INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Tradução Vídeo de Língua Gestual Portuguesa: Reconhecimento Dinâmico de Configurações de Mão

NUNO MIGUEL VELUDO PEREIRA
Outubro de 2020

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Tradução Vídeo de Língua Gestual Portuguesa:

Reconhecimento Dinâmico de Configurações de Mão

Nuno Miguel Veludo Pereira

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas de Informação e Conhecimento

Orientador: Nuno Escudeiro

Porto, outubro de 2020

Resumo

Atualmente, a comunidade surda representa uma parte significativa da população em Portugal. Sendo a audição um dos principais sentidos do ser humano, gera dificuldades no dia-a-dia quer a nível profissional e académico. São necessários recursos e produtos inovadores que permitam facilitar a inclusão das pessoas surdas.

Uma das limitações mais significativas da surdez, é a dificuldade na comunicação. Para isso, necessitam de recorrer à Língua Gestual Portuguesa (LGP), constituída por um conjunto de gestos com diferentes significados, que permitem comunicar com outras pessoas. No entanto, considerando que a maioria da sociedade não possui conhecimento de Língua Gestual Portuguesa, a comunicação para a comunidade surda torna-se um problema quando tentam estabelecer uma conversação com pessoas sem estes conhecimentos.

Esta dissertação pretende apresentar todo o processo de investigação e desenvolvimento de uma aplicação capaz de, através de vídeo, identificar os gestos, ou configurações de mão, característicos da Língua Gestual Portuguesa, classificando-os e traduzindo-os para texto.

Com isto, pretende-se uma solução que sirva de base para oferecer novas oportunidades à comunidade surda em Portugal e promover a inclusão destes nas diversas necessidades e atividades do dia-a-dia.

Palavras-Chave: surdez, surdos, audição, comunidade surda, Língua Gestual Portuguesa, visão computacional, aprendizagem máquina, classificação

Abstract

Currently, the deaf community represents a significant part of the population in Portugal. Since hearing is one of the main senses of the human being, it generates difficulties in daily life, both professionally and academically. Innovative resources and products are needed to facilitate the inclusion of deaf people.

One of the most significant limitations of deafness, is the difficulty in communication. For this, they need to use the Portuguese Sign Language (PSL), consisting of a set of gestures with different meanings, which allow communication with other people. However, considering that most of society has no knowledge of the Portuguese Sign Language, communication for the deaf community becomes a problem when trying to establish a conversation with people without this knowledge.

This dissertation aims to present the whole process of research and development of an application capable, through video, of identifying the gestures, or hand configurations, characteristic of the Portuguese Sign Language, classifying them and translating them into text.

With this, it is intended a solution that serves as a basis to offer new opportunities to the deaf community in Portugal and promote their inclusion in the various needs and activities of daily life.

Keywords: deafness, deaf, hearing, deaf community, Portuguese Sign Language, computer vision, machine learning, classification

Conteúdo

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

LGP	Língua Gestual Portuguesa
CNN	*Convolution Neural Network*
SVM	*Support Vector Machines*
MLP	*Multilayer Perceptron*
KNN	*K-Nearest Neighbors*
ROC	*Receiver Operating Characteristic*
AUC	*Area Under Curve*

1 Introdução

Neste primeiro capítulo, é feita uma introdução ao contexto do projeto, enquadrando a comunidade surda e as suas dificuldades na sociedade atual. Consequentemente, é apresentado o problema em questão, os objetivos e a abordagem para a resolução do problema identificado. Por fim, é também apresentada a estrutura desta dissertação.

1.1 Contexto

De acordo com estudos realizados, atualmente existem cerca de 85 mil surdos e cerca de 1 milhão de pessoas sofre de deficiência auditiva, em Portugal (Gomes, 2019; Mini Som, 2014). São números relevantes que revelam uma comunidade que necessita de apoios de modo a facilitar as suas atividades do dia-a-dia.

Deste modo, é fundamental proporcionar a estas pessoas os meios necessários para conseguirem ter uma vida social, pessoal e profissional ativa, considerando a Língua Gestual Portuguesa como base para isso. Porém, são poucas as pessoas com conhecimentos de língua gestual, revelando uma barreira entre o surdo e os restantes interlocutores, serviços ou sistemas.

1.2 Problema

A comunicação entre surdos e ouvintes é exigente e pouco eficaz dado que estes não utilizam uma língua comum entre si. Estas limitações dificultam a inclusão dos surdos na vida ativa e na sociedade em geral.

As tecnologias e ferramentas que permitam colmatar estas limitações e dificuldades ainda são escassas e pouco desenvolvidas. A tradução automática de língua gestual para correspondente língua oral pode ser conseguida automaticamente e sem equipamento evasivos, através do processamento de imagens ou vídeos.

Um sistema que possibilitasse a tradução de língua gestual para texto, seria capaz de promover o melhor acesso dos surdos à educação, ao mercado de trabalho ou à cidadania em geral, com rapidez e baixo custo, contribuindo para a inclusão e a igualdade de oportunidades. Uma das bases para a língua gestual é a configuração das mãos nas diversas palavras, números

ou letras. Assim, a sua análise através de vídeo e a sua respetiva tradução, podem facilitar o dia-a-dia dos surdos em tempo real, podendo ser usada na comunicação com outros interlocutores.

1.3 Objetivo

O principal objetivo deste trabalho é desenvolver um sistema capaz de fazer a tradução de configurações de mão, esquerda e direita, da Língua Gestual Portuguesa de vídeo para texto. Este sistema representa um primeiro passo para um problema complexo, mas que, com trabalho futuro, poderá ser capaz de promover a inclusão e a igualdade de oportunidades da comunidade surda. Como tal, é necessário um estudo do estado da arte, analisando ferramentas capazes da tradução automática de língua gestual através de imagem ou vídeo. A longo prazo, pretende-se desenvolver um sistema base para facilitar a comunicação de surdos com interlocutores que não usam Língua Gestual Portuguesa.

1.4 Contributos esperados

O sistema desenvolvido deve permitir aos seus utilizadores traduzir gestos da Língua Gestual Portuguesa automaticamente para texto. Desta forma, espera-se contribuir com investigação e desenvolvimentos numa área e problema pouco explorados, e assim, no futuro, uma das principais barreiras entre surdos e interlocutores que não tenham conhecimentos sobre língua gestual poderá ser minimizada, permitindo uma maior inclusão dos surdos na sociedade.

1.5 Abordagem preconizada

Inicialmente, para a resolução do problema apresentado anteriormente, é necessário efetuar uma análise do estado da arte e avaliar as soluções, ferramentas e tecnologias que atualmente existem para visão computacional e como se comportam. De seguida é fundamental analisar quais podem ser utilizadas para a resolução deste projeto, e explorar a sua utilização. Com isto, será possível fazer, a partir do vídeo, a extração das mãos e do respetivo gesto, utilizando *datasets* e algoritmos de classificação para a identificação final.

1.6 Organização da dissertação

Esta dissertação encontra-se dividida em dez capítulos principais, abordando diferentes temas em cada subcapítulo.

