilayer Perceptron (MLP)** é uma arquitetura ANN, sendo a mais utilizada em Neural Networks.

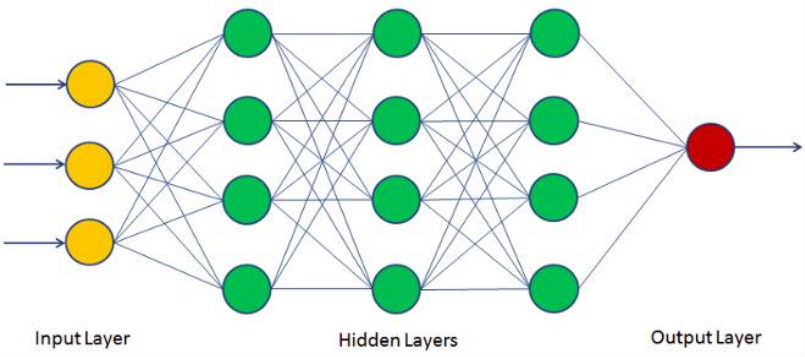


Figura 10 - Multilayer Perceptron

O ***Input Layer*** é a primeira *layer* de *neurons* deste modelo *multilayer*. A sua principal funcionalidade é receber dados de input, atribuir pesos, transformá-los usando uma função de ativação e transmiti-los à *layer* seguinte, que é normalmente a ***hidden layer*.** Estas *layers* imediatamente depois do input são chamadas de *hidden layers* por não estarem expostas ao input. Esta transforma os inputs e faz vários cálculos e extrações de recursos. Por fim, a **output *layer*** produz o resultado desejado (Idrissi & Amine, 2016).

Resumindo, como é possível observar na Figura 10, quando se recebe um input este é transformado ao longo de um número indefinido de *hidden layers*. Cada *hidden layer* é composta por um conjunto de *neurons*, onde cada um é independente e não partilha nenhuma conexão com *neurons* da mesma *layer*. No entanto, cada *neuron* dessa *layer* está totalmente conectado com todos os *neurons* das *layers* anteriores e seguintes. Por fim, a última *layer,* a "output *layer"*, apresenta o resultado da classe.

A habilidade de aprendizagem é um aspeto muito importante quando se fala em *neural networks.* Existem duas abordagens a ter em conta: aprendizagem supervisionada e sem supervisão (Pacheco et al., 2018). A aprendizagem supervisionada é uma abordagem de aprendizagem definida pelo uso de conjuntos de *datasets*. Estes conjuntos de dados são projetados para treinar ou "supervisionar" algoritmos para classificar dados ou prever resultados com precisão. Usando as entradas e saídas esperadas, o modelo pode medir a sua precisão e aprender com o tempo. Existem dois tipos de aprendizagem supervisionada: a de classificação e a de regressão, sendo a primeira a que será utilizada neste projeto. O modelo de classificação usa um algoritmo para atribuir os dados de teste em categorias especificas. Reconhece entidades específicas dentro do *dataset* e tenta tirar algumas conclusões sobre como essas entidades devem ser definidas. A ideia é fornecer ao modelo um conjunto de dados de treino que inclui exemplos rotulados, ou seja, dados em que as categorias ou classes de cada entrada são conhecidas. O processo de treino do modelo envolve o uso desses dados de treino para aprender a relação entre as características das entradas e as categorias a que pertencem. O objetivo é que o modelo aprenda a classificar as novas entradas com base na sua semelhança com as entradas de treino. De acordo com LeCun, Bengio e Hinton, o primeiro passo a ser dado é coletar um grande número de imagens com diferentes categorias selecionadas, sendo cada imagem rotulada com a sua respetiva categoria(Lecun et al., 2015). Os pesos precisam ser inicializados com alguns valores. Existem diferentes abordagens, sendo

a inicialização aleatória a mais comum. Após isso, o processo de treino começa e segue os seguintes passos:

- Treinar o modelo: Utilizar o conjunto de treino para treinar o modelo escolhido. O modelo ajustará os parâmetros para minimizar a perda e maximizar a precisão na classificação dos dados.
- Avaliar o modelo: Usar o conjunto de teste para avaliar o desempenho do modelo, ou seja, comparar as classificações do modelo com as classes verdadeiras e calcular as métricas de desempenho.
- Ajustar o modelo: Se o desempenho do modelo não for satisfatório, é necessário ajustar os pesos. Após isso, repetir o anterior e o presente passo até que o desempenho do modelo seja satisfatório.
- Usar o modelo para prever novos dados: Quando o modelo estiver treinado e tiver um bom desempenho no conjunto de teste, estará pronto a ser utilizado para prever a classe de novos dados que ainda não viu.

Em *deep learning,* existe o conceito de *loss*, que nos diz como o nosso modelo se comportou naquele preciso instante. Ao ter este indicador, é preciso treinar a rede para que ela passe a realizar as suas ações com melhor desempenho. Fundamentalmente, é necessário analisar esse indicador e tentar **minimizá-lo** ao máximo, pois quanto menor for a ***loss*** melhor será a execução do modelo. A este processo de minimizar a ***loss*** do modelo designamos por **optimização**.

Os optimizadores de algoritmos são utilizados para mudar os atributos da *neural network,* como é o caso dos pesos, para que o modelo possa reduzir a sua *loss*. É impossível saber quais devem ser os pesos do modelo numa fase inicial, mas após várias tentativas/erro baseados na *loss,* o modelo acaba por se treinar e corrigir esses pesos. Os optimizadores de algoritmos são responsáveis por reduzir as *losses* e fornecer o resultado mais preciso possível (Idrissi & Amine, 2016).

Nas *neural networks* são usados algoritmos de ***backpropagation***, com o algoritmo de ***gradient descent***. O algoritmo de *backpropagation* começa com o cálculo do custo da função gradiente (começa com a *layer* final dos pesos), que também pode ser chamada de valor de erro.

Para o ajuste de pesos, o algoritmo ***gradient descent*** é comumente utilizado. Como o vetor gradiente é a derivada do valor do erro com o vetor gradiente calculado, para cada peso, será conhecida a direção que é necessário para se mover, para atingir o mínimo custo. Fundada esta direção é preciso alterar o peso, com uma constante chamada taxa de aprendizagem e é necessário descobrir quanto deslocar o peso, para atingir o mínimos locais. Com o valor do gradiente e o passo da taxa de aprendizagem escolhidos, os valores atuais de cada parâmetro são atualizados. Isso acontecerá até que os pesos convirjam.

Por fim, é preciso **testar** a *network* com um *dataset* diferente (*dataset* de validação) do usado no processo de treino. Usar diferente inputs como teste é essencial para validar a precisão da *network* e o nível de generalização da mesma.

2.1.2 Convolutional Neural Network

Yann LeCun, diretor do *AI Research Group* do *Facebook*, é o pioneiro das *convolutional neural networks.* Construiu a primeira CNN chamada LeNet em 1988, só se tornando mais popular em 1998, quando LeCun aplicou um algoritmo de aprendizagem baseado em gradiente para as CNNs e obteve resultados bem-sucedidos para o problema de classificação de carateres, como leitura de códigos postais e dígitos.

As CNN's são muito semelhantes às *neural networks* comuns do ponto anterior. A principal diferença para as anteriores é que as arquiteturas assumem explicitamente que os *inputs* são imagens, o que nos permite codificar certas propriedades na arquitetura. Isto torna a função *forward* mais eficiente e reduz consideravelmente a quantidade de parâmetros na rede (CS231N, 2014). As CNN aproveitam o fato dos *inputs* consistirem em imagens e restringem a arquitetura de uma maneira mais sensata. Em particular, ao contrário de uma *Neural Network* regular, as *layers* de uma CNN possuem *neurons* organizados em 3 dimensões: largura, altura, profundidade como é representado na Figura 11.

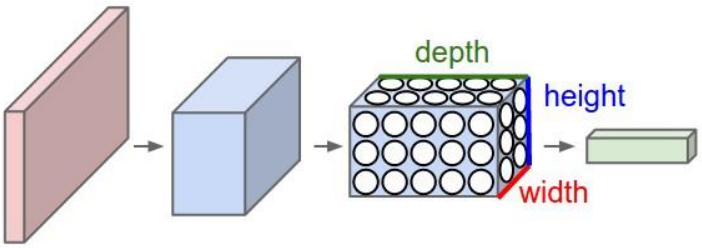


Figura 11 - Organização dos neurons em uma CNN

As CNN's transformam o volume de *input* 3D em um volume de *output* 3D de ativações de *neurons*. Neste exemplo, a layer de *input* vermelha contém a imagem, a sua largura e altura são as dimensões da mesma e a profundidade são as 3 cores RGB (Vermelho, Verde, Azul). Por exemplo, as imagens de *input* em CIFAR-10 (coleção de imagens que são comumente usadas para treinar algoritmos) têm dimensões 32x32x3 (largura, altura, profundidade, respetivamente). A *layer* de *output* final teria as dimensões 1x1x10, pois a imagem completa seria reduzida em um único vetor de pontuações de classe, organizado ao longo da dimensão de profundidade como é visto na figura anterior.

A hidden layer de uma CNN consiste no conjunto de várias *layers* diferentes que carregam uma extração de funcionalidades. Estas *layers* podem ser: *convolutional*, *nonlinear*, *pooling*, *normalization* ou *full-connected* (Alves.G, 2018).

As ***convolutional layers*** funcionam como filtros representados por matrizes e vão percorrendo toda a imagem captando os traços mais marcantes. Por exemplo, com uma imagem 32x32x3 e um filtro que cobre uma área de 5x5 da imagem, o filtro passará pela imagem inteira, formando no final um *feature map* ou *activation map* de 28x28x1.

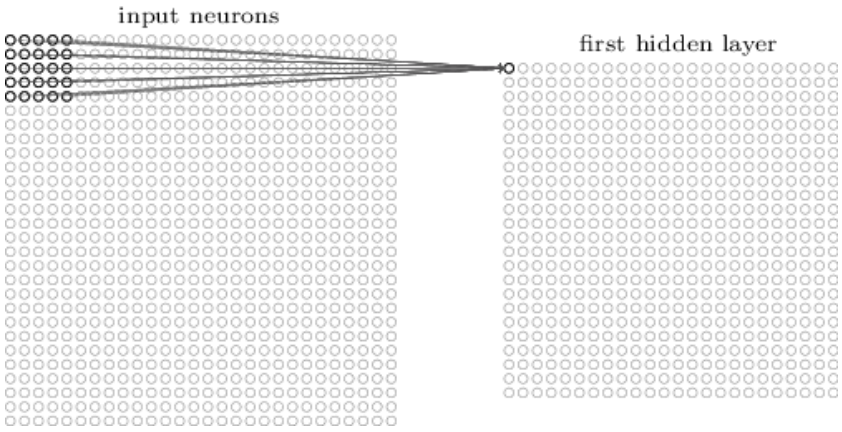


Figura 12 – Exemplo de uma operação convolucional, com um filtro de 5x5

A profundidade da *output* de uma convolução é igual a quantidade de filtros aplicados. Quanto mais profundas são as camadas das convoluções, mais detalhados são os traços identificados com o *activation map*.

As **funções de ativação** servem para trazer a não-linearidades ao sistema, para que a rede consiga aprender qualquer tipo de funcionalidade (Sharma S., 2017). A função que mais se adequa é a Relu por ser mais eficiente computacionalmente sem grandes diferenças de eficácia quando comparada a outras funções. Esta função de ativação substitui todos os valores negativos para 0 e permanece o mesmo com os valores positivos. Matematicamente pode ser expresso como na Equação 7 e o gráfico desta equação é mostrado na Figura 9 - Função ReLU.

As ***pooling layers*** servem para simplificar a informação da *layer* anterior. Assim como na convolução, é escolhida uma unidade de área, para percorrer todos os *outputs* da *layer* anterior. A unidade é responsável por resumir a informação daquela área em um único valor. Se o *output* da camada anterior for 24x24, a saída do *pooling* será 12x12. O ***max pooling*** é aplicado em cada mapa de características, dividindo-o em regiões (por exemplo, em quadrados de 2x2) e selecionando o valor máximo em cada região. O resultado é um novo mapa de características com metade do tamanho de cada dimensão, que preserva as características mais fortes do mapa de características original.

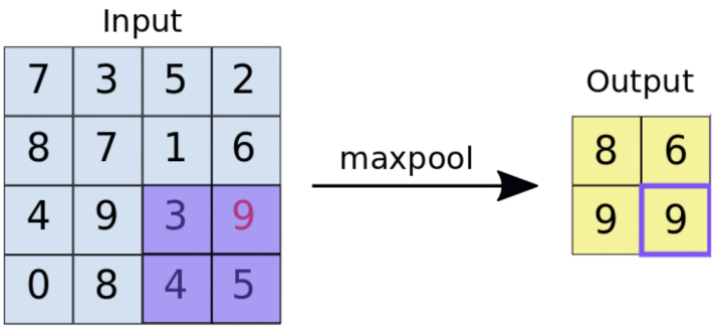


Figura 13 - Max pooling com 2x2 de área

No final da rede é colocada uma ***fully connected layer***, onde o seu *input* é o *output* da camada anterior e o seu output são *n neurons*, sendo *n* a quantidade de classes do seu modelo para finalizar a classificação.

2.1.3 Técnicas de treino

Existem muitas técnicas que podem ser aplicadas para treinar modelos de *deep learning* de forma eficaz. Aqui estão algumas técnicas populares:

- Pré-processamento e aumento de dados: dados de alta qualidade, limpos e adequadamente formatados são cruciais para o treino de modelos de *deep learning*. Técnicas de pré-processamento, como normalização de dados, redimensionamento de imagens e dimensionamento de recursos, ajudam a preparar os dados para o treino. Além disso, técnicas de aumento de dados, como recorte aleatório, inversão ou rotação de imagens, podem ajudar a gerar mais dados de treino e aumentar a capacidade de generalização do modelo.
- Regularização: Técnicas de regularização, como regularização L1 e L2 e *dropout*, ajudam a evitar o ***overfitting***. O *overfitting* é o caso em que os pesos de uma *neural network* convergem para o conjunto de dados de treino. O que significa que a rede executa com eficiência para esse conjunto de dados, mas com outro tipo de dados não funciona tão bem.
- Ajuste dos hiperparâmetros: hiperparâmetros como a ***lerning rate***, tamanho do ***batch*** e o **otimizador** podem afetar significativamente o treino de modelos de *deep learning*. O uso de técnicas como a normalização do *batch*, que visa normalizar a distribuição de outputs de cada nó numa *layer* permite que a rede seja mais estável.
- *Transfer Learning*: É útil quando os dados de treino são limitados ou quando há necessidade de melhorar o desempenho do modelo rapidamente. Ao reutilizar os pesos pré-treinados de uma *deep neural network* treinada com tarefas diferentes, mas relacionadas, o modelo pode aproveitar os recursos aprendidos anteriormente e reduzir o tempo de treino e os recursos computacionais necessários.
- Alterações da arquitetura: Alterar a arquitetura do modelo de *deep learning,* como por exemplo adicionar ou remover camadas ou alterar funções de ativação, pode afetar significativamente o desempenho do treino.

2.1.4 Avaliação

Uma ***confusion matrix*** é uma tabela usada para avaliar o desempenho de um modelo de classificação. É uma tabela com duas linhas e duas colunas que representa o número de verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos previstos pelo modelo (Sarang Narkhede, 2018).

Tabela 2 - Confusion Matrix

		Valor atual	
		Positivo	**Negativo**
Valor Previsto	**Positivo**	Verdadeiro Positivo	Falso Positivo
	Negativo	Falso Negativo	Verdadeiro Negativo

Os elementos da *confusion matrix* são definidos da seguinte forma:

- Verdadeiro Positivo (TP): O número de casos positivos previstos corretamente.
- Verdadeiro Negativo (TN): O número de casos negativos previstos corretamente.
- Falso Positivo (FP): O número de casos positivos previstos incorretamente.
- Falso Negativo (FN): O número de casos negativos previstos incorretamente.

É extremamente útil para medir o *recall*, a exatidão e a especificidade usada nas curvas AUC-ROC.

O ***recall***, também chamada de *True Positive Rate* (TPR), mede a proporção de resultados positivos reais, que são corretamente identificados como tal. Este valor deve ser o mais elevado possível e é dado pela equação seguinte:

$$Recall = \frac{TP}{TP+FN} \quad (9)$$

A **exatidão** descreve quanto um valor medido se aproxima de seu valor real. Está representado na Equação 10.

$$Exatidão = \frac{TP+TN}{TP+TN+FP+FN} \quad (10)$$

A especificidade, também conhecida como *True Negative Rate* (TNR), mede a proporção dos negativos reais que são corretamente identificados como tal. A equação da especificidade é dada por:

$$TNR = \frac{TN}{TN+FP} \quad (11)$$

"1-especificidade" é utilizado nas curvas ROC, é dada pela False Positive Rate (FPR), representada na seguinte equação.

$$FPR = 1 - TNR \quad (12)$$

Uma curva ROC *(Receiver Operating Characteristic)* é uma representação gráfica do desempenho de um modelo de classificação. É um gráfico da taxa de verdadeiros positivos (TPR) em relação à taxa de falsos positivos (FPR) em vários limites de classificação. A área sob a curva ROC é uma métrica comumente usada para avaliar o desempenho geral de um classificador. Varia de 0 a 1, onde o valor de 0,5 indica um classificador aleatório e o valor de 1 indica um classificador perfeito.

2.2 Dataset

Encontrar conjuntos de dados de imagens específicos para identificar ladrões, armas, comportamento suspeito ou até comportamento violento pode ser bastante complicado. Estes conjuntos de dados não são tão comuns quanto outros objetos ou cenas em *datasets* de imagens de rotina. Para além disso é importante considerar questões de privacidade e direitos de autor ao usar estes conjuntos de dados encontrados online e garantir que exista permissão adequada para usá-los para a finalidade pretendida.

Para este projeto, será necessário montar conjuntos de dados personalizados usando as seguintes ferramentas e fontes de dados:

- Gravações de vídeo de camaras CCTV (*Closed-Circuit Television Camera*). Antes de usar estes dados é necessário pré-processar para limpá-los, descartar o áudio, remover informação irrelevante e prepará-los para análise.
- Imagens estáticas de *datasets* encontrados online.
- Técnicas para aumento de dados. Técnicas adicionais aplicadas aleatoriamente ao conjunto de dados de treino (rotação, inversão, dimensionamento, translação, alterações de saturação de cor e corte) usando *Photoshop batch processing.*

Os conjuntos de dados para a construção de *datasets* de armas disponíveis online são limitados. As imagens de armas contidas no conjunto de dados geralmente são imagens de fontes da Internet, como imagens de uso comercial, conteúdo de vídeo de armas, filmes ou

até desenhos animados. Só um conjunto limitado de imagens de CCTV com pessoas armadas, conforme é exigido para um melhor treino do modelo e melhor eficácia de resultados.

Para além de armas, os ladrões usam máscaras para esconder a sua identidade durante a tentativa de roubo. Portanto, ao considerar este facto, se uma pessoa possuir mascara ajudará a determinar se o roubo é real ou falso. O conjunto de dados criados neste *dataset* são das fontes do *Google Images*, do *Kaggle* e do *Roboflow*. Com isto foram encontrados alguns conjuntos conforme indicado na Tabela 3.

Tabela 3 - Caracteristicas dos datasets

Dataset	**Tamanho da imagem**	**Classe**	**Número de imagens**
Deteção de pistolas	240 × 145 píxeis - 4272 × 2848 pixeis	Gun	1000
Deteção de pés-de-cabra	416×416 píxeis	Crowbar	112
Deteção de máscaras	416×416 píxeis	Mask	241
Deteção de ladrões	416x416 píxeis	Thief	5522

Em geral, treinar um modelo de visão computacional com mais dados tende a melhorar a sua capacidade de generalização, ou seja, de fazer previsões precisas em novos dados. O tamanho do conjunto de dados é um fator crítico para a precisão do modelo, quanto maior o conjunto de dados de treino, melhor será a capacidade do modelo de reconhecer novas imagens. No entanto, é importante lembrar que o tamanho do conjunto de dados pode não ser o único fator importante para o treino de um modelo de sucesso. A qualidade dos dados, a sua diversidade e a sua representatividade também são cruciais.

2.2.1 Deteção de Ladrões

Para desenvolver uma arquitetura de classificação de ladrões, foi realizada uma pesquisa. Essa pesquisa foi realizada com base em artigos, utilizando *datasets* diversos e com base em *deep learning*, mais precisamente em *convolutional neural networks* (CNNs). A escolha deste modelo deveu-se ao facto da sua deteção de objetos ter demonstrado resultados mais eficientes, escaláveis, precisos e capazes de aprender características importantes automaticamente e com mais facilidade.

2.2.1.1 Multi-task Cascaded CNN (MTCNN).

Saraswathy C propôs a utilização do *Multi-task Cascaded CNN* (MTCNN). Este modelo foi utilizado numa primeira fase para a deteção e alinhamento de rostos e numa segunda fase para monitorizar vários processos como deteção de máscara, armas, movimento e de expressão.

A deteção de rostos consiste em três módulos, cada um com sua própria CNN (Saraswathy C et al., 2021). Antes de proceder para esses módulos é necessário realizar um redimensionamento na imagem em diferentes escalas para construir uma pirâmide de imagens, que será o *input* da rede.

Modulo 1: Proposal Network (P-Net)

Este é o primeiro módulo da MTCNN e é responsável por gerar as propostas iniciais de faces na imagem de *input*. Utiliza uma CNN para detetar regiões candidatas a serem rostos na imagem, calculando uma pontuação de probabilidade para cada região.

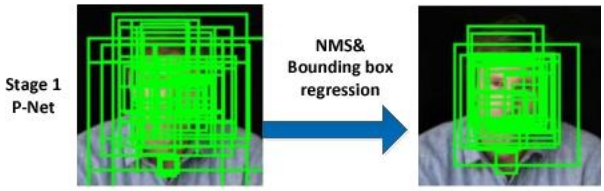


Figura 14 - Proposal Network (P-Net)

Modulo 2: Refinement Network (R-Net)

É responsável por refinar as propostas iniciais de faces geradas pelo primeiro módulo. Utiliza outra CNN para avaliar as propostas e melhorar a sua precisão. Além disso, a R-Net classifica as propostas como rosto ou não-rosto e regula as suas *bounding boxes* para ajustar as faces com maior precisão.

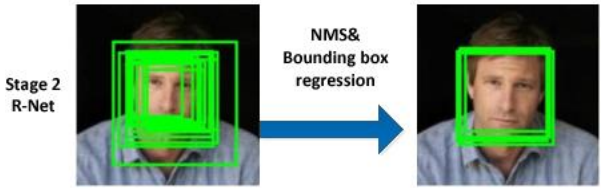


Figura 15 - Refinement Network (R-Net)

Modulo 3: Output Network (O-Net)

Este é o último módulo da MTCNN e é responsável por eliminar as propostas que não são faces. Ele utiliza outra CNN para avaliar as propostas refinadas e classificar as faces com base na sua probabilidade de serem um rosto real. A O-Net também regula as *bounding boxes* finais para ajustar as faces com a maior precisão possível.

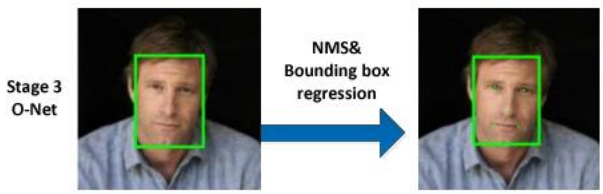


Figura 16 - Output Network (O-Net)

A deteção de movimento é muito importante para detetar a ação e o movimento da pessoa. Após o vídeo em *real time* ser convertido em *frames*, é dado ao filtro *contour*. O *contour* é útil para análise de forma, detetando e reconhecendo objetos. Aqui, desempenha um papel principal de encontrar a forma do objeto em movimento na imagem, juntando todos os pontos contínuos na forma de curva fechada. Depois de encontrar a forma do objeto, ela é enviada para a parte temporal do modelo para o próximo processo. De seguida, é comparado com os conjuntos de dados para detetar o movimento da pessoa. Onde, esses conjuntos de dados consistem no movimento e ação realizados pelo ladrão. Ao comparar com esse conjunto de dados, deteta com precisão o movimento e ação realizados pelo ladrão quando comparado com uma pessoa normal.

A deteção de expressão é aplicada após os passos anteriores para encontrar a expressão facial feita pela pessoa quando ela está de frente para a camara. O filtro *contour* desempenha um papel importante na descoberta da visão geral do objeto. Localiza o rosto e, em seguida, fornece à parte temporal a tarefa de identificar a expressão. Analisará a expressão feita pelo rosto comparando-a com o conjunto de dados. Este conjunto de dados consiste em expressões como raiva, medo, felicidade, tristeza e neutro. Após a comparação com o conjunto de dados, a expressão da pessoa pode ser identificada.

A deteção de armas é uma parte importante na identificação de ladrões, para mostrar a diferença entre uma pessoa normal e um ladrão. O vídeo em *real time* é usado como entrada e convertido em *frames*. A filtro *spatial* é utilizado para localizar a visão geral do objeto em movimento e negligenciar a parte restante no *frame*. De seguida, é comparado com o conjunto de dados pré-definido e classifica se é uma faca ou uma arma. O *dataset* pré-definido consiste em imagens de facas e armas. Após a comparação, conclui-se se a pessoa possui uma arma ou não.

Devido à situação do Covid, tornou-se uma situação comum o uso de máscara. O *output* da deteção de máscara ajuda a perceber se a situação presente é um roubo ou não. O processo de deteção de máscara é semelhante à deteção de armas. O *dataset* pré-definido consiste em imagens de máscaras de ladrões. Após a comparação, conclui-se se é ladrão ou não.

2.2.1.2 Faster R-CNN

Faster R-CNN é uma técnica de deteção de objetos em imagens e, portanto, pode ser usada para deteção de ladrões em sistemas de vigilância por vídeo. É capaz de identificar facilmente qualquer tipo de objeto numa cena e localizá-los com precisão em tempo real, o que pode ser útil para identificar pessoas suspeitas de roubo ou furto num ambiente monitorizado por camaras. No entanto, é importante ter em conta que a precisão da *Faster* R-CNN depende da qualidade das imagens de *input*. O processo geral pode ser dividido em três etapas:

1. Proposição de região: O modelo usa uma CNN para gerar uma série de regiões candidatas, ou "*bounding boxes*", que podem conter objetos. Estas regiões candidatas são geradas em diferentes tamanhos e posições na imagem de entrada.

2. Extração das características: Para cada região candidata, a *Faster R-CNN* extrai as características relevantes da imagem de *input* usando a CNN.
3. Classificação e refinamento: Com base nas características extraídas, o modelo classifica cada região candidata como uma *bounding box* contendo um objeto ou não. Se a *bounding box* for classificada como contendo um objeto, a Faster R-CNN refina a localização e o tamanho da caixa delimitadora para obter uma melhor precisão.

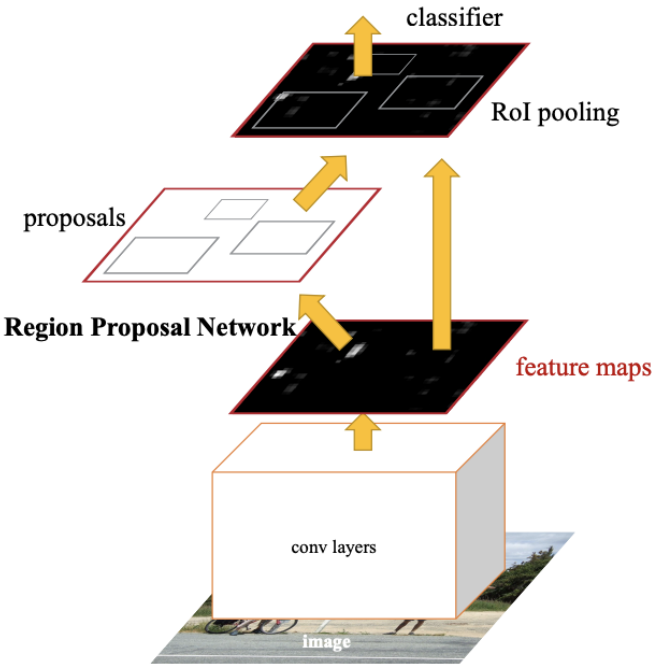


Figura 17 - Faster R-CNN

Para deteção de um ladrão, este modelo é treinado num conjunto de dados que contém imagens rotuladas como ladrões ou pessoas suspeitas de cometer roubo ou furto. Durante a deteção, analisaria cada *frame* de um vídeo e geraria uma série de regiões candidatas para objetos na cena. Em seguida, extrairia as características de cada região candidata e classificaria as *bounding boxes* correspondentes. Se uma região candidata for classificada como contendo um ladrão, o modelo refinaria a localização e o tamanho da caixa delimitadora para obter uma melhor precisão (Kakadiya et al., 2019).

2.3 Discussão

Nesta seção, são apresentadas as justificações da escolha da linguagem de programação e uma discussão sobre as conclusões do trabalho pesquisado.

C e *python* são duas linguagens de programação populares, mas têm diferentes pontos fortes e fracos dependendo da tarefa em questão. *C* é uma linguagem de programação de baixo nível que é usada para a programação de sistemas, sistemas operacionais e computação de alta performance. *C* é mais rápido que *python* devido ao seu acesso de baixo nível aos recursos do computador e hardware, mas requer mais esforço para escrever código. *python*, por outro lado, é uma linguagem de programação de alto nível que é usada para programação de desenvolvimento *web*, computação científica e análise de dados. *python* é mais fácil de escrever código devido à sua sintaxe de alto nível e tipagem dinâmica, mas é mais lento que C porque é uma linguagem interpretada.

Devido a este fator concluo que a ferramenta ideal a ser utilizada seria a linguagem *C* por ser menos pesada para os *Raspberry Pi's* utilizados na máquina. Contudo, devido à falta de tempo proposto para o projeto e elevada documentação que a linguagem *python* possui, é mais rentável a sua utilização para o desenvolvimento desta solução.

O *MTCNN* e *Faster R-CNN* são dois algoritmos diferentes usados em tarefas de deteção de objetos em imagens. O primeiro, é uma *CNN* desenvolvida para detetar principalmente rostos e as suas respetivas emoções. Uma das vantagens do *MTCNN* é que este algoritmo é capaz de lidar com uma ampla variedade de condições de iluminação, ângulos e poses faciais. O *Faster R-CNN* é um algoritmo de deteção de objetos, que usa uma *CNN* para extrair características da imagem e um classificador para identificar objetos de interesse.

No problema em questão, o algoritmo *Faster* R-CNN seria responsável por detetar e localizar os objetos de interesse (por exemplo, armas) na imagem, enquanto a MTCNN seria responsável por detetar as emoções nas expressões faciais. Juntas, estas técnicas poderiam ser usadas para detetar ladrões e monitorizar emoções relevantes em tempo real. Com isto é possível concluir que a combinação destes dois algoritmos pode oferecer um algoritmo mais completo e com melhores resultados.

2.3.1 Sumário

O *deep learning* provou superar os resultados na classificação de ladrões em comparação com métodos de classificação tradicionais. A maioria dos modelos de *deep learning* são baseados em *neural networks*. O MLP é o modelo mais comumente usado. As *neural networks* normalmente são compostas por uma camada de *input* e uma de *output*, com camadas *hidden* adicionadas com base na complexidade do problema. Depois dos dados serem passados por essas camadas, os *neurons* aprendem e identificam padrões. Esse modelo é treinado e após isso solicita à rede que encontre as previsões com base nos dados de teste. Por outro lado, as *CNNs* são um tipo especial de *neural network* que funciona excecionalmente bem em imagens, são uma MLP que foi estendida pelo espaço, usando pesos compartilhados.

Não existem *datasets* para identificarem todas as características dos ladrões, por isso foi necessário criar um conjunto com classes de vários objetos e expressões faciais de *datasets* encontrados na internet.

Após uma busca restrita por trabalhos de *deep learning* na literatura, dois que se enquadravam nas restrições foram analisados e discutidos: o algoritmo *MTCNN* e *Faster R-CNN.* O *MTCNN* é um algoritmo *CNN* desenvolvido para detetar principalmente rostos e as suas respetivas emoções. O *Faster R-CNN* é um algoritmo de deteção de objetos, que usa uma *CNN* para extrair características da imagem e um classificador para identificar objetos de interesse. Juntas, estas técnicas poderiam ser usadas para detetar ladrões e monitorizar emoções relevantes em tempo real.

3 Metodologia

O propósito deste capítulo é apresentar a metodologia adotada. Por conseguinte, é referida a hipótese do projeto, o conjunto de dados utilizados para a construção do *software*, uma breve apresentação da abordagem da construção dos modelos e a sua avaliação individual. Na figura seguinte está representada a sequência de etapas planeadas para a execução deste projeto.

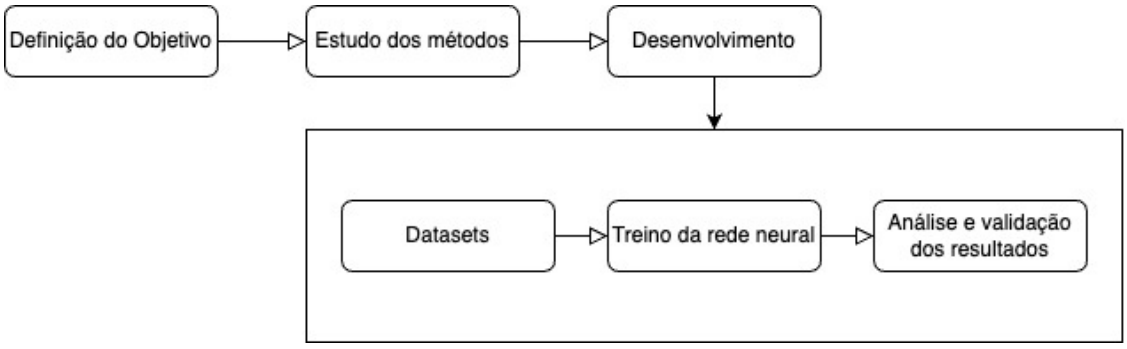


Figura 18 - Metodologia adotada para o desenvolvimento do projeto

3.1 Hipótese

A união de três modelos de inteligência artificial: reconhecimento de emoções, reconhecimento de objetos e reconhecimento de ações agressivas num sistema de software para deteção de vandalismo e ladrões aumentará significativamente a precisão e eficácia na identificação e categorização de incidentes de vandalismo e roubo em tempo real.

3.2 Datasets

Este projeto irá utilizar três conjuntos de dados distintos para a realização da pesquisa. O primeiro conjunto de dados é empregue na identificação de expressões faciais, visando compreender e analisar as emoções expressas pelos sujeitos estudados. O segundo conjunto de dados será utilizado para a deteção de objetos, possibilitando, desta forma, a identificação e classificação de objetos presentes nos vídeos analisados. Por fim, o terceiro conjunto de dados será aplicado na deteção de ações agressivas, com o objetivo de identificar comportamentos agressivos ou violentos.

3.2.1 Origem dos dados

Para a construção deste projeto serão utilizados conjuntos de dados do *Kaggle* e do *Roboflow* e serão construídos novos conjuntos por meio da captura e processamento de marcadores corporais de vídeos em tempo real.