O primeiro capítulo pretende apresentar ao leitor o contexto da dissertação, o problema que a originou e o objetivo que se pretende atingir. São ainda referidos os resultados esperados e é realizada uma breve análise de valor.

O segundo capítulo pretende aprofundar o contexto do problema, apresentar o estado da arte, ao nível das soluções existentes atualmente e outras ferramentas úteis.

O terceiro capítulo analisa o valor da solução proposta, identificando as suas oportunidades, ideias, conceitos e valor para o cliente. Também é feita uma comparação de tecnologias recorrendo ao método *Analythic Hierarchy Process* (AHP).

O quarto capítulo tem como objetivo relacionar o problema e objetivo com a diferentes tecnologias, soluções e abordagens existentes, identificadas no capítulo anterior, e introduzir ao leitor à abordagem a tomar no desenvolvimento da solução.

No quinto capítulo é efetuada e apresentada o design da solução desenvolvida, do ponto de vista tecnológico e arquitetural.

No sexto capítulo é descrito e demonstrado todo o processo de desenvolvimento do ponto de vista prático da aplicação.

No sétimo capítulo são avaliados e apresentados os resultados obtidos com a solução desenvolvida, com descrição das diversas experiências realizadas para demonstrar a eficácia da mesma.

No oitavo capítulo é demonstrada a integração do trabalho desenvolvido na aplicação SignToText.

No nono capítulo são apresentadas as diversas conclusões do trabalho desenvolvido, relacionando os resultados com os objetivos esperados. São também referidas possíveis melhorias do sistema a realizar no futuro e é apresentada uma reflexão final sobre todo o trabalho desenvolvido.

Por fim, são apresentadas as diversas referências utilizadas ao longo do desenvolvimento deste projeto e escrita deste relatório.

2 Contexto e Estado da Arte

Neste capítulo é aprofundado o contexto em que o problema se enquadra, identificando os diferentes interveniente e restrições.

Por fim, é efetuada a análise do estado da arte, identificando, em termos de soluções e ferramentas, aquilo que já existe e os projetos que já foram realizados e que podem servir como base para o desenvolvimento deste.

2.1 Contexto e Problema

De acordo com um estudo publicado pelo Instituto Nacional para a Reabilitação (Diário de Notícias, 2016; Isabel Ferreira Batista et al., 2013), em 1996 existiam em Portugal cerca de 19172 pessoas com surdez e 115066 pessoas com deficiência auditiva. O Instituto Nacional de Estatística e o Censos realizado em 2001, registaram 84172 deficientes auditivos, uma incidência de 0,8% de surdez para 10 milhões de habitantes em Portugal. Atualmente, apesar da incerteza relativamente ao número de pessoas com surdez, estima-se que cerca de 85 mil pessoas sofrem deste problema. São números relevantes e impactantes, que, se não forem considerados com a necessária preocupação, podem contribuir para uma desigualdade de oportunidades e exclusão social.

A Federação Portuguesa das Associações de Surdos é a instituição social, sem fins lucrativos, representativa das diferentes associações de surdos existentes em Portugal e, atualmente, está filiada com 10 associações espalhadas por todo o país: Amadora, Águeda, Alta Estremadura, Linha de Cascais, Guimarães e Vale do Ave, Algarve, Porto, Guarda e Évora (Federação Portuguesa das Associações de Surdos, 2020). A estas junta-se também a Associação Portuguesa de Surdos, existente desde setembro de 1958 (Associação Portuguesa de Surdos, 2012).

Ao longo dos últimos anos estas associações têm defendido os direitos das pessoas surdas e atualmente existem cerca de 17 escolas de referência com serviços de educação especial e com educação bilingue de alunos surdos, nos diversos ciclos de ensino – pré-escolar, primário, básico e secundário (Surdos Oralizados Portugueses, 2020). Em julho de 2019, foi lançada uma aplicação Android direcionada às pessoas surdas e que permite a estes fazerem uma videochamada ou trocarem SMS com a central do 112, facilitando a atuação destes em casos de emergência (Gomes, 2019).

No entanto, a comunidade surda continua a enfrentar problemas diários na sociedade devido à sua limitação. A comunicação entre as comunidades surdas e ouvintes, continua a ser o principal problema, visto que a maioria da população ouvinte não sabe Língua Gestual Portuguesa e não existem soluções capazes de permitir a comunicação entre ambos. Existe também pouca investigação e desenvolvimento nesta área e pouco foco para tentar solucionar este problema. Tal como referido anteriormente, isto pode resultar no isolamento social e aumento da dependência, limitando as atividades sociais e podendo mesmo levar à exclusão social (Isabel Ferreira Batista et al., 2013).

2.1.1 Processos e Intervenientes

Surdos

As pessoas surdas sofrem de problemas de comunicação porque, a maior parte das vezes, o ouvinte não sabe língua gestual. Igualmente, a pessoa surda também tem uma grande dificuldade em ler ou escrever, tendo em conta que não consegue interiorizar os sons associados a cada sílaba ou palavra. Como tal, geralmente estas pessoas têm dificuldades para comunicar com o mundo exterior, necessitando da companhia de alguém com conhecimentos de língua gestual ou da contratação de profissionais de tradução. Atualmente, são escassas as soluções tecnológicas capazes de atenuar este problema.

Ouvintes

Apesar de os principais atores serem as pessoas surdas, devemos considerar também as pessoas que os rodeiam, como por exemplo os recetores da mensagem da pessoa surda. Ou seja, é importante que as pessoas sejam capazes de entender a mensagem que os surdos estão a transmitir sem necessitar de recorrer a especialistas, tradutores ou a formações na área, que requerem tempo e dinheiro.

2.1.2 Restrições

Existem algumas restrições neste projeto que podem influenciar o desenvolvimento do mesmo. Por exemplo, em toda a extensão da Língua Gestual Portuguesa existem inúmeras configurações de mão e gestos que significam diferentes letras, palavras e frases. De igual forma, as línguas gestuais diferem de país para país, o que significa que, por exemplo, uma pessoa que comunica através da língua gestual portuguesa, não vai conseguir comunicar eficazmente com

outra pessoa que apenas tem conhecimentos sobre a Língua Gestual Americana. Ou seja, tal como as línguas faladas, as línguas gestuais também variam na sua sintaxe de país para país. Uma consequência disso é que materiais e informações que poderiam ser úteis a este projeto, como conjuntos de dados (*datasets*) e outras utilidades relacionadas com a Língua Gestual Portuguesa, são escassos ou inexistentes, necessitando de serem criados de raiz, alocando tempo e recursos.

2.2 Estado da Arte

Este projeto utiliza imagens e vídeos para a resolução do problema e, portanto, enquadra-se na área da visão computacional e de aprendizagem máquina, ou de *machine learning*. Relativamente ao mercado, neste capítulo são apresentadas algumas soluções existentes que, de certa forma, se assemelham ao problema apresentado. Do ponto de vista tecnológico, é apresentada a importância do pré-processamento de imagens no contexto de visão computacional e são apresentados diferentes tipos existentes. Por fim, no contexto de aprendizagem máquina e classificação, são apresentados diferentes algoritmos existentes.

2.2.1 Visão Computacional e Aprendizagem Máquina

O ser humano, recorrendo à sua visão, é capaz de observar e retirar informação do que o rodeia. Por exemplo, é capaz de olhar para uma planta num vaso e constatar quais são as suas caraterísticas, como cor, tipo de folha, entre outras. Da mesma forma que tem a capacidade de olhar para uma fotografia de grupo e identificar a emoção percetível que cada pessoa transmite: felicidade, tristeza, indiferença. Também muitos dos animais utilizam a sua visão para garantir a sua sobrevivência, como para identificar certos perigos ou para localizar alimento.

Assim, a visão computacional é a área capaz de automatizar e integrar uma vasta gama de processos e representações utilizados para a perceção da visão, recorrendo a vídeo e imagens (Ballard & Brown, 1982). Os seus objetivos podem ser analisados em duas áreas distintas. Do ponto de vista biológico, pretende a criação de modelos computacionais do sistema visual humano. Do ponto de vista tecnológico, a visão computacional consiste na construção de sistemas autónomos, capazes de realizar tarefas que o ser humano em conjunto com o seu sistema visual pode realizar (Huang, 1997).

Com a evolução do poder computacional e da inteligência artificial, a visão computacional é uma das áreas em constante crescimento e possuí várias aplicações na sociedade (Szeliski, 2010):

- **Retalho:** deteção de objetos em caixas de pagamento automáticas;
- **Medicina:** identificação de possíveis tumores através da análise de exames realizados;
- **Condução autónoma:** deteção de obstáculos inesperados, sinais e linhas de trânsito, entre outros;
- **Modelos 3D:** construção de modelos 3D de, por exemplo, monumentos, com base em fotografias aéreas.
- **Inspeção de máquinas:** validação do estado de máquinas como, por exemplo, de aerogerador de uma central eólica com recurso a drones.

A visão computacional é uma área de extrema complexidade. Enquanto que o ser humano consegue observar imagens e identificar as suas propriedades, formas, luminosidade e distribuições de cor instintivamente, sem praticamente qualquer esforço, os algoritmos de visão computacional são muito propensos ao erro, reduzindo a eficácia do tratamento dos dados.

A área de aprendizagem máquina, consiste na programação de computadores para otimizar um critério de desempenho utilizando dados de treino ou experiências passadas. Mais concretamente, num modelo, definido com certos parâmetros, a aprendizagem é a execução de um computador para otimizar os parâmetros do modelo (Alpaydin, 2014).

Nas últimas décadas, tem havido um grande interesse no desenvolvimento de técnicas de aprendizagem máquina para aplicações baseadas em visão computacional. Os seus algoritmos podem ser aplicados de pelo menos duas maneiras diferentes (Sebe et al., 2005):

- Para melhorar a perceção do ambiente em redor, isto é, para melhorar a transformação de sinais detetados para representações internas;
- Para colmatar a lacuna entre as representações internas do ambiente e a representação dos conhecimentos necessários para que o sistema cumpra a sua tarefa.

Com isto, é possível contastar que estas duas áreas estão totalmente relacionadas. Existem diversos paradigmas de aprendizagem máquina que podem ser aplicados ao domínio da visão computacional, como: concetuais (*decision trees* – árvores de decisão), estatísticos (*support vector machines*) e redes neuronais (Sebe et al., 2005). Alguns destes paradigmas são aprofundados mais à frente neste relatório.

2.2.2 Análise de soluções existentes

Tal como referido em capítulos anteriores, a existência de aplicações capazes de solucionar os problemas sentidos pela comunidade surda e pelos ouvintes que necessitam de recorrer à língua gestual para comunicar, são ainda escassas. É uma área que tem tido alguns avanços em termos de estudo e investigação, mas continuam a faltar alternativas para colmatar as dificuldades sentidas. A língua gestual é um problema complexo, especialmente considerando toda a sua flexibilidade e alterações de país para país. No entanto, apesar de escassas, é possível encontrarem-se algumas soluções que apresentam resultados interessantes e que, com mais algum crescimento, podem-se tornar numa solução viável no futuro.

Assim, de seguida são apresentadas algumas soluções interessantes que procuram solucionar alguns dos problemas que afetam a comunidade surda e as suas igualdades na sociedade. Todas estas são distintas entre si, apresentando diferentes abordagens tecnológicas para resolver um problema de considerável complexidade. De referir que, para além destas, existem outros projetos não aqui referidos por se tratar de soluções menos completas, maioritariamente desenvolvidas em contexto académico e sem atuação no mercado.

SignAll

O SignAll é um sistema pioneiro desenvolvido por uma *startup* de Budapeste, capaz de traduzir em tempo real língua gestual americana (ASL). Foi fundada em 2012 por János Rovnyai e Zsolt Robotka (Coldewey, 2018).

O sistema atual integra um Kinect 2 – um sensor de movimentos desenvolvido pela Microsoft – e três câmaras RGB, com diferentes pontos de vista. As câmaras são colocadas em ângulos laterais, de forma a captar dedos e gestos obstruídos, não visíveis na câmara frontal. Para uma tradução mais precisa, o processamento de vídeo não considera apenas o formato e os movimentos das mãos, mas também toda a parte superior do corpo e as diversas expressões faciais.

Todos estes fatores devem ser analisados para identificar frases completas e, portanto, não podem simplesmente ser traduzidos gesto a gesto. Assim, este é o maior desafio para a empresa, que aplicou o seu primeiro sistema piloto na Universidade de Gallaudet, uma universidade para surdos localizada nos Estados Unidos.

Como não existe uma representação 3D de todos os gestos possíveis, a SignAll define as suas bases de dados de acordo com o local onde o sistema é aplicado (Coldewey, 2018).

MyoSign

O MyoSign (Q. Zhang et al., 2019), desenvolvido na Universidade Jiao Tong, em Xangai, é um sistema baseado em *deep learning* que permite um reconhecimento de ponta-a-ponta de Língua Gestual Americana (LGA). É capaz de traduzir palavras e frases completas (Q. Zhang et al., 2019).

Este foi desenvolvido com o intuito de traduzir palavras solta e frases completas. A tradução de frases é um processo complexo e de difícil análise e deteção, porque os movimentos entre dois sinais contínuos dificultam a segmentação temporal.

Assim, o MyoSign foi criado considerando a utilização de uma *armband* com sensores de deteção inercial – acelerómetro de 3 eixos, giroscópio e orientação – e sinais de eletromiografia (EMG) de 8 canais. Os sensores de deteção inercial permitem capturar informações relativamente aos movimentos das mãos e braços, e o sensor EMG permite detetar os formatos da mão e padrões da atividade muscular.