As plataformas de *datasets* para além de permitirem o acesso a conjuntos de dados, permitem também a junção de conjuntos previamente preparados por outros investigadores, abrangendo uma variedade de domínios e promovendo um treino mais completo e profundo. Por outro lado, devido à inexistência de dados relativos a ações de violência, este estudo tem também como objetivo a construção de um novo *dataset* no qual essas ações deverão apresentar-se devidamente classificadas e identificadas. Esta tarefa é realizada por meio de captura e processamento dos *landmarks* do corpo humano presente nos vídeos de tempo real.

Os resultados obtidos a partir desta abordagem híbrida oferecem uma visão mais completa do caso em estudo, agregando valor significativo à pesquisa.

3.3 Construção e avaliação dos modelos

3.3.1 Reconhecimento de emoções

Para realizar o treino do modelo de reconhecimento de emoções, será utilizada uma abordagem de aprendizagem supervisionada, na qual cada classe de dados será dividida em conjuntos de treino e teste. O modelo será treinado com uma *convolutional neural network* (CNNs). Após o treino do modelo, este será avaliado utilizando métricas como a *accuracy*, a *loss, recall* e *f1-score* comparando-os com outros modelos existentes. Para além disso, será também realizada uma visualização das previsões do modelo em dados de teste, permitindo uma análise do desempenho do modelo, mostrando a sua capacidade de fazer previsões precisas em dados não vistos anteriormente.

3.3.2 Reconhecimento de objetos

Para a construção do modelo de reconhecimento de objetos será utilizado o *YOLOv8*. Inicialmente, será conduzido um rigoroso pré-processamento dos dados, realizando ajustes de tamanho, normalização e aumento de dados para garantir que o modelo seja treinado com informações variadas e de forma adequada. Seguidamente, a arquitetura *YOLO* será adaptada e personalizada para o contexto específico da aplicação, considerando a deteção de objetos específicos e a otimização de hiperparâmetros. Após o treino do modelo, a avaliação será realizada em duas fases. A primeira fase consistirá numa avaliação quantitativa, onde o modelo será testado num conjunto de dados de teste e, por conseguinte, obter-se-á as métricas de desempenho que possibilitarão a medição da capacidade do modelo em detetar com precisão os objetos de interesse. A segunda fase de avaliação envolverá uma análise qualitativa, onde amostras de imagens do conjunto de teste serão analisadas visualmente, permitindo uma avaliação da qualidade das deteções e a identificação de possíveis falhas.

3.3.3 Reconhecimento de comportamento agressivo

Após a construção do *dataset* com as *landmarks* composto por pontos-chave do corpo, será construído o modelo de deteção de comportamento agressivo baseado na arquitetura *LSTM*. Com o intuito de efetuar a análise e avaliação do modelo, serão utilizadas métricas padrão como *accuracy*, *precision*, *recall* e *F1-score*. Além disso, serão realizadas análises visuais e interpretações qualitativas para compreender os padrões e características identificados pelo modelo *LSTM* e verificar se o modelo funciona com êxito.

3.4 Ambiente de desenvolvimento

Este estudo foi conduzido utilizando o *Google Colab* como plataforma principal de desenvolvimento. A decisão da utilização deste ambiente de desenvolvimento colaborativo foi sustentada pelas seguintes vantagens:

- Ser grátis.
- Integração com o *github*
- O código ser executado nos servers da Google o que fornece elevado desempenho.
- Não necessitar de qualquer instalação.
- O sistema de partilha facilita a partilha do trabalho realizado.
- Correr na *cloud* permite o acesso simplificado a recursos de computação grátis.
- Compatibility with Jupyter Notebook allowing a better organization of the code.

4 Desenho da solução

Nesta secção é apresentada a arquitetura da solução, a escolha da linguagem de programação e da *framework* de *deep learning* que será utilizada e informação acerca da visualização do sistema de um ponto de vista estrutural, uma análise das funcionalidades e a solução para resolver o problema.

4.1 Linguagem e Framework de Deep Learning

A linguagem escolhida para a realização deste projeto foi o python (Python, 2023) considerando a quantidade de bibliotecas disponiveis para trabalhar com *deep learning*. Na secção 24 existe mais informação acerca desta linguagem e das suas vantagens.

Em relação à framework que será utilizada foi escolhido o *Keras* (Keras, 2023), visto ser uma excelente ferramenta tendo em conta a rapidez de construção e de experimentação de modelos de *deep learning*. Em combinação com o *Keras* será utilizado o *Tensorflow* (Tensorflow, 2023) que facilita a produção de sistemas em grande escala que requerem elevada escalabilidade, robustez. Além disto, a agregação destas duas *frameworks* fornecem ferramentas ao sistema para a alteração dos seus parâmetros para melhor atender às necessidades do projeto.

4.2 Análise de requisitos

O software para deteção de ladrões deve ser capaz de realizar a análise de vídeo em tempo real de uma câmara previamente instalada, utilizando técnicas avançadas de processamento de imagens e *deep learning*. Entre as funcionalidades necessárias estão a deteção de comportamento agressivo, reconhecimento de emoções para identificar as sensações e sentimentos das pessoas analisadas e o reconhecimento e classificação de

objetos. O sistema deve ser altamente responsivo, garantindo uma deteção rápida e precisa de atividades criminosas.

A escalabilidade é fundamental, permitindo que o sistema lide com grandes volumes de dados e se adapte a diferentes ambientes. A eficiência computacional também é importante para garantir que o software possa ser implantado em diversos dispositivos, sem comprometer o desempenho geral. Além disso, o *software* deve ser projetado para ser altamente preciso e confiável, minimizando alarmes falsos e garantindo que apenas atividades criminosas legítimas sejam identificadas.

4.2.1 Requisitos funcionais

Depois da análise feita acima, foram identificados 3 casos de uso:

- Deteção de vandalismo em tempo real - O software deve ser capaz de analisar em tempo real, através de uma camara, para identificar comportamentos suspeitos associados a atividades de roubo ou vandalismo.
- Alertas - O sistema deve fornecer alertas em tempo real para os proprietários, quando atividades suspeitas forem detetadas, permitindo uma resposta rápida.
- Guardar intervalos de vídeos com atividades suspeitas em tempo real.

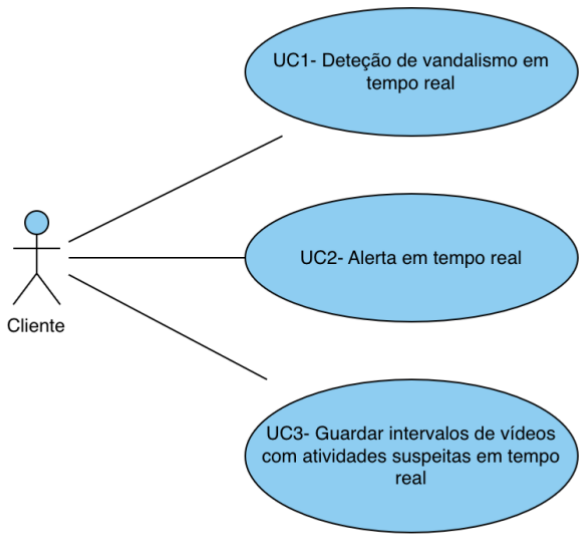


Figura 19-Diagrama de casos de uso

4.2.2 Requisitos não funcionais

Para avaliar os requisitos não funcionais, foi utilizado o modelo FURPS+. Este modelo categoriza os requisitos não funcionais em mais de 5 categorias: funcionalidade, usabilidade, confiabilidade, desempenho e capacidade de suporte. O "*plus*" refere-se a restrições de design, requisitos de implementação, requisitos de interface e requisitos físicos. A tabela seguinte contém os requisitos não funcionais identificados, organizados por categoria.

Tabela 4 - Requisitos não funcionais

Código	**Funcionalidade**	**Tipo**
F01	O sistema deve ser otimizado para utilizar recursos computacionais de forma eficiente.	Funcional
U02	O software deve fornecer feedback visual claro sobre a deteção de atividades suspeitas e a interação do usuário com o sistema.	Usabilidade
R01	O software deve ser altamente preciso, minimizando alarmes falsos e garantindo a confiabilidade dos resultados.	Confiabilidade
R02	O sistema deve estar disponível de forma confiável durante todo o tempo de operação, evitando períodos de inatividade que possam comprometer a segurança.	Confiabilidade
P01	O sistema deve ser altamente responsivo para garantir deteção e resposta rápidas a atividades criminosas em tempo real.	Desempenho
S01	O software deve ser de fácil manutenção e atualização	Suporte

O requisito físico para construir modelos utilizando técnicas de *deep learning* é o uso das unidades de processamento do computador, as GPUs. A GPU utiliza milhares de núcleos menores e mais eficientes para uma arquitetura massivamente paralela que lida com múltiplas funções simultaneamente. As GPUs modernas oferecem uma potência de processamento e eficiência superiores em comparação com suas contrapartes, as CPUs. Os requisitos de implementação são que alguns dos *datasets* utilizados nesta dissertação provêm de um conjunto de dados públicos, que possuem dados previamente rotulados e, por essa razão, ajudam a reduzir os custos iniciais do projeto. Idealmente, a criação de um conjunto de dados foi posta em causa, no entanto, foi decidido que um conjunto de dados público seria uma opção melhor para alcançar resultados de alto desempenho no treino do modelo durante as etapas de desenvolvimento e avaliação do sistema.

4.3 Desenho do sistema

A arquitetura proposta encontra-se essencialmente divida em quatro módulos: aquisição de dados, deteção de movimento, identificação e classificação através dos modelos de *deep learning* e ações a serem realizadas. O fluxo começa com a captura e pré-processamento de dados provenientes da câmera. Antes da classificação das características, o sistema realiza uma verificação da existência de movimento e, caso seja detetado, direciona o fluxo para a etapa seguinte, onde são extraídas várias características relevantes das imagens, permitindo a deteção de padrões associados a atividades suspeitas. Por fim, quando um ladrão é identificado, a arquitetura ativa uma função alarmística e armazena automaticamente um trecho do vídeo na base de dados, permitindo uma análise posterior e fornecendo evidências valiosas em caso de investigações.

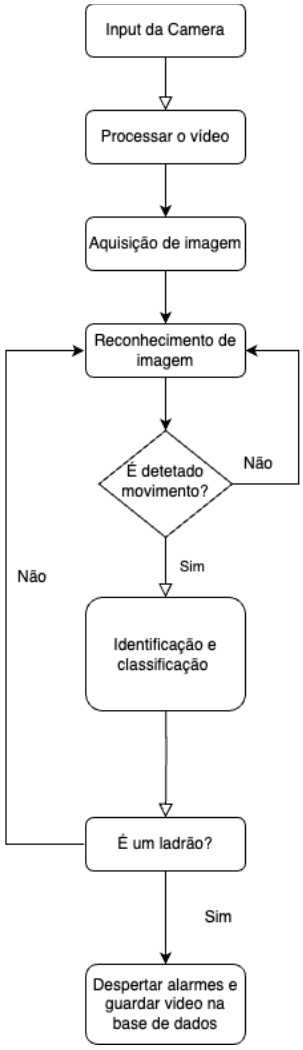


Figura 20 - Fluxo dos módulos que compõe o sistema anti-roubo

4.3.1 Módulo de aquisição de imagens

O primeiro módulo é responsável pela aquisição de imagens por meio uma câmera. Este módulo coleta imagens em tempo real e extrai periodicamente uma cena para enviar ao próximo módulo. Decidiu-se utilizar imagens periodicamente para não sobrecarregar o processador, sendo ainda possível classificar e identificar todas as características representadas pelo utilizador com uma frequência de *frames* adequada. O intervalo entre cada extração será determinado por um parâmetro, que também determinará a frequência das classificações. Para a implementação deste módulo, será utilizado o *OpenCV*.

4.3.2 Módulo de deteção de movimento

Após a aquisição da imagem, é aplicada a técnica de *background subtraction*, uma abordagem utilizada para o reconhecimento de movimento. Inicialmente, o algoritmo cria um modelo de fundo estático da cena ao analisar a primeira imagem de vídeo. Este modelo representa a estrutura da cena sem a presença de objetos em movimento. Em seguida, à medida que as imagens subsequentes são processadas, o algoritmo compara cada pixel com seu respetivo valor no modelo de fundo. As diferenças significativas entre os valores do pixel atual e os valores do modelo são interpretadas como movimento, indicando a presença de objetos em movimento na imagem. O uso do *background subtraction* permite que o módulo de deteção de movimento isole efetivamente as regiões em movimento, filtrando o ruído e destacando apenas os objetos que se estão a deslocar na cena. Para a implementação deste módulo, será utilizado o *OpenCV*.

4.3.3 Módulo dos modelos de classificação

O módulo dos modelos de classificação é responsável por identificar e classificar os padrões relevantes nos dados de entrada. Neste contexto, são consideradas duas alternativas: a primeira opção consiste em implementar três modelos distintos - o modelo de reconhecimento de expressões faciais, o modelo de deteção de objetos e o modelo de deteção de ações agressivas. Cada modelo terá um foco específico e será treinado para desempenhar a sua tarefa de forma otimizada. O modelo de reconhecimento de expressões faciais irá analisar e identificar as emoções presentes nas imagens faciais capturadas, como alegria, tristeza, raiva, entre outras. O modelo de deteção de objetos será responsável por identificar a presença de objetos relevantes no ambiente, como armas, objetos suspeitos, ou materiais que possam representar uma ameaça. O modelo de ações agressivas identifica se a pessoa em análise está ou não a tentar vandalizar a propriedade.

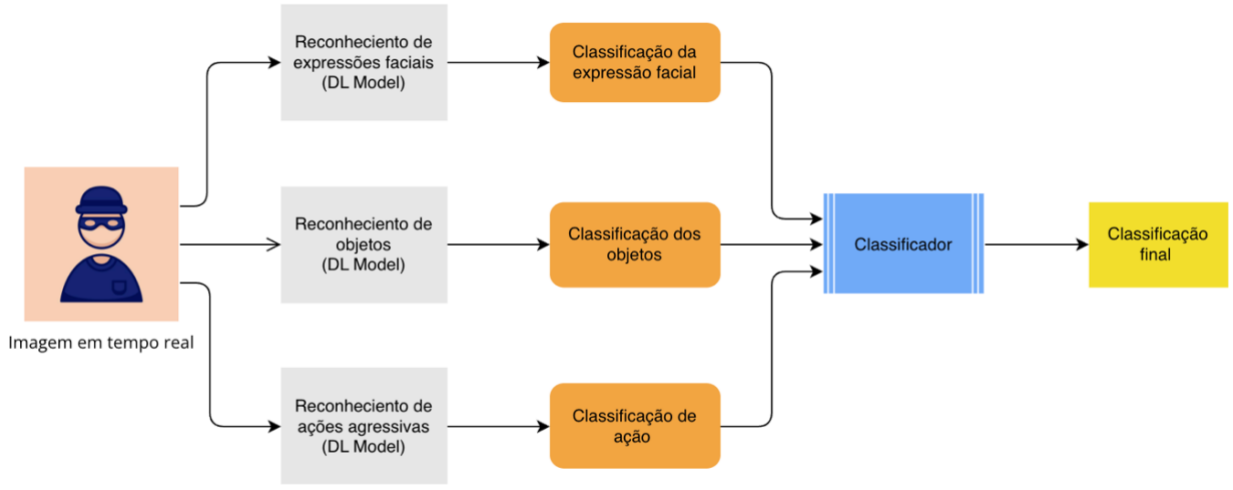


Figura 21 - Arquitetura dos modelos de classificação (alternativa 1)

Por outro lado, a segunda alternativa considera a utilização de um modelo único que combina os três domínios mencionados anteriormente. Esta abordagem busca criar um modelo integrado capaz de realizar todas as tarefas em conjunto, aproveitando as sinergias entre as diferentes classificações.