Inicialmente, analisada cada série de dados obtida do fluxo de medições dos sensores, é elaborado uma *Convolutional Neural Network* (CNN) multimodal para aprender os diversos recursos locais de cada modalidade e os recursos globais de todas as modalidades. Posteriormente, estes são processados numa *Long Short Term Memory* (LSTM) bidirecional para modelar a dinâmica temporal. A combinação destas duas redes neuronais permite a abstração de relações intramodalidade, modalidade cruzada e temporal, promovendo a heterogeneidade do sensor e a diversidade do utilizador.

Para a interpretação de frases, foi considerada a língua gestual sem segmentação temporal como uma tarefa de aprendizagem de sequências supervisionada, ou seja, em que apenas a sequência dos gestos é fornecida, sem limites de tempo para cada um destes. Por fim, a *Connectionist Temporal Classification* (CTC) é aplicada sobre a LSTM para treinar todas as redes.

Todo este sistema permite a tradução de frases diferentes das utilizadas no processo de treino, eliminando a necessidade de recolher todas as frases existentes.

O resultado é a tradução de língua gestual efetuada pelo utilizador para língua inglesa, através do uso de um *smartphone* ou *smartwatch*. O sistema também é capaz de detetar a língua inglesa e convertê-la para texto, mostrando-o no aparelho utilizado pelo surdo. A análise do sistema revelou uma eficácia de 93.7% na tradução de palavras e 93.1% na tradução de frases, com configurações independentes do utilizador. Por fim, o MyoSign apresentou uma eficácia de 92.4% no reconhecimento de frases nunca testadas anteriormente.

VirtualSign

O VirtualSign (*VIRTUALSIGN*, 2020) é um projeto desenvolvido no Instituto Superior de Engenharia do Porto e enquadrado no grupo de investigação GILT (*Games, Interaction and Learning Technologies*) (*GILT*, 2020).

Este projeto (Escudeiro et al., 2015) tem como principal objetivo desenvolver e avaliar um modelo capaz de facilitar o acesso de surdos ou deficiente auditivos a conteúdos digitais, considerando principalmente o contexto da educação, como conteúdos educativos e outros objetos de aprendizagem. Para isto, foi desenvolvido um tradutor bidirecional, isto é, de língua gestual portuguesa para texto escrito e de texto escrito para os seus respetivos gestos em língua gestual portuguesa.

Na tradução de gestos para texto, é usado um Microsoft Kinect e umas luvas sensoriais que permitem recolher os dados necessários para a respetiva tradução. O Microsoft Kinect fornece dados sobre a orientação e movimentação das mãos, enquanto que o par de luvas (*5DT Data Gloves*) providencia informações sobre a configuração da mão, ou seja, o gesto. O conjunto destes dados é classificado, com recurso a um modelo de classificação, e de seguida o texto resultante é analisado e transformado numa frase. Por fim, este é apresentado ao utilizador.

Relativamente à tradução de texto para língua gestual portuguesa, é utilizado um Avatar para reproduzir os gestos com base no texto pretendido pelo utilizador. Inicialmente, o texto é pré-processado, a frase é analisada e, com recurso a uma base de dados texto/gesto, o gesto é identificado. Por fim, o gesto é reproduzido com recurso ao Avatar (ver Figura 1). Em termos de ferramentas tecnológicas, o Avatar foi criado com o Blender (*blender.org*, 2020), um programa para modelagem e animação. A base de dados é do tipo MySQL (*MySQL*, 2020), o código é escrito em C# e todas as funcionalidades são incorporadas entre si através de Unity (*Unity Real-Time Development Platform | 3D, 2D VR & AR Engine*, 2020).

O trabalho desenvolvido pela equipa do VirtualSign revela-se também interessante na etapa de criação do modelo de classificação. Para a classificação dos gestos, foram testados diversos algoritmos, com recurso ao *k-fold cross validation*, método de avaliação em que o *dataset* é dividido em *k* número de blocos – neste caso 10 – e por cada iteração um bloco diferente é reservado para teste e os restantes são utilizados para treinar os classificadores.

Assim, foram testados os algoritmos Random Trees (RT), Boost Cascade (BC), Neural Networks (NN), K-Nearest Neighbours (KNN), Naive Bayes (NB) e Support Vector Machine (SVM), dividindo-se a mão direita e a mão esquerda em testes distintos. No geral, os diversos algoritmos apresentaram resultados positivos, em especial o SVM, que revelou uma precisão de 98% e uma

exatidão de 100% para ambas as mãos. Os restantes resultados obtidos estão representados nas tabelas da Figura 2.

Figura 1 - Exemplo do VirtualSign

%	RT	BC	NN	KNN	NB	SVM
Precision	98,6	82,0	98,1	98,8	97,5	98,6
Accuracy	85,5	95,4	78,1	97,3	97,1	100,0

%	RT	BC	NN	KNN	NB	SVM
Precision	98,8	86,1	97,2	98,0	98,0	98,1
Accuracy	87,3	96,6	80,4	98,2	96,8	100,0

Figura 2 - Percentagens dos treinos do VirtualSign para mão direita (cima) e esquerda (baixo)

2.2.3 Reflexão sobre as soluções

As soluções reveladas anteriormente representam passos importantes no apoio à comunidade surta e às suas dificuldades. Demonstram que este é um problema real e que já existem equipas, profissionais e investigadores a trabalhar para o reduzir, promovendo alguma igualdade de oportunidades. No entanto, este não é um problema de apenas uma face. É complexo e revela-se nas mais diversas áreas da sociedade. Como tal, não é esperado que

apenas uma única solução seja capaz de eliminar todos os problemas existentes, principalmente quando o desenvolvimento nesta área ainda está pouco presente. Porém, é com um trabalho progressivo e com diversos desenvolvimentos que se pode chegar a uma solução ideal.

Posto isto, todas as soluções apresentadas, apesar de distintas, revelam-se muito interessantes e até inovadoras, no entanto, ainda não são perfeitas. Todas apresentam a necessidade de uma infra-estrutura complexa: o SignAll necessita de um Microsoft Kinect 2 e três câmaras RGB, o MyoSign recorre a uma *armband* com sensores de deteção inercial e o VirtualSign, na funcionalidade de tradução de língua gestual portuguesa, utiliza um Microsoft Kinect e umas luvas sensoriais. Esta necessidade de equipamentos complexos, caros e, no caso destas câmaras, pouco portáteis, revela um entrave para a utilização no dia-a-dia. De igual forma, também por serem ainda soluções recentes e em fase de experimentação, são soluções muito locais e pouco espalhadas, tendo um desafio grande pela frente de propagação por outros países e cidades, não esquecendo que os *datasets* precisam sempre de adaptação de acordo com a língua gestual do país em questão.

A solução apresentada neste relatório revela-se diferente das apresentadas principalmente na infraestrutura usada. A utilização de uma única câmara pode facilitar o seu uso no dia-a-dia, possibilitando uma maior flexibilidade e adaptabilidade aos diversos cenários na sociedade. No entanto, também é preciso reconhecer que não é possível apresentar uma complexidade tão elevada como as já existentes soluções, pois a tradução de gestos é apenas uma etapa de todo o processo de tradução da língua gestual, sendo necessária aliá-la aos movimentos do corpo e até às expressões faciais para uma funcionalidade completa. Porém, a tradução de gestos da língua gestual com baixos recursos já revela um passo importante e relativamente inovador.