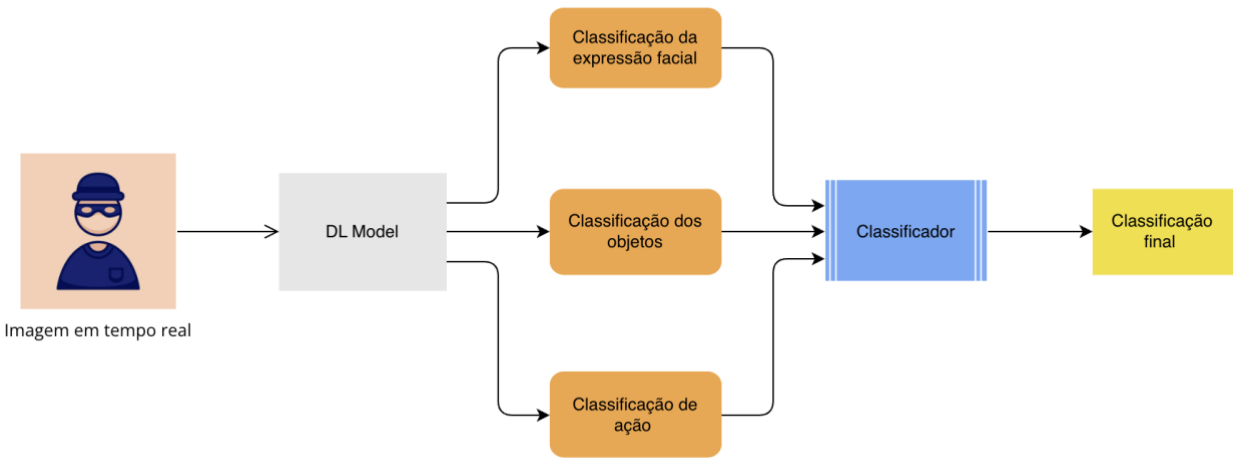
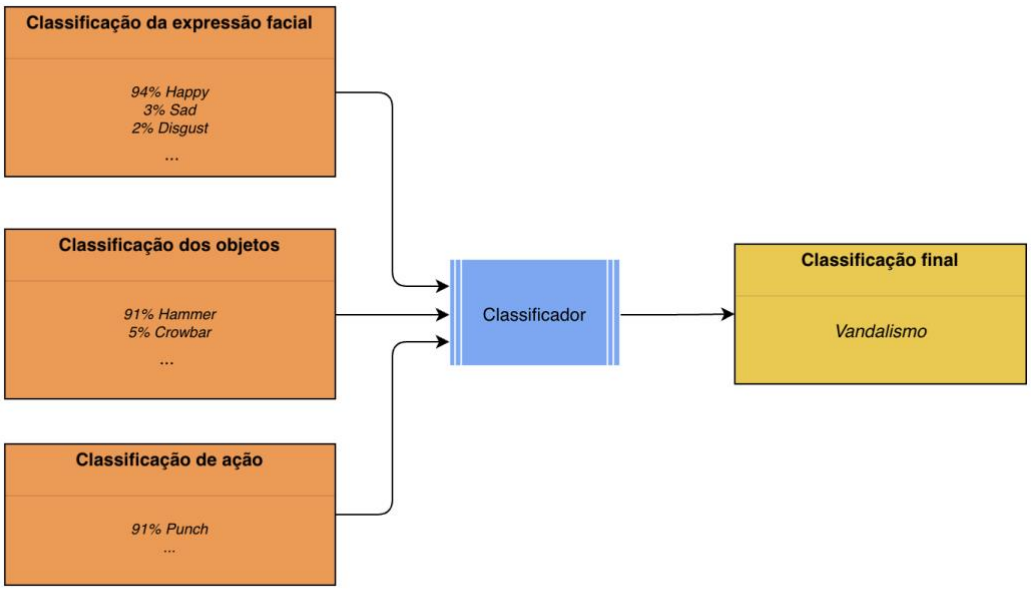


Figura 22 - Arquitetura dos modelos de classificação (alternativa 2)

As alternativas mencionadas anteriormente utilizam o mesmo modelo de classificação. Ambas possuem um classificador que irá combinar as previsões de cada classificador individual e inferir se a pessoa que está a ser visualizada tem ou não intenções de vandalizar.



Quando a imagem é processada pelos modelos, as classificações necessitam de ser unificadas para fornecer uma resposta. Cada um dos três classificadores apresenta as suas previsões e, posteriormente essas previsões agregadas, ponderadas pelas percentagens associadas a cada classe, fornecem uma visão geral da probabilidade de a pessoa estar envolvida em atividades de vandalismo.

A decisão sobre a abordagem a adotar depende das necessidades do sistema, dos recursos disponíveis e das metas de desempenho desejadas. Ambas as alternativas têm vantagens e desvantagens, sendo por isso importante realizar análises comparativas para determinar qual é a solução que melhor se adequa ao contexto da aplicação. Usando os três modelos de *deep learning* separados:

Vantagens:

- Especialização - Cada modelo é treinado para uma tarefa específica (expressões faciais, objetos suspeitos e ações agressivas), o que pode resultar num melhor desempenho e precisão de cada tarefa.
- Fácil adaptação - Caso seja necessário atualizar ou aprimorar um dos modelos para melhorar a deteção de alguma característica específica, é possível fazê-lo sem afetar os outros modelos.

Desvantagens:

- Complexidade – Possuir três modelos separados pode aumentar a complexidade do sistema como um todo, resultando num maior uso de recursos computacionais.
- Integração - Integrar e coordenar os três modelos distintos num sistema é mais desafiador do que trabalhar com um único modelo.

Usando um único modelo de *deep learning* com todas as características:

Vantagens:

- Simplificação: Utilizar um único modelo pode simplificar o sistema, reduzindo a complexidade geral.
- Eficiência: Pode reduzir a latência, já que todas as características podem ser processadas numa única passagem pela rede.

Desvantagens:

- Desempenho: O desempenho geral do modelo pode ser afetado pela complexidade de combinar todas as características num único sistema. Algumas tarefas podem ser mais complexas na sua aprendizagem, o que pode afetar a precisão global do modelo.
- Dificuldade de atualização: Se uma das características precisar de atualização ou melhoria, isso implica alterar o sistema global, exigindo um treino conjunto e potencialmente mais complexo.

Em conclusão, ao avaliar as vantagens e desvantagens das abordagens de utilizar três modelos de *deep learning* especializados em comparação com um único modelo abrangente, constata-se que ambas as opções têm méritos significativos. No entanto, após uma análise das necessidades específicas do nosso projeto, decidiu-se adotar a primeira alternativa, que envolve a implementação de três modelos distintos.

Essa escolha é fundamentada na busca por um alto nível de especialização em cada tarefa de deteção. A especialização aprimorará significativamente a precisão e o desempenho geral do sistema, contribuindo para resultados mais confiáveis o que, para este projeto, é uma necessidade fundamental. Além disso e embora seja um fator importante, a latência não será afetada de maneira significativa pela utilização dos três modelos separados.

4.3.4 Módulo das ações

Após os modelos de *deep learning* produzirem resultados, ações imediatas são executadas: um alerta é emitido e um trecho relevante do vídeo será armazenado na base de dados para análise posterior. Isto permite notificar o cliente da deteção de um possível vandalismo e garante dados para investigações futuras.

4.3.5 Diagrama de sequência

Na figura seguinte, encontra-se representado o diagrama de sequência abrangendo todos os módulos.

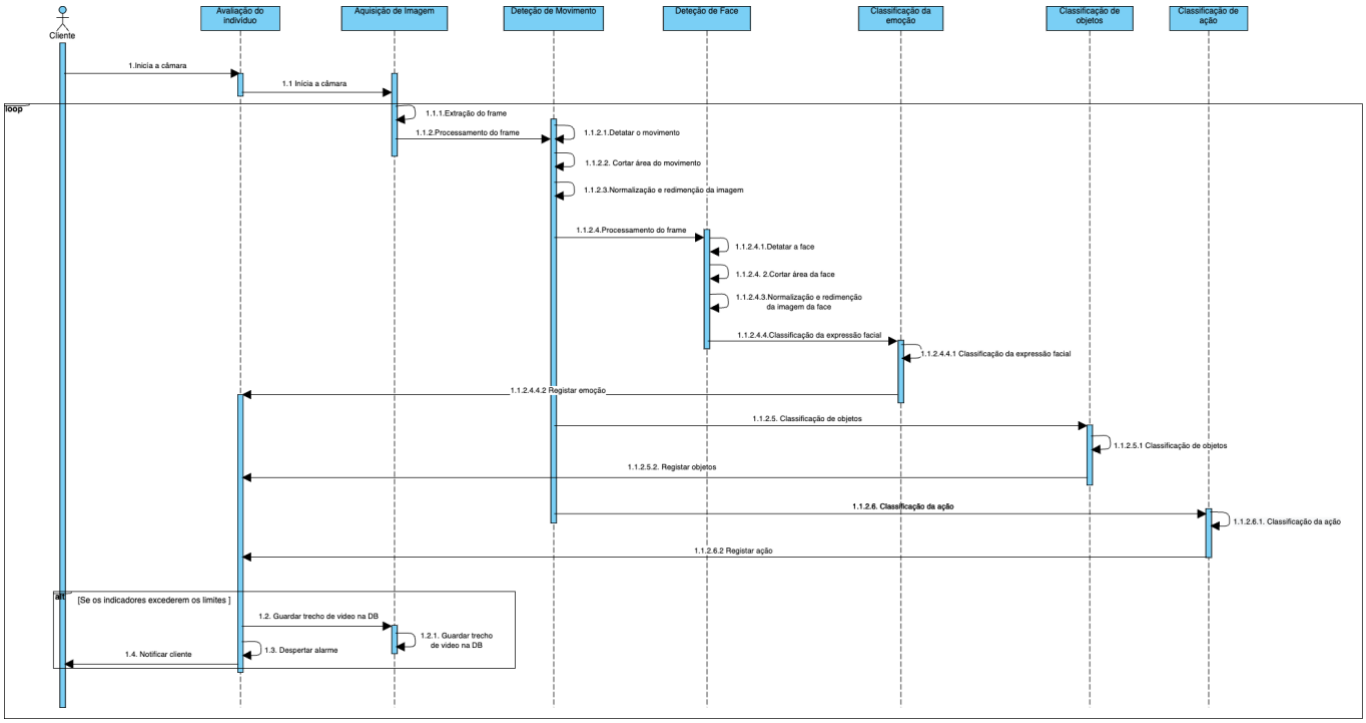


Figura 23 - Diagrama de sequência

5 Implementação

O conteúdo deste capítulo dá visibilidade de como foi realizada a implementação, tendo em consideração a abordagem definida anteriormente neste documento, principalmente no capítulo do desenho da solução. Para além disso, fornece também detalhes técnicos sobre as decisões tomadas durante a fase de execução. A implementação encontra-se essencialmente divida em duas partes: a primeira referente à construção dos modelos e a segunda referente à integração dos modelos no sistema.

5.1 Construção dos modelos

A aprendizagem supervisionada usa conjuntos de treino para ensinar o modelo a executar uma tarefa ou prever um valor (ou valores). É importante lembrar que estes dados de treino precisam ser denominados com o resultado esperado/resposta certa para cada exemplo individual no conjunto. Grande parte dos *datasets* existentes *online* já contém essa rotulação, como é o caso do *dataset* utilizado para a deteção de expressões faciais, mas quando se trata da criação de um novo conjunto de dados, assim como o da deteção de objetos e de ações agressivas, é necessário algum esforço adicional. Na prática, é preciso extrair 3 subconjuntos destes dados rotulados originais: os conjuntos de treino, validação e teste. Esta é uma etapa importante para avaliar o desempenho de diferentes modelos e o efeito do ajuste de hiperparâmetros.

5.1.1 Construção dos conjuntos de dados

Como referido anteriormente, o conjunto de dados para a deteção de objetos e de ações agressivas necessitou de um esforço maior para a sua construção. O primeiro foi composto pela junção dos conjuntos de dados individuais, cada um correspondente a diferentes categorias de objetos e contendo em cada categoria os conjuntos de treino, validação e teste.

O segundo foi elaborado através de um sistema que consegue identificar e seguir a postura do corpo humano. O sistema analisa continuamente o que a câmera está a captar e, em cada imagem, localiza pontos-chave que representam diferentes partes do corpo. À medida que as imagens são processadas, os pontos-chave são registados para criar um conjunto de dados. Estes dados são guardados num formato *.txt* que permite a futura utilização no sistema. A seguir estão representadas as funções utilizadas para guardar os *landmarks* nos ficheiros *.txt*:

```
import cv2
import mediapipe as mp
import pandas as pd

cap = cv2.VideoCapture(0)

mpPose = mp.solutions.pose
pose = mpPose.Pose()
mpDraw = mp.solutions.drawing_utils

lm_list = []
label = "neutral2"
no_of_frames = 600
i=0

while len(lm_list)<=no_of_frames:
    ret, frame = cap.read()
    if ret:
        frameRGB = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        results = pose.process(frameRGB)
        if results.pose_landmarks:
            lm = make_landmark_timestep(results)
            lm_list.append(lm)

        if cv2.waitKey(1) == ord('q'):
            break

df = pd.DataFrame(lm_list)
df.to_csv(label+".txt")
cap.release()
cv2.destroyAllWindows()
```

Código 1 – Criação dos dados

```
def make_landmark_timestep(results):
    c_lm = []
    for id, lm in enumerate(results.pose_landmarks.landmark):
        c_lm.append(lm.x)
        c_lm.append(lm.y)
        c_lm.append(lm.z)
        c_lm.append(lm.visibility)
    return c_lm
```

Código 2 – Função para a criação dos *landmarks*

O código acima começa por capturar vídeo da câmera em tempo real e utiliza as bibliotecas *OpenCV*, *mediapipe* e *pandas* para detetar a postura humana. Processa os *frames* do vídeo, identificando os *landmarks* da pose através do objeto *mpPose*. Na existência de *landmarks*, o código extrai as coordenadas x, y, z para cada ponto, armazenando-os em sequência, sendo posteriormente armazenados numa lista chamada *lm_list*. O processo é repetido até que o número desejado de *frames* seja capturado. Após isso, os dados são convertidos num *dataframe* do *pandas*, que é exportado para um arquivo de texto com base numa variável previamente definida, que tem por nome a ação executada. O código encerra a captura de vídeo e fecha as janelas abertas. O processo acima necessita de ser realizado para todas as classes que o modelo irá detetar, neste caso, neutro e agressivo.

5.1.2 Divisão do conjunto de dados

A abordagem recomendada para a avaliação do modelo é separar os dados de entrada aleatoriamente nos diferentes conjuntos:

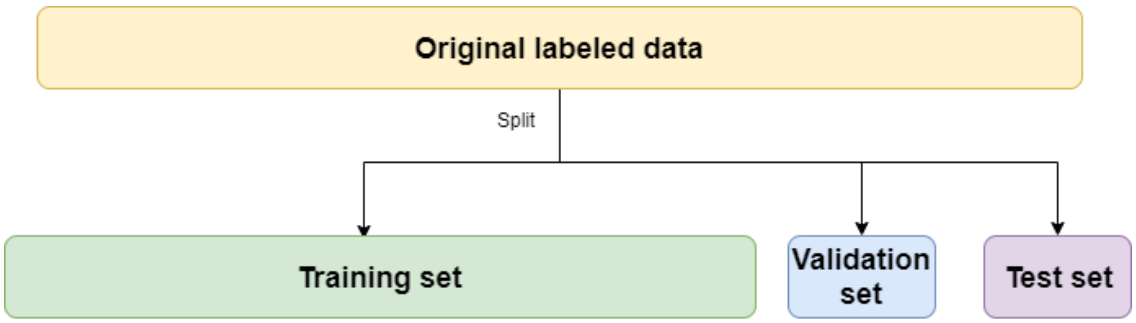


Figura 24 - Visualização da divisão

- Conjunto de dados de treino: É o conjunto de dados usado para treinar e fazer o modelo aprender as características/padrões dos dados.
- Conjunto de dados de validação: É o conjunto de dados que é utilizado para avaliar a performance do modelo durante o seu treino. O processo de validação fornece informações que ajuda a ajustar os hiperparâmetros e as configurações do modelo.

- Conjunto de dados de teste: É um conjunto separado de dados usado para testar o modelo após a conclusão do treino. Fornece uma métrica de desempenho do modelo final imparcial em relação à precisão, exatidão, entre outros.

5.1.3 Modelo de Expressões faciais

5.1.3.1 Construção do modelo

Na imagem seguinte está representada a arquitetura do modelo utilizado para a deteção de expressões faciais. Esta arquitetura segue uma estrutura típica de CNN para tarefas de deteção de expressões faciais. Começa com 4 *convolutional layers* que aprendem a detetar padrões simples, como bordas e texturas, e à medida que a rede se aprofunda, capturam padrões mais complexos e abstratos associados às expressões faciais. As camadas de *MaxPooling* e *Dropout* dentro destas *layers* ajudam a reduzir a dimensionalidade das representações e a prevenção de *overfitting*, tornando a rede mais eficiente e geral. As *dense layers* finais interpretam estas características para fazer previsões sobre as expressões faciais. Para além disso, utilizam funções de ativação, como neste caso a 'relu', para introduzir não linearidade, sendo importante para aprendizagens mais complexas. Por fim, a função de ativação *softmax* é usada para converter as saídas em probabilidades, indicando a probabilidade de cada expressão facial.

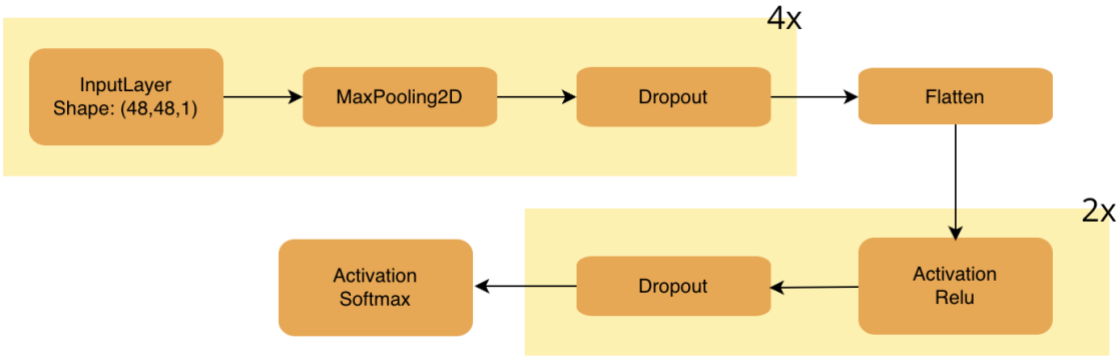


Figura 25 - Modelo de Classificação de Expressões Faciais

5.1.3.2 Compilação do modelo

É necessário compilar o modelo escolhendo uma *loss function*, um otimizador e as métricas de avaliação.

```
    metrics = ['accuracy', f1_m,precision_m, recall_m]
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=metrics)
```

Código 3 – Compilação do modelo

Neste modelo, foram utilizados como otimizador o "*adam*" para ajustar os pesos durante o processo de treino, como *loss function* o *"categorical_crossentropy"* devido ao modelo possuir várias classes, e como métricas a "*accuracy*", "*f1_score*", "*precision*" e "*recall*".

5.1.3.3 Treino do modelo

A função *train_model*, representada no código seguinte, é responsável por conduzir o processo de treino. Recebe como entrada o modelo a ser treinado e os conjuntos de dados de treino e teste. Durante o treino, a função ajusta os parâmetros da rede para otimizar o desempenho em relação à função de perda escolhida (neste caso, a *categorical_crossentropy*). Isto é executado pelas *ephocs*, onde em cada *epoch*, os dados de treino são divididos em lotes (*batch*) para atualizações iterativas dos pesos. Além disso, a função monitora métricas como *accuracy, F1-score*, *precision* e *recall* nos conjunto de dados de validação (conjunto de teste), permitindo uma avaliação contínua do desempenho do modelo ao longo do processo de treino. Por fim, o resultado é um objeto *history* que contém informações detalhadas sobre as métricas registadas em cada *epoch*.

```
def train_model(model, x_train, y_train, x_test, y_test):
      # train the model
      history = model.fit(x=x_train, y=y_train, batch_size=129,
      epochs=100, validation_data=(x_test, y_test))
      return history
```

Código 4 – Função de treino do modelo

5.1.3.4 Previsão com novos dados

O modelo foi exportado e utilizado para prever dados não vistos. Neste caso, os dados não vistos são o conjunto de dados de teste.

```
def test_images(model, test, x_test, le):

      image_index = random.randint(0, len(test))
      print("Original Output:", test['label'][image_index])
      pred = model.predict(x_test[image_index].reshape(1, 48, 48, 1))
      prediction_label = le.inverse_transform([pred.argmax()])[0]
      print("Predicted Output: ", prediction_label)
      plt.imshow(x_test[image_index].reshape(48,48), cmap= 'gray')
      plt.show()
```

Código 5 – Previsão do modelo

5.1.4 Deteção de Objetos

5.1.4.1 Arquitetura do modelo

A arquitetura escolhida para o desenvolvimento deste modelo foi o YOLOv8 (you only look once). A escolha do YOLOv8 para como modelo de deteção de objetos é alimentada pela procura de uma abordagem moderna e eficaz para resolver estas tarefas, com foco no desempenho em tempo real e precisão.

O YOLOv8 utiliza uma *convolutional neural network* que pode ser dividida em duas partes principais: a *backbone* e a *head*. Uma versão modificada da arquitetura *CSPDarknet53* forma o *backbone*. Esta arquitetura consiste em 53 *convolutional layers* e emprega conexões parciais para melhorar o fluxo de informações entre as diferentes camadas. Isto implica que a rede possua conexões parciais que passem informações não apenas dentro do mesmo estágio, mas também entre diferentes estágios da rede. Estas conexões parciais têm o objetivo de melhorar o fluxo de informações entre as *layers*, permitindo que as características extraídas em estágios anteriores sejam mais eficazmente utilizadas por *layers* posteriores, podendo isto ajudar a capturar informações de diferentes níveis de abstração e melhorar a representação das características nas várias etapas da rede. A *head* do YOLOv8 consiste em múltiplas *convolutional layers* seguidas por uma série de camadas totalmente conectadas. Estas camadas são responsáveis por prever caixas limitadoras, pontuações de objetividade e probabilidades da classe para os objetos detetados na imagem.

5.1.4.2 Treino do modelo

Com a junção dos *datasets* dos objetos, foi essencial realizar alterações e unir o ficheiro de configuração *data.yaml* antes de iniciar o processo de treino. Este ficheiro fornece informações importantes sobre o conjunto de dados que será usado para treinar o modelo, contendo os caminhos para os dados de treino e teste, o número e nome de classes presentes no conjunto de dados, o tamanho das imagens, entre outros parâmetros importantes.

```yaml
names:
- Linggis
- cap
- glasses
- hammer
- hat
- hood
- scissors
- sunglasses
- wrench
nc: 9
roboflow:
  license: CC BY 4.0
  project: thief-detection-dataset
  url: https://universe.roboflow.com/dataset-veqxx/thief-detection-dataset/dataset/1
  version: 1
  workspace: dataset-veqxx
test: test/images
train: train/images
val: valid/images
```

Figura 26 - Ficheiro data.yaml

Após a configuração do arquivo "data.yaml", o modelo está pronto para iniciar o processo de treino com a linha de comando representada na figura seguinte:

```
!yolo task=detect mode=train model=yolov8m.pt data={dataset.location}/data.yaml epochs=100 imgsz=640
```

Figura 27 - Linha de comando que inicia o processo de treino

O parâmetro *"mode=train"* indica que o modelo está a ser configurado para o processo de treino. O *"model=yolov8m.pt"* referencia um ficheiro chamado "yolov8m.pt", que contém um modelo pré-treinado. Este modelo é disponibilizado pelos programadores da ferramenta *YOLO* e serve como ponto de partida ao treino do modelo. Como referenciado em cima, o modelo necessita de estar apontado para um ficheiro de configuração *.yaml,* que contém informação necessária sobre o conjunto de dados a usar. Após isso é definido o número de *epochs* e o tamanho das imagens que serão utilizadas.

5.1.4.3 Previsão com novos dados

O modelo foi exportado no passo anterior, sendo utilizado para prever dados não vistos no código a seguir representado. Foi necessário alterar o mode para *"predict",* ajustar o caminho para o modelo treinado e utilizar o conjunto de imagens de teste para esta previsão.

```
!yolo task=detect mode=predict model=weights/best.pt conf=0.25 source={dataset.location}/test/images
```

Figura 28 - Linha de comando que inicia a previsão do modelo

5.1.5 Deteção de ações agressivas

5.1.5.1 Construção do modelo

Ao considerar o modelo a ser construído, foi crucial reconhecer que a deteção de ações requer a consideração da sequencialidade dos dados, isto é possível com a utilização de camadas *LSTM*. As *LSTMs* são um tipo de arquitetura de *recurrent neural networks (RNN)* que são conhecidas por lidar bem com a sequências de dados, como séries temporais ou dados com dependências temporais. A construção do modelo com camadas *LSTM (Long Short-Term Memory)* foi motivada por essa capacidade de lidar eficazmente com padrões de dados sequenciais. Estas camadas são capazes de capturar relações complexas ao longo do tempo, permitindo que a *neural network* aprenda a partir de padrões temporais presentes nos dados.

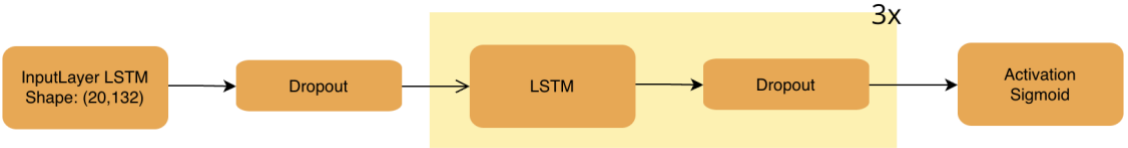


Figura 29 - Modelo de Classificação de Ações Agressivas

Este modelo inicia com várias camadas LSTM sequenciais, o modelo aprende progressivamente a reconhece padrões temporais, começando por características simples e avançando para padrões complexos relacionados à ação em análise. As camadas de *Dropout* inseridas entre estas camadas evitam o *overfitting*. A camada final *dense*, com ativação sigmoidal, interpreta as características aprendidas e gera previsões binárias. Estas previsões podem ter como resultado 0, no caso de ação neutra ou 1, no caso de ação agressiva.

5.1.5.2 Compilação do modelo

Como nos anteriores, o modelo de deteção de ações agressivas necessita de ser compilado antes de partir para o processo de treino. Foi empregue o mesmo otimizador e métricas de avaliação, visando manter a consistência nas métricas de desempenho durante o treino de todos os modelos. No entanto, a escolha da função de perda foi tomada de acordo com o tipo de classificação final que era esperada pelo modelo, a classificação binária. Nesse sentido, a função de perda "*binary_crossentropy*" foi escolhida para avaliar a diferença entre as previsões da rede e os rótulos reais dos dados.

```
metrics = ['accuracy', f1_m,precision_m, recall_m]
model.compile(optimizer="adam",metrics=metrics, loss="binary_crossentropy")
```

Código 6 – Compilação do modelo

5.1.5.3 Treino do modelo

Após o passo anterior, o modelo é treinado segundo o código representado abaixo, o qual orienta o modelo a 100 *ephocs* de treino com um tamanho do *batch* de 32 exemplos por cada *epoch*. Por fim, a validação é realizada usando os dados de teste fornecidos por *X_test* e *y_test*.

```
model.fit(X_train, y_train, epochs=100, batch_size=32,
validation_data=(X_test, y_test))
```

Código 7 – Treino do modelo

5.2 Construção da aplicação

Após a elaboração e implementação dos modelos especializados na classificação de comportamentos suspeitos, foi desenvolvida uma aplicação de software integradora. Esta aplicação tem a capacidade de unificar, analisar e consolidar as informações geradas por estes modelos. Estes resultados são submetidos a um sistema de análise que lhes atribui percentagens, resultando na identificação de um indivíduo como suspeito com base na soma das características representadas na imagem.

5.2.1 Deteção de movimento

Antes da ativação dos modelos, um loop de pré-processamento é implementado, responsável por constantemente verificar a presença de movimento na imagem. Este sistema de deteção de movimento analisa os *frames* de vídeo provenientes da câmera e divide cada

frame em quadrados calculando a diferença média entre os valores dos pixéis de cada quadrado em comparação com o quadrado anterior. Se a diferença for considerável (maior que 80%), o sistema considera esse quadrado como movimento. Quando um número significativo de quadrados presentes no *frame* mostrarem movimento (cerca de 20%), chama a função face_crop para recortar as faces detetadas na imagem, o modelo deteção de objetos e a deteção de ações agressivas.

A mudança no número significativo de quadrados presentes no *frame* é uma forma de ajustar a sensibilidade do sistema de deteção de movimento. Quanto maior o número mínimo de quadrados que são precisos para detetar movimento, o sistema ficará menos sensível a pequenas variações de luz e sombra, tornando-o mais robusto contra falsos movimentos positivos.

No entanto, ao realizar esta alteração, é importante encontrar um equilíbrio para garantir que o sistema ainda seja capaz de detetar movimentos relevantes sem ignorar movimentos importantes devido a um limiar muito alto. Portanto, o ajuste preciso do número significativo de quadrados foi considerado tomando em conta possíveis ambientes com elevadas variações de luz em que este sistema se poderá encontrar.

Na figura seguinte encontra-se uma *frame* onde existe uma movimentação da cadeira identificada pelos pontos verdes aplicados pelo sistema. Além dos pontos justificados pelo movimento, existem também outros devido às constantes mudanças de luz e sombras presentes na cena.

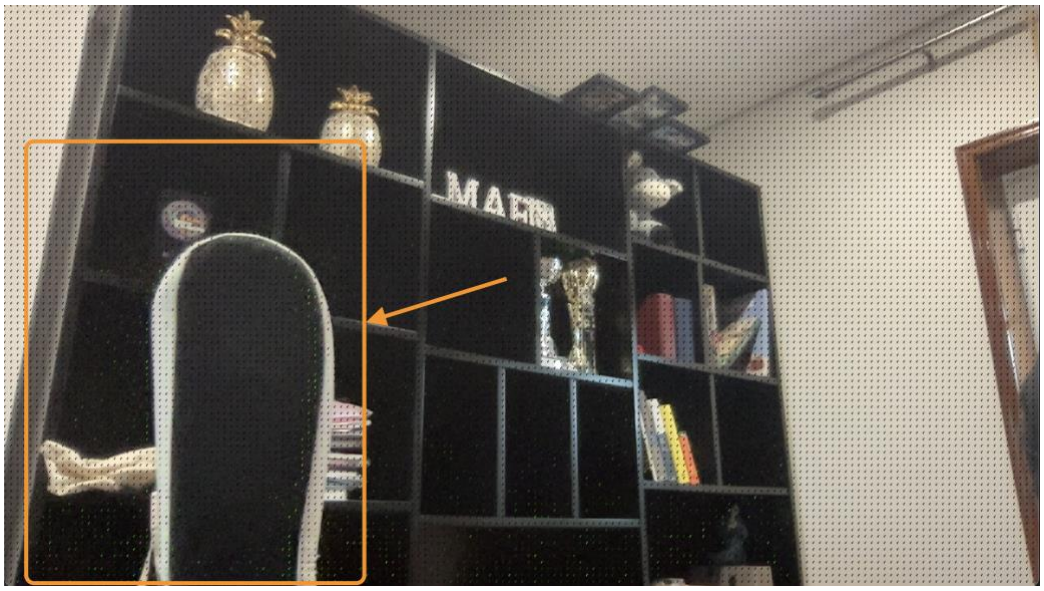


Figura 30 - Representação do movimento do *frame*

5.2.2 Deteção de faces

A deteção de faces é responsável por identificar e localizar os rostos dentro dos *frames* de vídeo capturados pela câmera. Para realizar esta tarefa, o sistema utiliza o método *Haar Cascade*, que é um algoritmo de deteção de objetos utilizado frequentemente em *computer vision*.

Primeiramente, o sistema converte o *frame* de vídeo para a escala de cinza, este passo é importante para simplificar o processamento e melhorar a precisão da deteção. Em seguida, aplica o classificador *Haar Cascade* treinado especificamente para a deteção de faces. Este classificador é treinado previamente com um conjunto de características que o algoritmo procura no *frame* para identificar padrões correspondentes a faces humanas. Quando a face é detetada, o sistema desenha um retângulo, destacando a sua posição no *frame*. Além disso, o sistema recorta a região da face e aplica posteriormente a análise da expressão facial dessa mesma face.

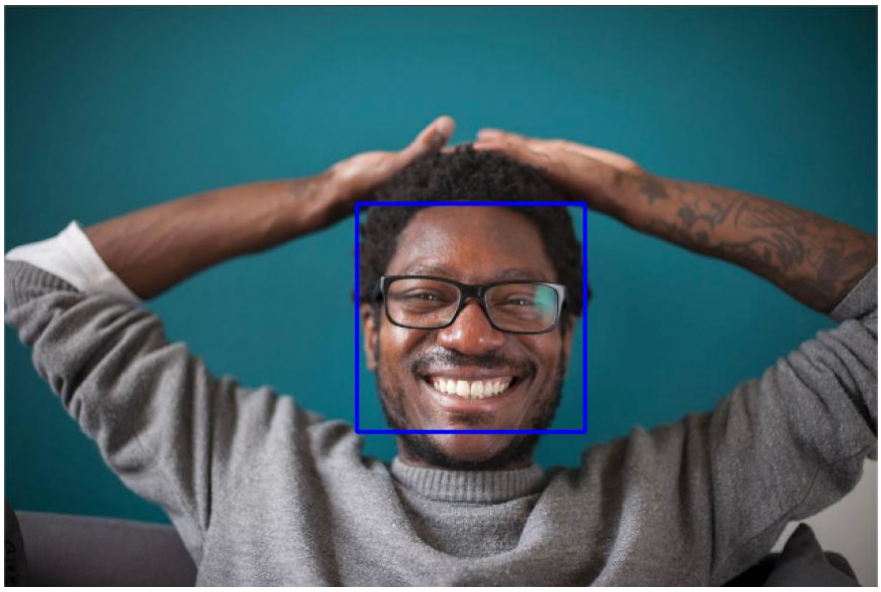


Figura 31-Deteção facial

5.2.3 Deteção da expressão facial

A deteção é realizada pelo modelo mencionado no ponto 5.1.3, sendo que, à medida que cada face é processada, o sistema avalia e identifica a emoção predominante na expressão facial da pessoa. No momento da previsão, o modelo associa uma das sete expressões listadas na tabela a seguir, com base nas características faciais identificadas na imagem analisada.

Tabela 5 - Características das expressões

Expressão facial	**Características**
Angry (Raiva)	Sobrancelhas franzidas, olhos estreitos e uma boca tensa.
Disgust (Repugnância)	Nariz franzido e lábios torcidos
Fear (Medo)	Olhos arregalados e lábios apertados
Happy (Felicidade)	Sorriso, olhos brilhantes e bochechas levantadas.
Neutral (Neutralidade)	Face relaxada e ausência de características expressivas
Sad (Tristeza)	Sobrancelhas franzidas, cantos da boca voltados para baixo e olhos tristes.
Surprise (Surpresa)	Sobrancelhas levantadas, olhos arregalados e uma boca ligeiramente aberta.

Uma vez identificada a emoção predominante, o sistema complementa a análise visual exibindo a emoção em tempo real nos *frames* de vídeo, como é representado na figura seguinte.

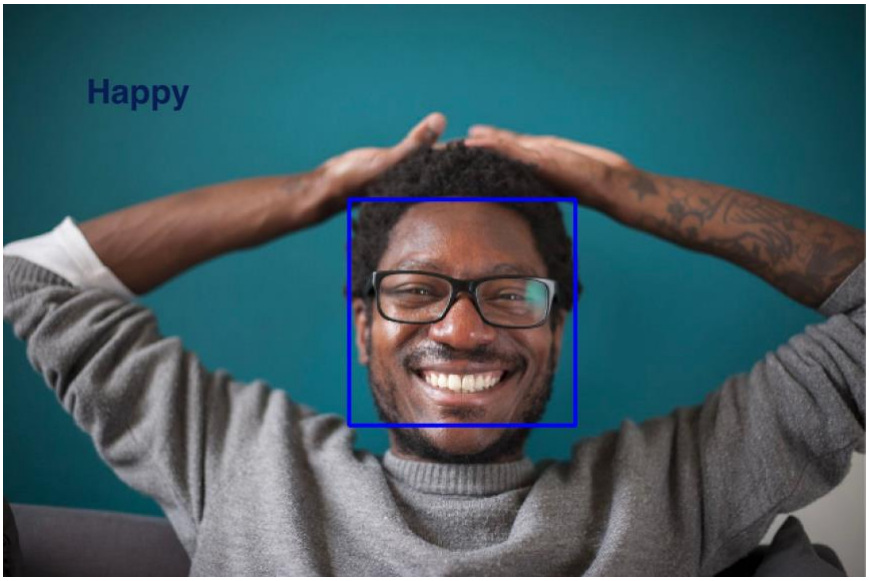


Figura 32 - Classificação da expressão facial

5.2.4 Deteção de objetos

Conforme mencionado no subcapítulo 5.2.1, após a deteção de movimento, o sistema realiza uma identificação dos objetos relevantes em cena. O sistema processa cada *frame* da

câmara usando o modelo YOLO para detetar objetos, desenha os retângulos delimitadores para identificar os objetos detetados, sobrepondo as informações dos objetos no *frame* exibidos pela janela da câmara. Além disso, exibe também a confiança da deteção e o nome da classe dos objetos na consola.