2.2.4 Visão Computacional – Processamento de Imagem

Em aplicações que recorrem à visão computacional para classificação, é muito importante o processo de pré-processamento das imagens, isto é, as transformações que se efetuam às imagens captadas antes do processo de treino ou classificação final. Estes métodos de pré-processamento permitem uniformizar os dados entre os diversos recursos/*dataset* e melhorar a informação das imagens, eliminando dados desnecessários ou destacando a informação mais importantes. Considerando isto, o pré-processamento ideal pode variar de acordo com o projeto, algoritmo ou objetivo. Assim, uma escolha acertada relativamente a esta

etapa, pode revelar melhores resultados ao nível da geração e desempenho de modelos de classificação.

De seguida, são apresentados alguns dos processos de transformação de imagem essenciais e mais comuns, importantes no desenvolvimento desta solução.

Redimensionamento

O redimensionamento de imagem é uma das etapas mais importante do pré-processamento de imagens porque permite uniformizar o tamanho de um conjunto de imagens, melhorando o modo como um determinado algoritmo processa as mesmas (Canuma, 2018).

Transformação de Cor

A transformação da cor permite manipular as imagens de forma a destacar determinadas caraterísticas ou pontos da imagem.

Numa imagem RGB, cada píxel da imagem é especificado por três valores de cada uma das três cores primárias: vermelho (*red*), verde (*green*) e preto (*black*). Numa imagem de 24-bits, cada um destas cores apresenta 8-bits e um valor entre 0 e 255 e, em conjunto, definem a cor final do píxel (Verma, 2010). Assim, é possível alterar a cor de uma imagem manipulando os valores de RGB da mesma. Por exemplo, a transformação de uma imagem para preto e branco é muito comum na área de visão computacional pois permite uma maior distinção entre diferentes tons – claros e escuros. Esta conversão recorre ao espaço de cores RGB e existem diferentes algoritmos que o possibilitam, variando o seu resultado em termos de luminância, contraste ou sombras (Verma, 2010).

Para além do RGB, existem mais espaços de cor, como o BGR (blue, green, red), HSV (hue, saturation, value), CIE, entre outros, sendo o RGB o mais comum atualmente.

Desfoque Gaussiano

O desfoque gaussiano é muito simples e consiste numa desfocagem da imagem, de forma a reduzir ruído e detalhe. Esta suavização permite reduzir detalhe desnecessário, aumentando a eficácia do algoritmo (OpenCV, 2020b).

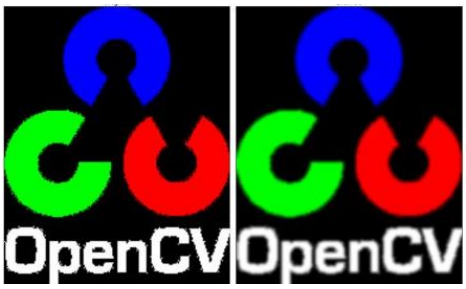

Figura 3 - Exemplo de desfoque gaussiano (OpenCV, 2020b)

Segmentação

A segmentação permite separar os elementos mais próximos dos mais distantes, ou seja, fazendo uma distinção entre o plano frontal e o plano de fundo. Esta distinção pode ser útil quando se pretende, por exemplo, descartar o plano de fundo e apenas utilizar a informação frontal (Canuma, 2018).

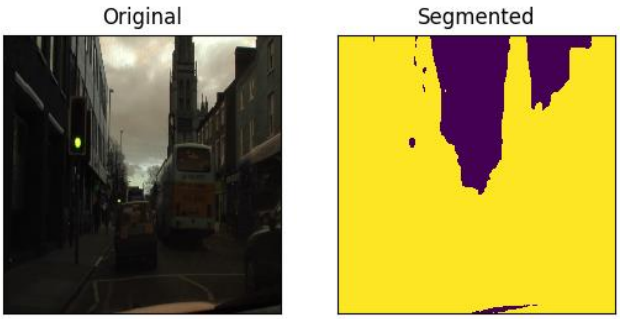

Figura 4 - Exemplo de Segmentação

Deteção de Limites

A deteção de limites permite extrair toda a informação da imagem e apenas destacar aquilo que são os limites dos diferentes elementos presentes. Este tipo de processamento associado à segmentação possibilita a deteção dos limites do plano frontal, removendo informação desnecessária da imagem (Dan, 2019).

Figura 5 - Exemplo de deteção de limites

Exemplo prático de pré-processamento

Numa faculdade da Índia (Nikam & Ambekar, 2017a, 2017b), foi desenvolvido uma aplicação capaz de efetuar a tradução de letras e números da língua gestual indiana para texto, através do uso de uma câmara. Neste projeto, a imagem capturada é transformada de forma a garantir as condições ideias para ser finalmente identificada.

Inicialmente, a imagem é segmentada, ou seja, é convertida de RGB para HSV (ver Figura 6) (Ribeiro & Gomes, 2010), permitindo identificar a mão do utilizador e os seus limites. Os recursos do HSV – *hue*, *saturation*, *value* – permitem ao utilizador especificar os limites da cor da pele em termos de matriz (*hue*) e saturação (passo A da Figura 7).

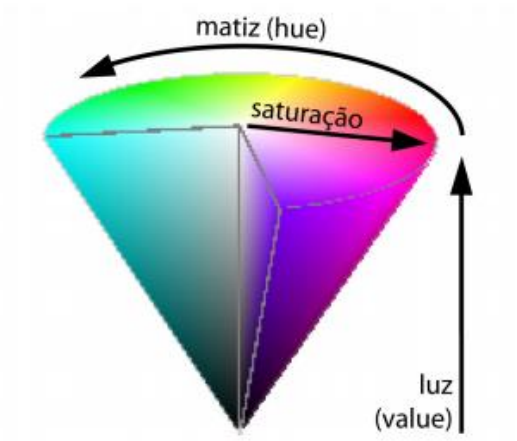

Figura 6 - Sistema de cores HSV (Ribeiro & Gomes, 2010)

Posteriormente, é realizado um processo de erosão e dilatação: a erosão permite diminuir os limites da imagem, ampliar os buracos e remover ruídos, e a dilatação é usada para adicionar pixéis na região dos limites, preencher buracos ampliados durante o processo de erosão e conectar pixéis soltos e adicionar pixéis nas bordas (etapas B e C da Figura 7).

Por fim e após um processo de suavização (etapa D da Figura 7), a imagem é convertida para uma imagem binária, ou seja, cada um dos pixéis é convertido para branco ou preto conforme a sua intensidade (etapa E da Figura 7) (Nikam & Ambekar, 2017a, 2017b).