Nos modelos YOLO, é possível definir limites de confiança. Estes limites atuam como critério para determinar quais deteções devem ser consideradas válidas e quais devem ser descartadas. Em termos simples, o modelo avalia a probabilidade de cada deteção e compara essa probabilidade com o limite pré-definido. As deteções com probabilidade superior a esse limite são consideradas fiáveis e, consequentemente, mantidas para análise e apresentação, enquanto as deteções com probabilidade inferior são descartadas. Isto é particularmente útil para a precisão das deteções, uma vez que permite controlar o nível de confiança dos resultados fornecidos.

A figura abaixo apresentada, representa algumas imagens que foram utilizadas como validação da funcionalidade do modelo e a sua respetiva classificação.

Figura 33 – Classificação de objetos

5.2.5 Deteção de ações agressivas

No contexto de um vandalismo, as "ações agressivas" referem-se a comportamentos violentos que um ladrão pode adotar ao tentar cometer o delito. Mais precisamente, neste projeto, as ações agressivas incluem o uso de força física, como socos, ou movimentos bruscos em direção à câmara instalada, sendo consideradas como indicadores de possíveis ameaças.

Assim como os restantes classificadores do sistema, este começa a sua execução quando uma deteção de movimento é registada pela câmara. O processo começa com a aquisição de *frames* da câmara, onde cada *frame* é convertido para o espaço de cores RGB para que possa ser utilizado pelo *MediaPipe*, que espera imagens nesse formato. Esta biblioteca identifica as posições das partes do corpo em cada *frame*, que são posteriormente guardadas através de coordenadas e transformadas numa sequência. Quando a sequência atinge um comprimento de 20, é iniciada uma nova *thread* para a execução do modelo que que classifica a ação. Com base na classificação feita, o código desenha um retângulo na imagem do vídeo. Se a ação for "*neutral*", o retângulo será desenhado a verde, se for "*violence*", o retângulo é vermelho e mais espesso. Além disso, as *landmarks* da pose são desenhadas na imagem como estão representadas na imagem seguinte.

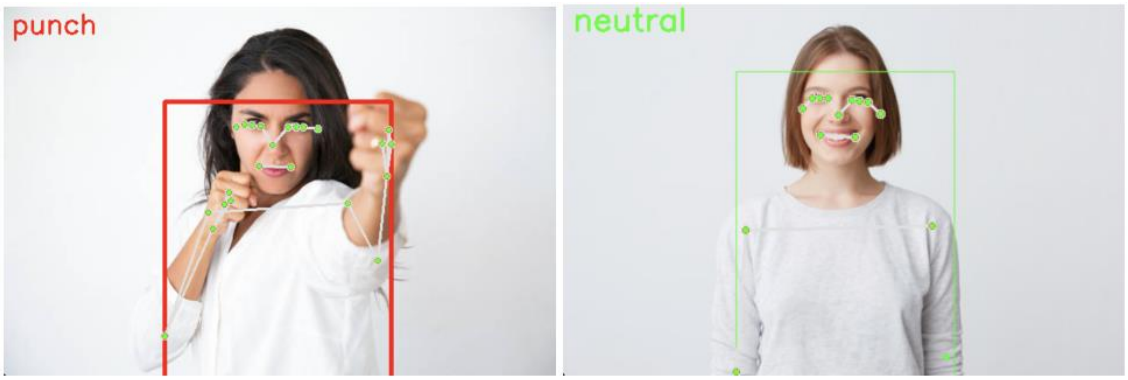


Figura 34 - Classificação de ações

5.2.6 Avaliação final do indivíduo

A integração dos três modelos mencionados para a identificação de suspeitos neste sistema envolveu a consolidação das previsões dos modelos, com o objetivo de gerar uma única probabilidade final que indicasse se a pessoa representada na cena poderia ser considerada uma potencial suspeita ou não. Para isso, foi necessário recolher informações sobre as classes identificadas pelos modelos individuais de deteção e atribuir a cada uma delas um peso. Estes pesos encontram-se distribuídos em cinco níveis de importância:

- **Crítico** (peso=1) - Classes identificadas como "Críticas" são aquelas que têm o maior impacto na decisão final. Isto significa que se uma característica crítica for detetada, aumentará substancialmente a probabilidade do modelo.
- **Alto** (peso=0.8) - Classes identificadas como "Altas" têm importância significativa, mas não tão crítica quanto as "Críticas". Influenciam a probabilidade final, mas em menor grau do que as classes críticas.
- **Médio** (peso=0.5) - As classes identificadas como "Médias" têm uma importância moderada. Podem contribuir para a probabilidade final, mas não de forma tão substancial quanto as classes "Críticas" ou "Altas".
- **Ligeiro** (peso=0.3) - As classes identificadas como "Ligeiras" têm uma influência menor na probabilidade final. Representam características que são de interesse, mas que têm um impacto limitado na decisão final.
- **Nulo** (peso=0) - Classes identificadas como "Nulas" não têm peso na probabilidade final, o que significa que sua presença ou ausência não afeta a decisão. Isto é usado para descartar certas características que não são relevantes para a deteção de ladrões.

Como referido acima, estes níveis e pesos foram distribuídos por todas as classes tendo em consideração a sua proximidade com as características dos potenciais suspeitos. A seguinte figura demonstra essa mesma distribuição.

Modelo	Classe	Nível	Peso
Expressões faciais	Happy	Nulo	0
	Neutral	Ligeiro	0.3
	Surprise	Médio	0.5
	Disgust		
	Sad	Alto	0.8
	Fear	Crítico	1
	Angry		
Objetos	Hat	Ligeiro	0.3
	Cap		
	Sunglasses		
	Hood	Médio	0.5
	Scissors	Alto	0.8
	Wrench		
	Hammer	Crítico	1
	Crowbar		
Ações	Não Agressivo	Nulo	0
	Agressivo	Crítico	1

Figura 35 - Níveis de importância para potenciais classes identificadas

Com os níveis distribuídos pelas classes, é possível realizar o cálculo da probabilidade de o modelo identificar características suspeita. Este cálculo é determinado pelo somatório das probabilidades de todas as classes identificadas multiplicadas pelo peso das mesmas. O somatório é realizado apenas quando o modelo é capaz de identificar mais do que uma característica simultaneamente, sendo apenas aplicado ao modelo de reconhecimento de objetos. A equação deste cálculo é representada da seguinte forma:

$$Pr_m = \sum(Pr_c * P_c) \quad (13)$$

Onde Pr_m representa a probabilidade de um modelo identificar características suspeitas, Pr_c a probabilidade de classe e P_c o peso associado à classe identificada pelo modelo. Para classes que apenas identifiquem uma única classe o cálculo é dado por:

$$Pr_m = Pr_c * P_c \quad (14)$$

A probabilidade global de uma pessoa ser considerada um potencial criminoso é calculada considerando as probabilidades do modelo possuir características suspeitas multiplicando pela sua importância relativa ao resultado final. A equação é representada por:

$$Pr_f = \sum(Pr_m * P_m) \quad (15)$$

Onde Pr_f representa a probabilidade global da pessoa ser um potencial criminoso, Pr_m representa a probabilidade de um modelo identificar características suspeitas e P_m a importância do modelo relativamente ao resultado final.

As importâncias relativas atribuídas a cada modelo foram ajustadas consoante a sua criticidade e adaptadas consoante os resultados. Por fim, as importâncias atribuídas a cada modelo foram as seguintes:

- Reconhecimento de Expressões Faciais: 10% de importância (0.1)
- Identificação de Objetos: 50% de importância (0.5)
- Reconhecimento de Ações Agressivas: 40% de importância (0.4)

Substituindo os valores na equação acima temos:

$$Pr_f = (Pr_{me} * 0.1) + (Pr_{mo} * 0.5) + (Pr_{ma} * 0.4) \quad (16)$$

Onde Pr_{me}, Pr_{mo}, Pr_{ma} representam as probabilidade de o modelo de expressões faciais, de objetos e de ações agressivas, respetivamente, identificarem características suspeitas.

5.2.7 Ações

Após o modelo de deteção identificar um potencial suspeito com uma probabilidade superior a 50%, toma medidas proativas para garantir a documentação do incidente. Isto envolve a ativação automática da captura de trechos de vídeo a partir do momento em que o comportamento suspeito é identificado. Esses segmentos de vídeo são armazenados no *Google Cloud Storage*, assegurando não apenas a preservação segura dos dados, mas também a acessibilidade para futuras análises. No código desenvolvido, foi construída uma função chamada *upload_video_to_gcs* que recebe como entrada o caminho local do arquivo do vídeo já cortado (com um tamanho igual ao tempo de deteção de movimento a partir do momento que o modelo constata um suspeito), o nome do *bucket* no *GCS* e o nome desejado para o arquivo no *GCS*. O código inicializa o cliente *GCS*, obtém o *bucket* de destino, define um objeto *blob* com o nome e, de seguida, faz o upload do vídeo para o *GCS* por meio desse *blob.*

Além disso, o sistema é configurado para enviar um email contendo a informação de um possível suspeito para o proprietário do sistema, proporcionando uma notificação imediata e permitindo uma resposta eficaz a qualquer incidente suspeito detetado. O processo começa com a integração da biblioteca *smtplib* no *Python* que fornece acesso à funcionalidade de e-mail. Em seguida, um cliente de *e-mail* foi configurado, permitindo o fornecimento de informações como o servidor *SMTP* e as credenciais de autenticação.

Por fim, para tornar a comunicação mais dinâmica e personalizada, o sistema permite a incorporação de variáveis como a percentagem que foi identificada pelo modelo. O email segue o padrão encontrado na figura seguinte.

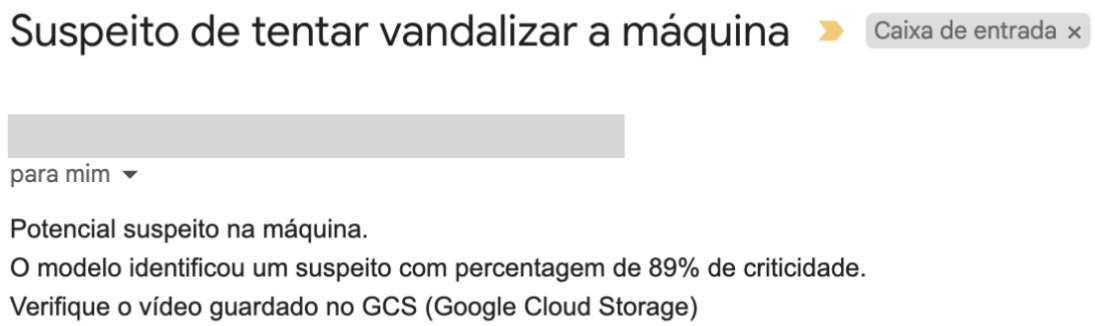


Figura 36 - Modelo de email

6 Experiências e Resultados

Neste capítulo são apresentados os resultados obtidos pelos modelos criados para este projeto, uma comparação entre estes com outros já existentes e uma apresentação dos resultados do sistema final.

6.1 Experiência e análise dos modelos

No treino de modelos de *machine learning*, tanto os conjuntos de dados como os próprios modelos necessitam de diferentes configurações de hiperparâmetros. Estes hiperparâmetros são parâmetros ajustáveis que não são aprendidos pelo modelo, mas sim definidos antes do treino com o objetivo de controlar e otimizar o comportamento do modelo durante o esse processo e, posteriormente, a sua capacidade de fazer previsões precisas em dados novos e não vistos. A forma de os determinar passa por uma série de experiências nas quais são comparadas as arquiteturas implementadas numa primeira fase deste projeto, como é indicado no capítulo da implementação, com outras variantes disponíveis. Para além disso, foram selecionados e experienciados conjuntos de hiperparâmetros aplicados ao modelo, verificando posteriormente a evolução da função *loss* ao longo do processo de treino. Os conjuntos de hiperparâmetros usados nesta fase de experimentação são:

- ***Learning rate*** - A *learning rate* é um parâmetro que permite definir a velocidade com que uma CNN muda os seus pesos. Para altos valores de *learning rate*, os pesos da neural network variam constantemente. No entanto, modelos com foco nos últimos padrões identificados têm o learning rate mais baixo, e os pesos variam lentamente. Determinar se o modelo possui uma *learning rate* adequada é possível através da curva da função *loss* durante o processo de treino: uma rápida redução seguida de estabilidade indica um bom ajuste, enquanto convergência lenta ou oscilações podem sinalizar que a taxa está inadequada.

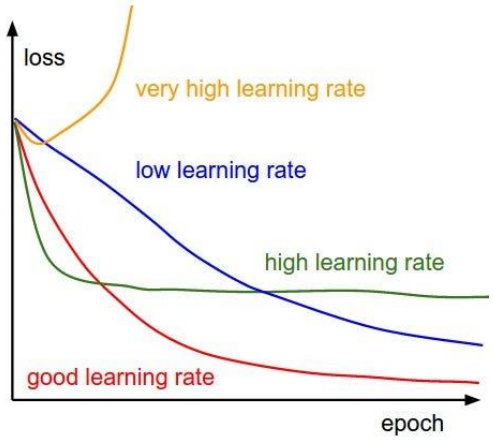


Figura 37 - Impacto da learning rate na função loss

- ***Batch*** - O tamanho do *batch* é um parâmetro ligado ao número de *ephocs*, pois define o número de elementos que o conjunto de dados usa em cada ciclo de iteração de *ephoc*. Uma *ephoc* representa uma iteração de dados composto por vários ciclos, onde cada ciclo possui um tamanho correspondente ao tamanho do *batch*.

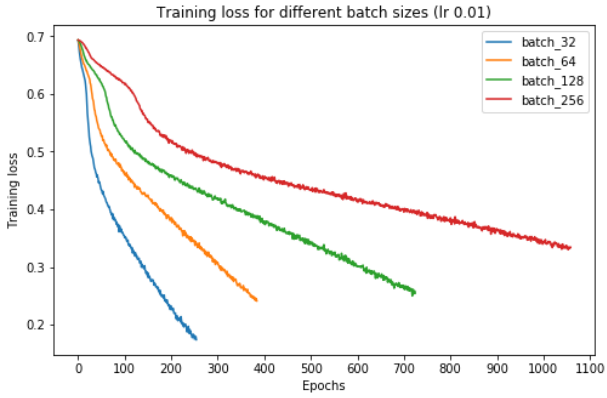


Figura 38 - Impacto do tamanho do *batch* na função *loss*

- **Epochs** - O número de epochs significa o número de vezes que a *neural network* itera durante o treino, isto é, o número de vezes que os dados iteram pela rede para provocar o ajuste dos pesos. O número de *epochs* impacta o tempo de treino quando um sistema possui uma *learning rate* adequada, tornando o modelo mais preciso. Geralmente, variações entre epochs iniciais impactam mais do que as finais devido ao conhecimento adquirido pelas *epochs* anteriores.

6.1.1 Modelo de expressões faciais

O estudo do reconhecimento de expressões faciais empregou o *dataset FER-2013* criado por (Sambare, 2020), composto por milhares de imagens faciais, onde cada imagem é identificada com uma das várias expressões faciais, tais como felicidade, tristeza, raiva e surpresa. O conjunto de dados foi dividido de forma aleatória em conjuntos de treino e teste,

seguindo uma proporção de 75% de dados de treino (28710 imagens) e 25% de teste (7179 imagens). Na figura seguinte estão representados a divisão das imagens pelas respetivas classes, o gráfico da esquerda representativo do conjunto de dados de treino e o da direita de teste.

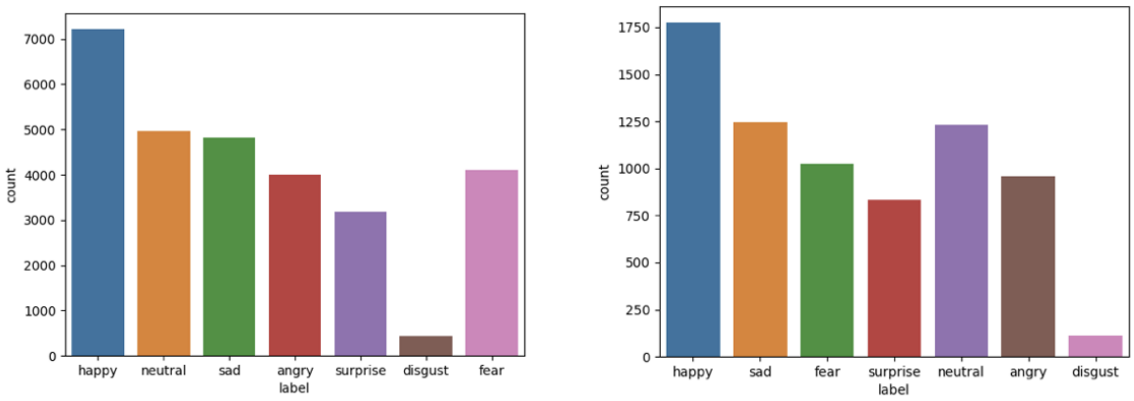


Figura 39 - Distribuição de imagens ao longo do *dataset* de expressões faciais

Para estabelecer um ponto de partida sólido, foram conduzidas experiências iniciais definindo um conjunto de hiperparâmetros específicos como valores de referência. Esses valores de referência incluíram um *learning rate* de 0.0001, um *batch size* de 129 e 100 ephocs de treino. Estes valores foram escolhidos como ponto de partida tendo em conta uma abordagem equilibrada entre eficiência computacional e a busca pela convergência eficaz do modelo. Na figura seguinte, encontra-se representado o gráfico da *loss*, de acordo com estes hiperparâmetros.

Learning Rate	*Batch Size*	*Epochs*
0,0001	129	100

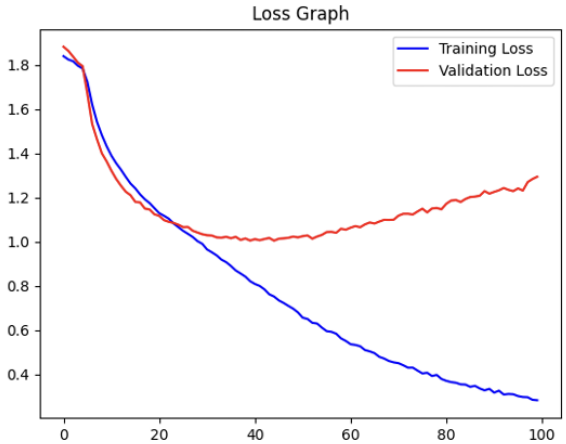


Figura 40 - Resultado da função loss para learning rate de 0,0001

Na curva de treino, identifica-se uma linha diagonal a azul, que sugere que o *learning rate* foi definido com um valor muito baixo. Isto resultou numa convergência lenta do modelo, onde o progresso na redução da *loss* era gradual. Na curva de validação representada a vermelho, existiu uma queda acentuada na *loss*, atingindo valores mínimos de aproximadamente 1,08, indicando que o modelo generalizou bem e fez progressos significativos na redução da discrepância entre as previsões e os dados de validação. No entanto, após a 30ª *ephoc* de treino, a curva de validação começou a estabilizar e, posteriormente, mostrou sinais de *overfitting*, significando que o modelo se ajustou excessivamente aos dados de treino, perdendo a capacidade de generalizar novos dados.

Com o intuito de otimizar ainda mais o desempenho do modelo e acelerar a sua convergência, foi decidido realizar uma nova experiência, aumentando o *learning rate* para 0,001. Esta alteração levou a uma melhoria no gráfico da *loss*, resultando numa curva mais suave, o que mostra uma convergência mais estável do modelo durante o treino. Além disso, notou-se que o *overfitting* ocorreu mais tarde, apenas a partir da 37ª *epoch*. O ponto mais destacado deste ajuste foi a obtenção de um valor mínimo da *loss* de aproximadamente 1,00, representando um desempenho ainda melhor do que o caracterizado anteriormente. O gráfico desta segunda experiência está representado na figura seguinte.

Learning Rate	*Batch Size*	*Epochs*
0,001	129	100

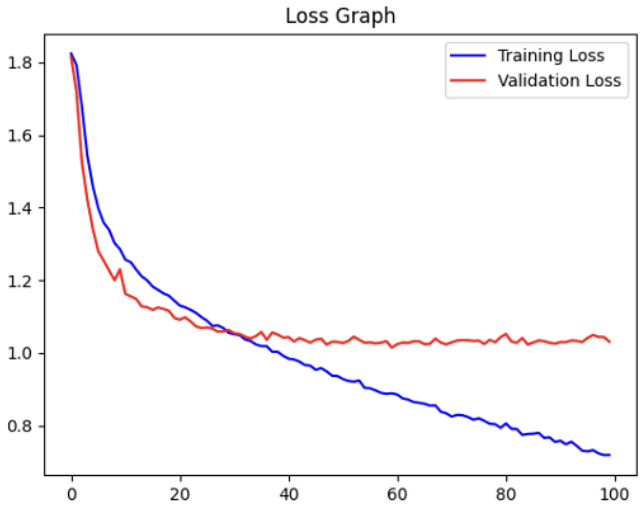


Figura 41 - Resultado da função loss para learning rate de 0,001

Para obter o melhor modelo, foi aplicada uma técnica chamada *early stopping,* que permitiu a seleção do modelo ideal com base na menor *loss* durante a fase de testes. Com isto, foi possível obter a *confusion matrix* e as métricas de avaliação. De acordo com a *confusion matrix* representada na Figura 42, o modelo CNN proposto em combinação mostra-se mais preciso em previsões de expressões felizes, neutras, tristes e surpresas e menos precisa nas expressões de repulsa.

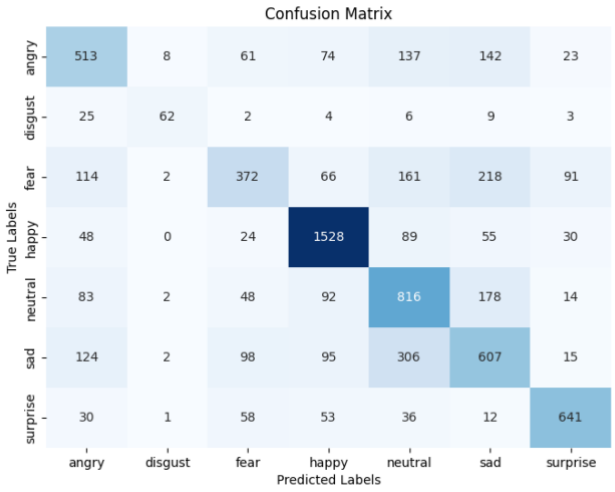


Figura 42 - Confusion matrix do modelo de expressões faciais

De acordo com os resultados do melhor modelo, a *loss* apresentou valores de aproximadamente 1,00, a *accuracy* de 0,64 e a *precision* 0,74.

Após a obtenção dos resultados, foi possível realizar uma análise comparativa da *accuracy* do modelo em relação a outros modelos de deteção de expressões faciais previamente desenvolvidos que utilizaram o conjunto de dados *FER-2013* para o seu próprio desenvolvimento.

Tabela 6 - *Accuracy* de trabalhos relacionados

Trabalhos relacionados	**Método Proposto**	***Accuracy***
(Quinn Minh-An et al., 2017)	Linear SVM	47.80%
(Talegaonkar et al., 2019)	CNN	60,12%
(Gan, 2018)	CNN - GoogleNet	63.91%

De autoria própria	**CNN**	**64.10%**
(Gan, 2018)	CNN – AlexNet	64.24%
(Quinn Minh-An et al., 2017)	CNN	66.50%

6.1.2 Modelo de deteção de objetos

Como referido anteriormente, o modelo desenvolvido utilizou um *dataset* que foi preparado através da junção de diversos conjuntos de dados do *Roboflow*. Este *dataset* é composto por milhares de imagens de 7 objetos diferentes: pé-de-cabra, boné, martelo, chapéu, capuz, tesoura e chave inglesa. Foram separados de forma aleatória em conjuntos de treino e teste, contendo no primeiro 3804 imagens (91,7%) e no segundo 315 imagens (8,3%). Na figura seguinte estão representados a divisão das imagens pelas respetivas classes, o gráfico da direita representa do conjunto de dados de treino e o da esquerda de teste.

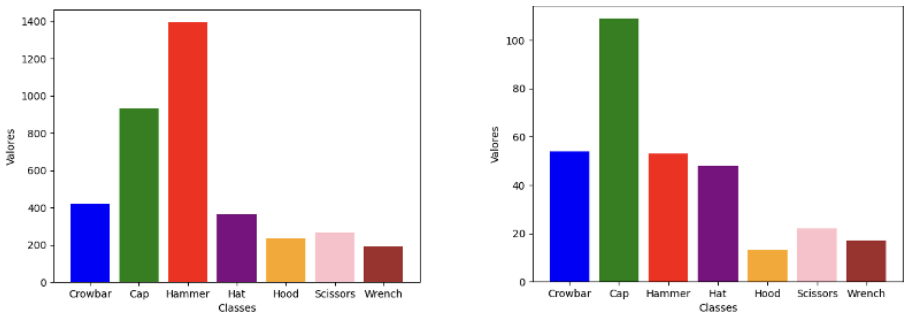


Figura 43 - Distribuição de imagens ao longo do *dataset* de objetos

A primeira decisão crucial a tomar ao usar o detetor YOLOv8 é a seleção do tamanho da rede a partir das diferentes variantes disponíveis. Os autores do YOLOv8 tornaram publicamente acessíveis cinco redes pré-treinadas que podem ser prontamente utilizadas numa aplicação específica, abrangendo desde a rede mais pequena até às redes mais extensas, avaliadas em termos do número de parâmetros: nano (YOLOv8n), pequena (YOLOv8s), média (YOLOv8m), grande (YOLOv8l) e extra grande (YOLOv8x). A escolha de uma rede mais compacta resulta numa performance final mais modesta, mas ao mesmo tempo consome menos memória no dispositivo e permite inferências mais rápidas. Por outro lado, uma rede mais robusta oferece um desempenho superior na deteção de objetos. As experiências realizadas consistiram em medir o tempo de processamento das redes mais compactas e com inferências mais rápidas, desde o momento da captura da imagem até à projeção das *bounding boxes* na visualização. A comparação foi conduzida com um tamanho de imagem padrão de 640x640 pixéis e os resultados estão apresentados na Figura 44.

Estes resultados sugerem que, para uma experiência do utilizador otimizada em cenários dinâmicos, a rede mais pequena YOLOv8n é capaz de cumprir com mais eficiência os requisitos de tempo real. Contudo, os restantes modelos podem ser mais apropriados apenas para cenários em que o desempenho em tempo real não seja crítico para os objetivos finais. Nestas situações, a deteção de objetos será capaz de posicionar mais eficazmente o objeto no contexto do utilizador, mas levará ligeiramente mais tempo para obter esses resultados.

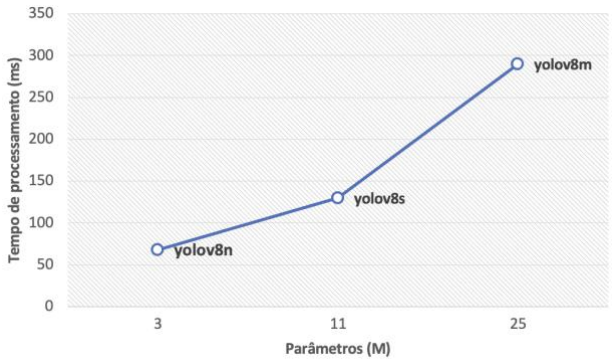


Figura 44 - Tempo de processamento em relação às diferentes variantes do YOLOv8

A utilização de modelos de rede com um número reduzido de parâmetros resulta numa performance inferior. A métrica comum para quantificar o desempenho é a média da precisão média (mAP). O mAP@A indica o desempenho quando existe pelo menos uma sobreposição de A% entre a caixa delimitadora da deteção e a caixa delimitadora real do objeto. Os desempenhos das diferentes variações do modelo de deteção YOLOv8 são apresentados na Figura seguinte para mAP@50 e mAP@50-95.

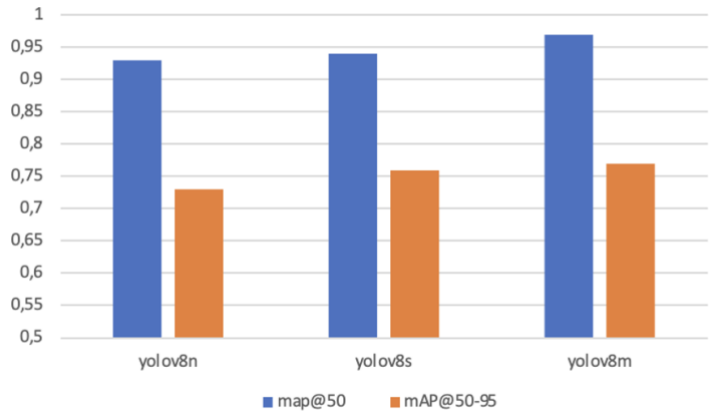


Figura 45 - Média de precisão média de acordo com as variantes do modelo YOLOv8

Os resultados obtidos sugerem que não se espera um impacto significativo no desempenho em relação à métrica *mAP* ao utilizar modelos com maior tamanho e, por tanto, a utilização de modelos mais compactos (*yolov8n*) é mais indicada neste estudo, onde a velocidade de resposta necessita ser mais rápida.

Neste contexto, a avaliação se concentra na exposição do desempenho do modelo YOLOv8n. É importante observar que, até ao momento, não existem modelos comparáveis para análise, o que limitou a realização de uma comparação direta. Em vez disso, este estudo concentra-se na demonstração de resultados. O modelo YOLOv8n foi treinado ao longo de 25, 50, 75 e 100 *ephocs*, e os resultados de *precision*, *recall* e *mAP* encontram-se detalhados na tabela e gráficos a seguir.

Tabela 7 - Métricas yolov8n

Nº de epochs	**Loss**	**Precision**	**Recall**	**mAP@50**	**mAP@50-95**
25	0,80	0,84	0,64	0,85	0,64
50	0,62	0,87	0,73	0,88	0,68
75	0,52	0,92	0,77	0,90	0,71
100	0,48	0,95	0,79	0,93	0,73

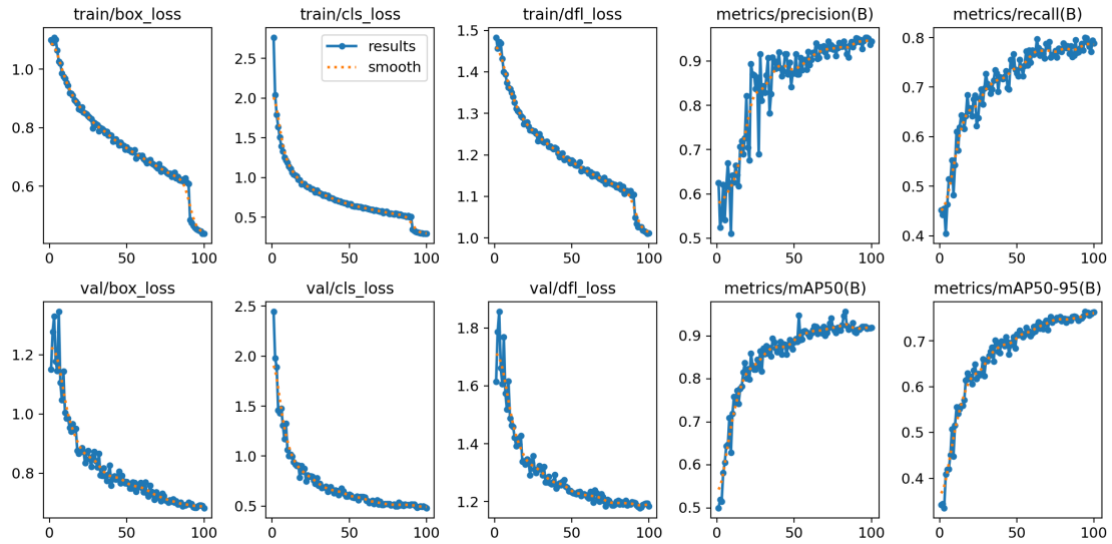


Figura 46 - Resultados do YOLOv8n para 100 epochs

A partir do gráfico, podemos observar que a *loss* do modelo diminui à medida que treinamos com um maior número de *epochs*. A *loss* do modelo deve ser mantida o mais baixa possível. Podemos também inferir a forma como a *precision*, o *recall* e o *mAP* do modelo se alteram, sendo importante que estes parâmetros sejam elevados para obter uma grande precisão. O modelo foi testado em tempo real utilizando o *OpenCV* e apresentou bons resultados, no entanto, identificou-se que o desempenho do modelo é afetado adversamente em cenários de baixa luminosidade. Em contrapartida, em condições de iluminação adequada,

o modelo demonstrou resultados promissores. A figura abaixo ilustra estas condições e exibe as previsões do modelo no conjunto de imagens de teste.

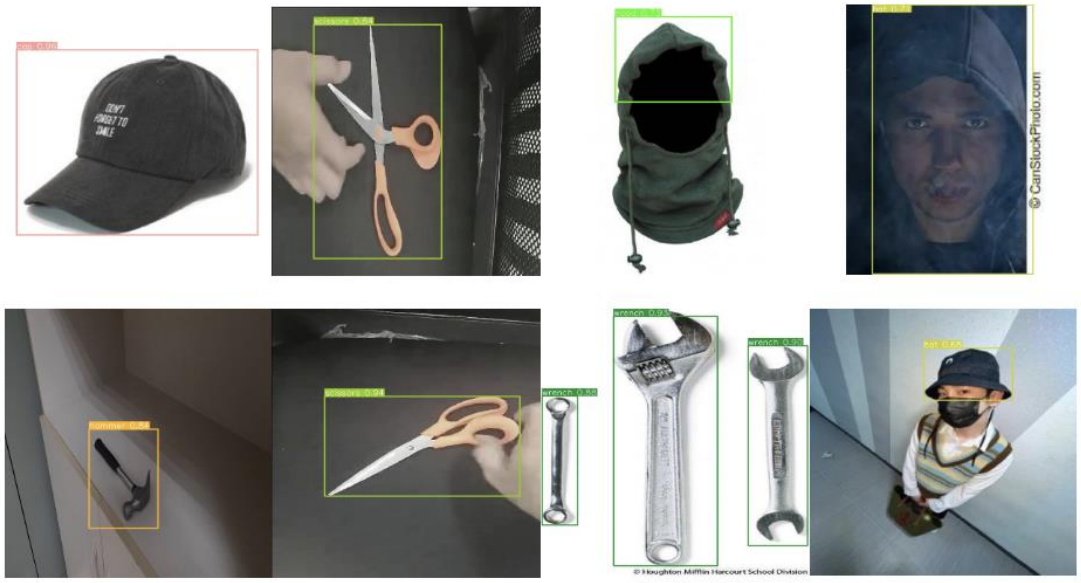


Figura 47 - Previsões do conjunto de teste do modelo YOLOv8n

6.1.3 Modelo de deteção de ações agressivas

Como já foi referido anteriormente no subcapítulo 5.1.1, este modelo emprega um conjunto de dados construído através de *landmarks* do corpo humano. O conjunto de dados em questão é cuidadosamente dividido, destinando 80% dos dados para o treino do modelo e alocando os 20% restantes para fins de teste.