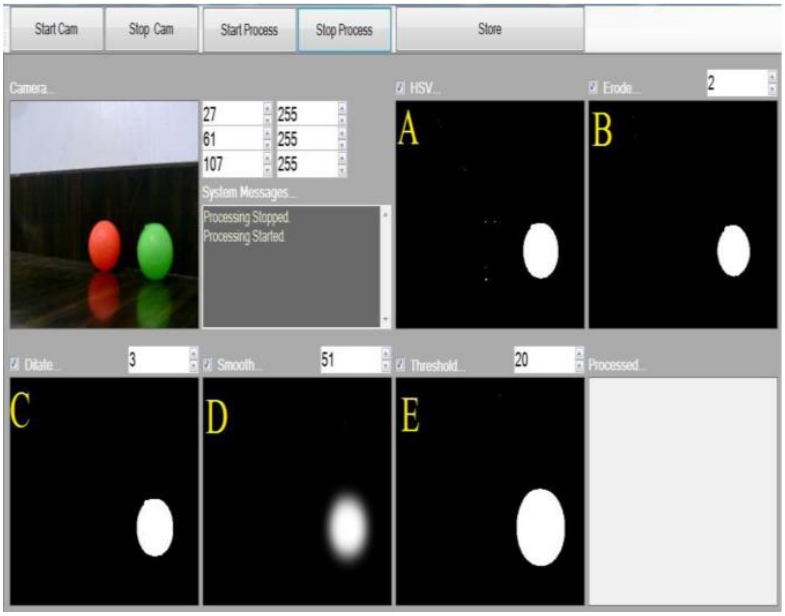

Figura 7 - Processos de transformação de imagem (Nikam & Ambekar, 2017a, 2017b)

Para o processo de reconhecimento, foram identificados pontos ou regiões da imagem que mais claros ou escuros que a área envolvente e, finalmente, foram usados algoritmos baseados no invólucro ou fecho complexo (ver Figura 8) (Martins, 2005), que permitiram o reconhecimento de configurações de mão (Nikam & Ambekar, 2017a, 2017b).

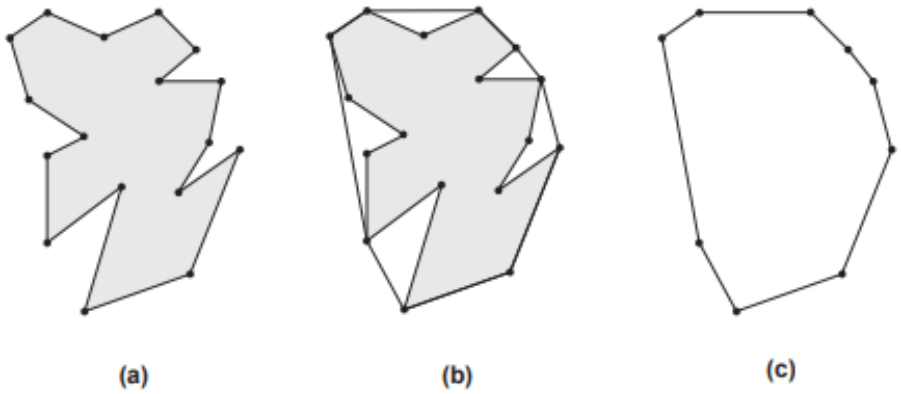

Figura 8 - (a) Polígono simples; (b) Invólucro convexo; (c) Polígono convexo (Martins, 2005)

2.2.5 Aprendizagem Máquina – Algoritmos de Classificação

Na área de aprendizagem máquina, a classificação é um processo de aprendizagem computacional em que um determinado programa aprende com os dados de entrada usando, posteriormente, estes dados para classificar novas observações (Sidana, 2017). Esta aprendizagem é feita recorrendo a algoritmos já existentes e devidamente testados, capazes de fazer diversos tipos de classificações com base no treino efetuado e nos modelos criados.

Tal como descrito no subcapítulo anterior, uma das abordagens utilizadas para identificar gestos ou configurações de mão consiste em identificar pixéis da mão, após o processamento de um *frame*/imagem, extrair a informação e, por fim, utilizá-la para reconhecer a posição em causa. Neste exemplo, a informação extraída, isto é, cada uma das diferentes posições dos pixéis da mão, pode estar associado a um gesto e, com a utilização destes dados para treino, recorrendo a um algoritmo de classificação, pode ser criado um modelo capaz de classificar futuras informações.

Assim, de seguida são descritos alguns algoritmos de aprendizagem e classificação existentes, que podem ser aplicados para a classificação de gestos.

K-Nearest Neighbour (k-NN)

Este algoritmo supõe que, numa dimensão, objetos semelhantes estão próximos uns dos outros. A classificação é conseguida através da identificação dos vizinhos mais próximos, denominados *k*, de um determinado dado e utilizando esses vizinhos para determinar a respetiva classe. Para isso, o k-NN captura a ideia de semelhança através do cálculo da distância entre pontos (ver Figura 9) (Harrison, 2018).

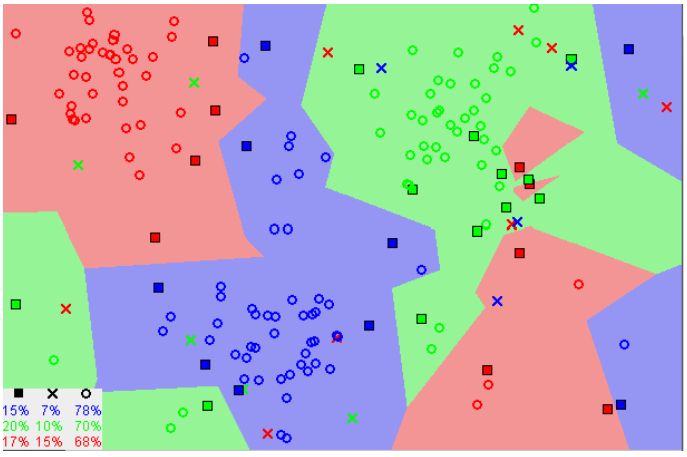

Figura 9 - Imagem representativa do K-Nearest Neighbour (wikimedia, 2020)

A Figura 10 (Cunningham & Delany, 2020) representa um exemplo de um algoritmo *3-Nearest-Neighbour*, ou seja, onde o *k* é 3, num problema com duas classes (X e O) e num espaço bidimensional (x e y). Analisando o gráfico, a classificação de q_1 é simples – como todos os vizinhos são da classe O, q_1 será classificado como O também. Relativamente a q_2, a sua classificação é mais complexa pois dois dos seus vizinhos são X e o restante é O. Como tal, a classificação de q_2 pode ser feita através de votação por maioria ou votação ponderada à distância.

Concluindo, a classificação k-NN é constituída por duas fases: a primeira é a determinação dos vizinhos mais próximos e a segunda é a determinação da classe utilizando esses vizinhos (Cunningham & Delany, 2020).