O processo de treino com a arquitetura LSTM foi executado por 20 *ephocs*. O modelo alcançou a menor *loss* na 20ª iteração, com uma *accuracy* de 99,78%. Para além deste modelo, foram implementados mais dois, o *Simple RNN* e o *GRU*, que são arquiteturas de *RNN* que serviram de comparação com o modelo *LSTM*, sendo essa comparação realizada através das métricas fornecidas por todos os modelos. Com a arquitetura *Simple RNN* também foi realizado ao longo de 20 *ephocs*. O modelo alcançou a *loss* perda na 20ª *ephoc*, com uma *accuracy* de 99,25 %. No caso da arquitetura *GRU*, o processo de treino também ocorreu ao longo de 20 *epochs*, com a menor *loss* alcançada na 18ª iteração e uma *accuracy* de 99,46%.

Os gráficos da *accuracy* e da *loss* dos modelos LSTM*, Simple RNN* e *GRU* podem ser visualizados nas figuras seguintes.

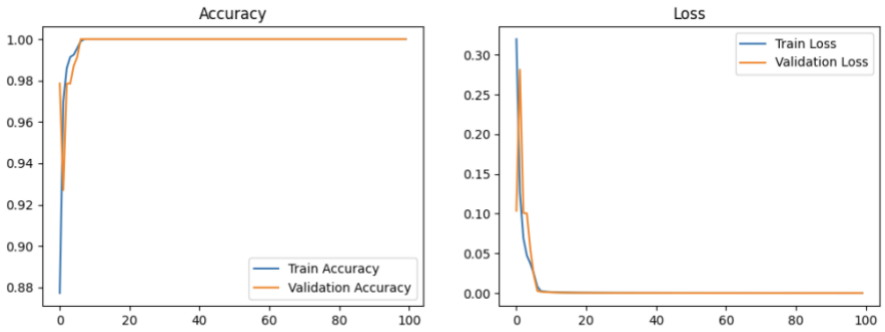


Figura 48 - Gráfico da accuracy e da loss da arquitetura LSTM

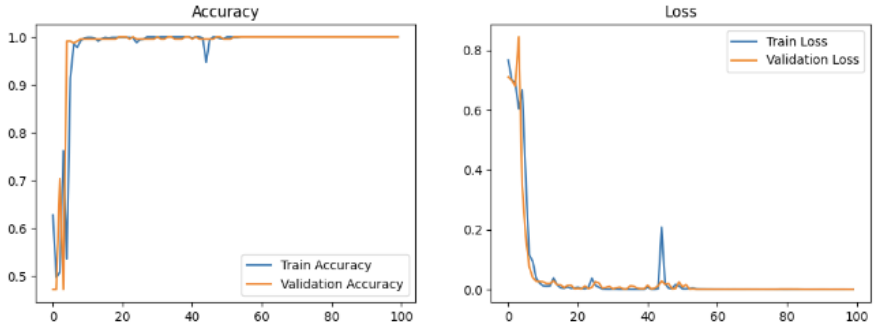


Figura 49 - Gráfico da accuracy e da loss da arquitetura *Simple RNN*

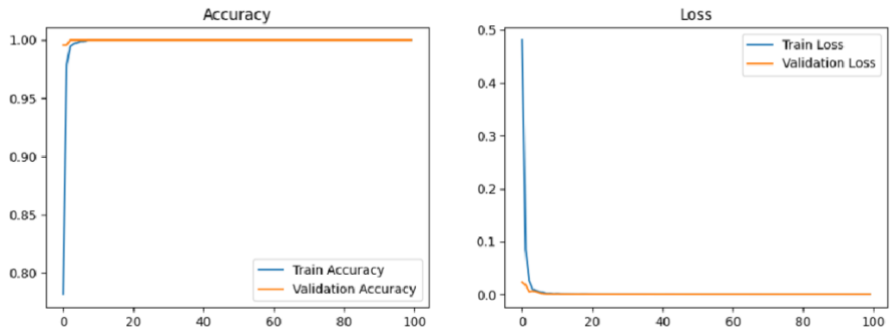


Figura 50 - Gráfico da accuracy e da loss da arquitetura Simple *GRU*

Embora os modelos tenham alcançado resultados notáveis no conjunto de treino, é importante enfatizar que os resultados altos das métricas nem sempre se traduzem numa capacidade robusta de generalização. Compreender os resultados obtidos é essencial para justificar a alta *accuracy* e a baixa *loss* alcançadas pelos modelos em estudo. Existem essencialmente três razões subjacentes a essas métricas elevadas:

- **Complexidade da tarefa** – Neste cenário específico, a tarefa de classificação pode ser considerada intrinsecamente simples. Isso significa que os modelos enfrentam uma tarefa relativamente fácil e direta em distinguir entre duas classes, o que pode contribuir para as altas precisões observadas.

- **Pré-processamento Simplificado:** Os dados armazenados em arquivos de texto requerem menos pré-processamento em comparação aos outros formatos de dados mais complexos. Isto pode economizar tempo e esforço na preparação dos dados, permitindo que os modelos se concentrem mais rapidamente na aprendizagem dos padrões relevantes.
- **Potencial para Memorização** - Dado o tamanho limitado do conjunto de dados, os modelos têm a capacidade de memorizar os exemplos de treino em vez de aprender padrões gerais. Isto pode resultar em altas precisões durante o treino, uma vez que os modelos se podem ajustar facilmente aos dados disponíveis.

6.1.4 Avaliação final do indivíduo

A abordagem no aprimoramento do sistema de avaliação envolveu experiências nas percentagens de importância atribuídas a cada modelo. Esta experiência permitiu o ajuste das ponderações atribuídas inicialmente para otimizar a avaliação final do sistema. As experiências foram divididas em três momentos, nos quais foram realizados dois tipos de avaliação distintos. O primeiro tipo de avaliação tem como foco principal enfatizar os modelos que possuem as principais características associadas a potenciais suspeitos. Já o segundo tipo de avaliação tem como ênfase os modelos que obtiveram os melhores resultados nas métricas, a fim de aprimorar a confiabilidade do sistema como um todo. As experiências realizadas estão demonstradas na tabela seguinte:

Tabela 8 - Experiências realizadas para ajuste das percentagens de cada modelo

	Características dos modelos	**Resultados dos modelos**
1ª experiência	✓	X
2ª experiência	X	✓
3ª experiência	✓	✓

A primeira experiência atribuiu 10% de importância à deteção de expressões faciais, reconhecendo a relevância de capturar pistas expressivas nos rostos dos indivíduos. No entanto, uma parcela menor foi destinada a esta categoria, uma vez que o reconhecimento de uma determinada expressão num indivíduo não indicar diretamente que ele possa ou não ser um criminoso. Em seguida, foi reservada 50% da importância para o reconhecimento de objetos. Esta parcela focou a capacidade de o sistema identificar objetos potencialmente perigosos ou relacionados a atividades suspeitas. Por fim, 40% de importância à deteção de ações, reconhecendo a necessidade de avaliar o comportamento e as ações dos suspeitos.

Na segunda experiência, foram atribuídos 20% de importância à deteção de expressões faciais, 70% ao reconhecimento de objetos e 10% à deteção de ações no sistema.

Esta alocação refletiu uma ênfase do modelo de reconhecimento de objetos devido à confiança atribuída aos resultados obtidos pela métrica *accuracy.* A percentagem de importância do modelo de expressões faciais foi também aumentada nesta segunda experiência e o modelo de deteção ações desempenharam papeis complementares.

Na terceira experiência, foi adotada uma abordagem que envolveu a união das percentagens atribuídas nas experiências anteriores. Esta união foi alcançada através da média das percentagens atribuídas aos modelos nas duas experiências anteriores, resultando numa distribuição equilibrada que teve em consideração tanto a ênfase nos resultados das métricas da *accuracy* como a caracterização de cada modelo. Tendo em conta o referido anteriormente, foi obtido uma distribuição de 15% para a deteção de expressões faciais, 60% para o reconhecimento de objetos e 25% para a deteção de ações.

As três experiências realizadas foram submetidas a um conjunto de teste composto por 150 vídeos, nos quais os indivíduos foram caracterizados como suspeitos, juntamente com 50 vídeos que apresentavam pessoas com comportamento considerado normal. O objetivo principal desses testes foi avaliar a capacidade de cada experiência em detetar a presença de um possível ladrão em cada um dos vídeos.

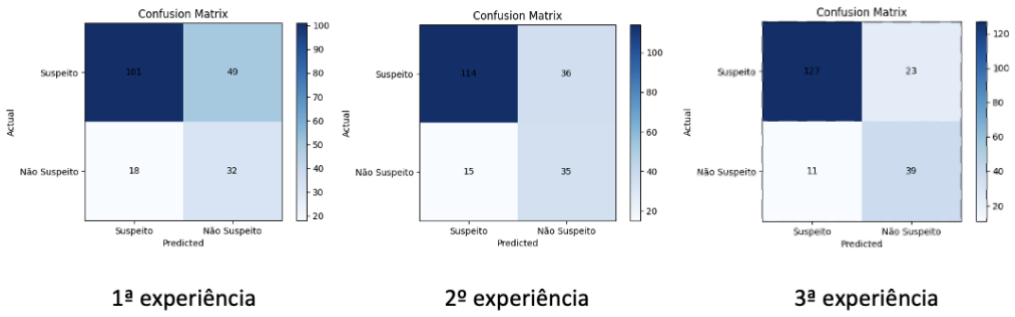


Figura 51 - *Confusion Matrix* das três experiências

Com base nos resultados apresentados pela *confusion matrix*, verifica-se que a variação do método de avaliação, e consequentemente das percentagens atribuídas, influenciou significativamente os resultados. A avaliação com base nas características dos modelos apresentou o desempenho inferior, com 101 avaliações corretas de um total de 150 para suspeitos e 32 avaliações corretas de um total de 50 para não suspeitos.

Em relação à segunda experiência, o sistema identificou corretamente 114 vídeos com presença suspeita e 35 avaliações corretas para não suspeitos.

Por fim, a experiência que combinou as duas abordagens obteve os melhores resultados, com 127 avaliações corretas num total de 150 para suspeitos e 39 avaliações corretas num total de 50 para não suspeitos, apresentando uma *accuracy* de 0,83.

7 Conclusão e Trabalho Futuro

A evolução da criminalidade constitui uma preocupação para os proprietários de bens e propriedades. No âmbito deste projeto, os avanços tecnológicos, em particular no campo do *deep learning*, desempenharam um papel crucial na luta contra o crime, disponibilizando ferramentas mais sofisticadas para a identificação de potenciais criminosos.

A caracterização dos ladrões é realizada considerando três aspetos fundamentais: as expressões faciais, os objetos e o tipo de vestuário que utilizam e as ações que executam. Tendo isto em consideração, o sistema desenvolvido foi estruturado em quatro fases distintas, com três delas dedicadas à deteção dessas características específicas. A quarta fase, por sua vez, consiste na integração dos resultados de todos os modelos.

Durante o processo de implementação, foi adquirido um vasto conhecimento relacionado ao processamento de imagem e vídeo, à criação e ajuste de modelos de *deep learning*, bem como componentes mais específicos, como o tipo de *layers* disponíveis e as suas funções, de como o conjunto de dados deve ser composto, entre outros aspetos.

No modelo de deteção de expressões faciais, foram realizadas várias experiências ajustando os hiperparâmetros. Os hiperparâmetros que melhor se ajustaram ao modelo tiveram bons resultados, com a *loss* a apresentar valores de aproximadamente 1,00, a *accuracy* de 0,64 e a *precision* 0,74. Ao comparar estes resultados com outros modelos, observou-se que o desempenho foi considerado satisfatório em relação à métrica da *accuracy* avaliada.

No contexto do modelo de deteção de objetos, as experiências foram conduzidas com as diferentes variantes oferecidas pelo YOLOv8, sendo elas o YOLOv8n, YOLOv8s e YOLOv8m. As três variantes analisadas apresentaram resultados bastante satisfatórios e semelhantes nas métricas de avaliação. A variante escolhida foi o YOLOv8n por este apresentar tempos de

deteção bastante inferiores aos demais devido a ser uma rede mais pequena. Os resultados mais destacados alcançados por este modelo foram a *loss* com 0,48, a *precision* 0,95, a *recall* 0,79, a *mAP@50* 0,95 e a *mAP@50-95* 0,73.

No terceiro e último modelo de deteção deste sistema, o modelo de deteção de ações, foram realizadas três experiências com diferentes arquiteturas: *LSTM*, *Simple RNN* e *GRU.* Os resultados obtidos através destas arquiteturas apresentaram valores muito idênticos, mas apresentando-se melhor a LSTM com valores da *loss* muito próximo de 0% e a *accuracy* muito próxima dos 100%.

Por fim, de forma a otimizar o sistema, foram manipuladas as percentagens de importância de cada modelo no resultado do sistema. Foram realizadas experiências que deram enfase a dois atributos: as características que o modelo e os resultados dos modelos. Para além disso, existiu uma terceira experiência com a média das percentagens atribuídas nas duas primeiras experiências, sendo esta a que melhor resultado obteve, com 127 avaliações corretas de um total de 150 para suspeitos e 39 avaliações corretas de um total de 50 para não suspeitos.

O sistema implementado obteve bons resultados e pode ser uma contribuição valiosa para a área da segurança de propriedades. A pesquisa realizada nesta dissertação também será de grande ajuda para a comunidade científica no desenvolvimento de sistemas na área de proteção de propriedades e bens.

Como perspetiva de trabalho futuro, está planeado alargar de forma substancial o dataset utilizado na deteção de ações. Para além disso, é pretendido aplicar a técnica *data augmentation*, de modo a abranger uma variedade mais ampla de situações e contextos. Outras abordagens, como a utilização de arquiteturas de modelos diferentes ou a utilização de um único modelo de classificação geral também poderiam ser investigadas.

Referências

Alves.G. (2018). *Entendendo Redes Convolucionais (CNNs) | by Gisely Alves | Neuronio BR | Medium*. https://medium.com/neuronio-br/entendendo-redes-convolucionais-cnns-d10359f21184

CS231N. (2014). *Convolutional Neural Networks for Visual Recognition*. https://cs231n.github.io/convolutional-networks/

Gan, Y. (2018). Facial expression recognition using convolutional neural network. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/3271553.3271584

IBM. (2021). *What are Neural Networks?* https://www.ibm.com/topics/neural-networks

Idrissi, J., & Amine, M. (2016). *Multilayer Perceptron: Architecture Optimization and Training*. https://doi.org/10.9781/ijimai.2016.415

Kakadiya, R., Lemos, R., Mangalan, S., Pillai, M., & Nikam, S. (2019). AI Based Automatic Robbery/Theft Detection using Smart Surveillance in Banks. *Proceedings of the 3rd International Conference on Electronics and Communication and Aerospace Technology, ICECA 2019*, 201–204. https://doi.org/10.1109/ICECA.2019.8822186

Keras. (2023). *Keras: Deep Learning for humans*. https://keras.io/

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/NATURE14539

Pacheco C., & Pereira N. (2018). *Vista do Deep Learning Conceitos e Utilização nas Diversas Áreas do Conhecimento*. http://anais.unievangelica.edu.br/index.php/adalovelace/article/view/4132/2770

Parlevel Systems. (2014). *Steps to Avoid Vending Machine Theft and Beef up Your Security*. https://www.parlevelsystems.com/2014/01/01/vending-theft/

Python. (2023). *Python*. https://www.python.org/downloads/

Quinn Minh-An, Reis Guilherme, & Sivesind Grant. (2017). *Real-time Emotion Recognition from Facial Expressions*. https://doi.org/10.1016/j.neunet.2014.09.005

Raspberry Pi Trading Ltd. (2021). *Raspberry Pi 4 Computer Model B*. www.raspberrypi.org

Sambare, M. (2020). *FER-2013*. https://www.kaggle.com/datasets/msambare/fer2013

Sarang Narkhede. (2018). *Understanding Confusion Matrix | by Sarang Narkhede | Towards Data Science*. https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

Saraswathy C, Karthikeyan S, & Ishwarya S. (2021). IDENTIFICATION OF THIEF USING ARTIFICIAL INTELLIGENCE BASED ON FACE AND ACTION RECOGNITION. *International Research Journal of Modernization in Engineering Technology and Science Www.Irjmets.Com @International Research Journal of Modernization in Engineering*, 2582–5208. www.irjmets.com

Sharma S. (2017). *Activation Functions in Neural Networks | by SAGAR SHARMA | Towards Data Science*. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Statista. (2012). *Vending machines - total vended volume in the U.S. 1999-2010*. https://www.statista.com/statistics/200646/total-sales-volume-of-vending-machines-in-the-us-since-1999/

Talegaonkar, I., Joshi, K., Valunj, S., Kohok, R., & Kulkarni, A. (2019). *Real Time Facial Expression Recognition using Deep Learning*. https://ssrn.com/abstract=3421486

Tensorflow. (2023). *TensorFlow*. https://www.tensorflow.org/?hl=pt-br