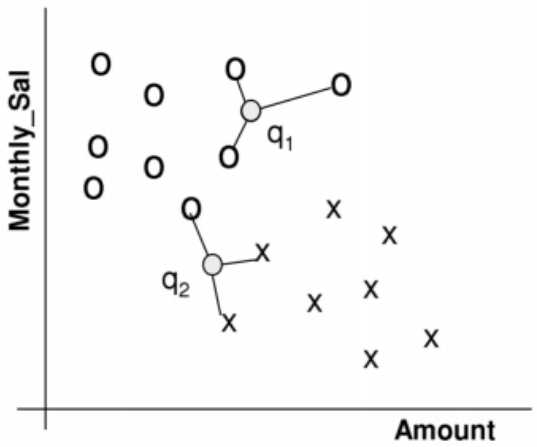

Figura 10 - Exemplo da análise do algoritmo kNN

Naïve Bayes (NB)

O classificador Naïve Bayes (Rish, 2014) é baseado no teorema de Bayes, que descreve a probabilidade de um evento, com base no conhecimento prévio de condições que possam estar relacionadas com o evento. Este classificador consegue simplificar significativamente a aprendizagem, assumindo que as características são independentes da classe e atribuindo a classe mais provável a um determinado dado com base no seu vetor de caraterísticas. Por exemplo, este permite calcular a probabilidade de Y (campo com *n* valores possíveis) ser *y*, considerando um ponto de dados X=(x_1, x_2,…, x_n). Isto pode-se traduzir na seguinte expressão (Z. Zhang, 2019):

$$P(Y = y \mid X = (X_1, X_2, \dots, X_k))$$

Support Vector Machines (SVM)

O algoritmo de classificação *Support Vector Machines* é capaz de encontrar um limite para separar diferentes classes. Essa linha ou limite encontrado é o hiper-plano que maximiza a margem ou distância entre diferentes classes (Chen, 2019).

Decision Tree

Tal como o nome indica, este algoritmo assemelha-se a uma árvore e aos seus respetivos troncos e ligações. Uma árvore de decisão é constituída por uma raíz (root), que não tem nenhumas ligações de entrada; nós de teste (*test nodes*), que são nós com ligações de saída e que representam atributos; ligações, que representam uma decisão ou regra; e por folhas (*decision nodes*), que representam uma decisão (Rokach & Maimon, 2006).

Para cada folha é atribuída uma classe que representa o valor alvo mais apropriado. A folha também pode ter associada um vetor de probabilidade, que indica a probabilidade de o atributo alvo ter um determinado valor. As instâncias da árvore são classificadas navegando desde a raiz da árvore até uma folha, de acordo com o resultado dos testes ao longo de cada nível da árvore (Rokach & Maimon, 2006).

Artificial Neural Network (ANN) e Convolution Neural Network (CNN)

Uma rede neuronal (*Neural Network*) é um classificador inspirado na aprendizagem dos neurónios do ser humano. Consiste numa rede artificial de funções, que permite ao computador aprender e ajustar-se consoante a introdução de novos dados, desenvolvido para processar conjuntos estruturais de dados, como imagens (Knocklein, 2019). É um dos algoritmos mais utilizados na área da visão computacional, apresentando resultados muito promissores.

Uma Rede Neuronal Artificial (*Artificial Neural Network*, ANN) é a implementação base das redes neurais, abrangendo várias arquiteturas como as Redes Neuronais Convolucionais (*Convolution Neural Network*, CNN) ou as Redes Neuronais Recorrentes (*Recurrent Neural Networks*, RNN).

As redes neuronais são constituídas por camadas (*layers*) com diferentes funcionalidades e tarefas. Assim, a principal diferença entre uma *Artificial Neural Network* e uma *Convolution Neural Network*, é que numa CNN apenas a última camada está totalmente ligada, enquanto que numa ANN cada neurónio está ligado a todos os outros neurónios (ver Figura 11) (Gogul & Kumar, 2017).

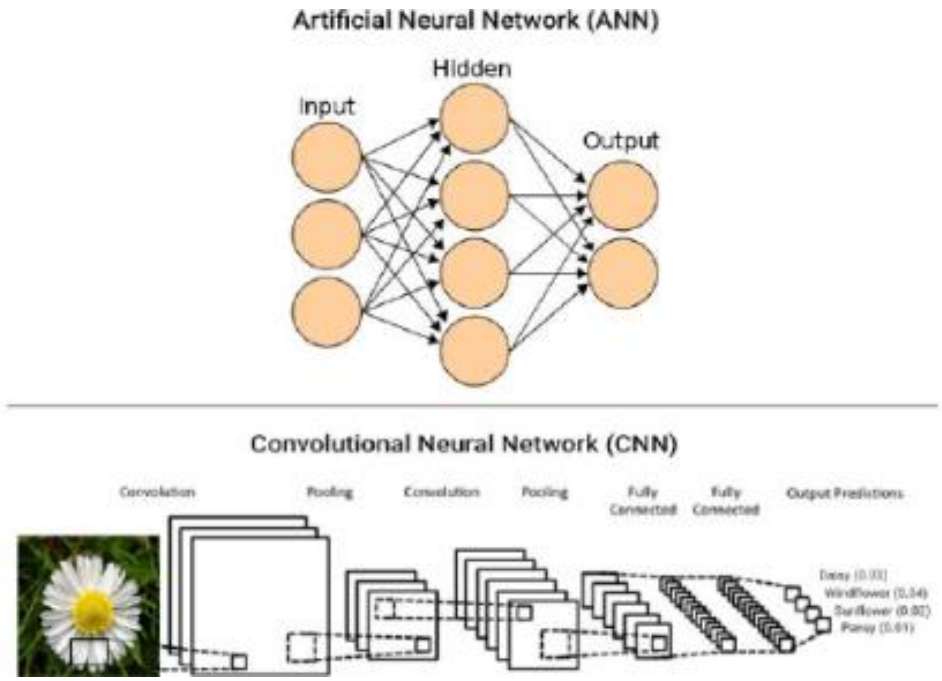

Figura 11 - Artificial Neural Network e Convolution Neural Network (Gogul & Kumar, 2017)

Nas redes neuronais convolucionais, a camada convolucional extrai caraterísticas importantes da imagem e guarda-as num mapa de caraterísticas (*feature map*), reduzindo o tamanho da imagem e aumentando assim a velocidade de processamento. Apesar de alguma informação da imagem ser perdida, as caraterísticas mais importantes para a classificação são preservadas (Jacovi et al., 2019).

Ao contrário das CNN, as ANN não são indicadas para imagens, pois os processamentos que estas fazem às imagens para as converter em vetores pode resultar facilmente em *overfitting*, ou seja, quando o modelo se ajusta demasiado bem ao conjunto de dados em questão (Gogul & Kumar, 2017).

Multilayer Perceptron (MLP)

O *Multilayer Perceptron* (MLP) é outro algoritmo de classificação inserido nas redes neuronais (NN) ou nas Redes Neuronais Artificais (ANN) mas com caraterísticas *feedforward*, isto é, em que as relações entre os nós não formam um ciclo. Isto significa que cada uma das ligações direcionam-se para o resultado ou para os dados de saída (*output*). Uma MLP e uma CNN são muito semelhantes, porém uma CNN contém camadas de convolução e *pooling* (Scikit-Learn, 2020a).

Estudo de comparação de algoritmos

Um estudo realizado (Trigueiros et al., 2012), comparou alguns destes algoritmos na classificação e identificação de gestos. Recorrendo as dois *datasets*, foram aplicados quatro algoritmos distintos – k-NN, NB, ANN e SVM. As diferentes experiências foram realizadas com o auxílio do método *k-fold*, que permite determinar com que precisão um algoritmo de aprendizagem será capaz de prever dados com os quais não foi treinado. Este estudo demonstrou que os algoritmos k-NN e ANN apresentam resultados muito promissores na classificação de gestos, como demonstra a seguinte tabela.

No entanto, estes dados, apesar de indicativos, podem variar de acordo com parâmetros, tempos de treino e qualidade do *dataset*.

Tabela 1 - Resultados da comparação de algoritmos (Trigueiros et al., 2012)

	Classifier	k-NN	Naïve Bayes	ANN	SVM
DataSet 1	Accuracy (%)	95,45	25,87	96,99	91,66
	Time	8''	1''	46'33''	3'10''
DataSet 2	Accuracy (%)	88,52	66,50	85,18	80,02
	Time	1''	1''	32''	1'08''

3 Análise de Valor

Neste capítulo, é efetuada a análise de valor, recorrendo ao modelo *New Concept Development* (NCD). É também identificado o valor que tem este projeto para o cliente e qual a sua proposta de valor, bem como se pretende entregar o valor ao cliente, através do modelo de Canvas. São ainda comparadas diferentes alternativas e decidida qual a mais adequada, de acordo com diferentes critérios e seguindo o método *Analythic Hierarchy Process* (AHP).

3.1 New Concept Development Model

O processo de inovação de um produto é constituído por três principais fases essenciais ao desenvolvimento do mesmo: *Fuzzy Front-End* (FFE), *New Product Development* (NPD) e comercialização (ver Figura 12) (Belliveau et al., 2002).

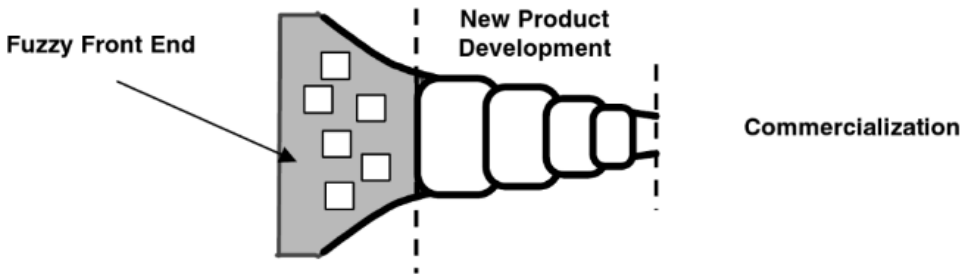

Figura 12 - Fases do processo de inovação

A fase FFE é geralmente considerada como uma das maiores oportunidades para melhorar o processo de inovação geral. Com o objetivo de fornecer uma visão e terminologia comum para o FFE (Belliveau et al., 2002), foi criado um modelo denominado *New Concept Development* (NCD) (ver Figura 13). Este é dividido em três partes:

- O **motor**, ou centro do modelo, responsável pela visão, estratégia, liderança e questões gerais de gestão que impulsionam os elementos de atividade da corporação;
- Os **cincos elementos** do FFE – identificação da oportunidade, análise da oportunidade, geração de ideias, seleção de ideias e definição do conceito;
- Os **fatores externos**, que influenciam o motor e os cinco elementos *Front-End* (Kahn et al., 2013).

Nas seguintes secções é efetuada uma análise aos cinco elementos do FFE, considerando o projeto em questão.

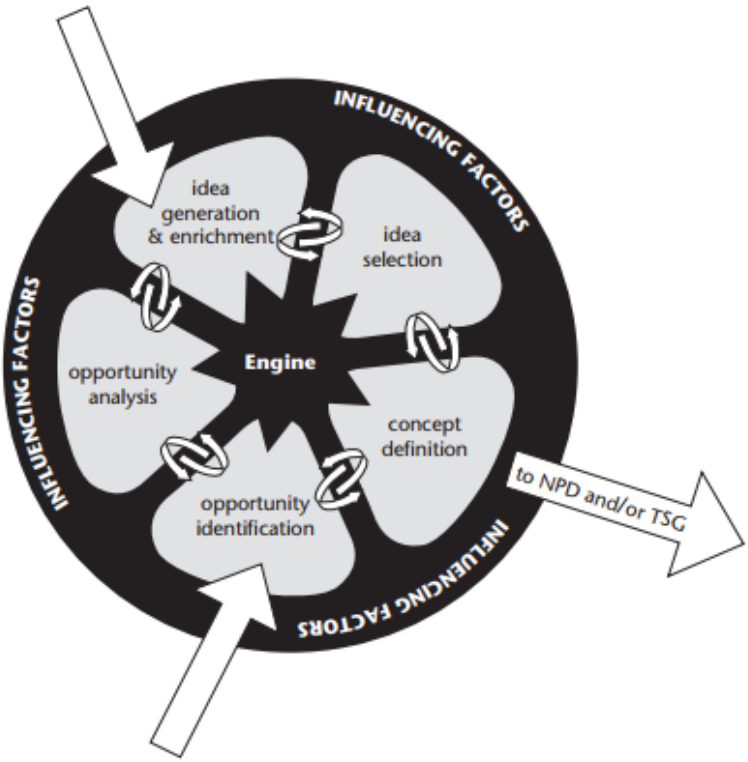

Figura 13 - New Concept Development (NCD) (Kahn et al., 2013)

3.1.1 Identificação da oportunidade

Para este elemento, o primeiro dos cinco elementos do FFE; são identificadas as oportunidades que se quer perseguir (Belliveau et al., 2002).

Neste projeto, foi detetado um problema relacionado com a comunicação das pessoas surdas e a falta de capacidade da sociedade em entender o modo de comunicação destas – a língua gestual. O VirtualSign é uma equipa do Instituto Superior de Engenharia do Porto, inserido no grupo de investigação GILT, focada em identificar problemas e dificuldades relacionadas com a surdez. Assim, após uma análise, foi identificada uma oportunidade de desenvolver uma solução capaz de atenuar este problema, possibilitando a tradução de gestos da língua gestual portuguesa.

3.1.2 Análise da oportunidade

Neste elemento, é avaliada a oportunidade para confirmar se justifica persegui-la e avançar para as próximas etapas (Belliveau et al., 2002).

De acordo com o Censos realizado em 2001 (Diário de Notícias, 2016), registaram 84172 deficientes auditivos, uma incidência de 0,8% de surdez para 10 milhões de habitantes em Portugal. São números consideráveis, que têm aumentado desde os anos 90.

Apesar de atualmente já existirem tratamentos cirúrgicos ou aparelhos especializados capazes de reduzir a surdez e outros problemas auditivos, muitos dos surdos não optam por estes, por impossibilidade ou problemas financeiros. Para além disso, estes apresentam uma eficiência variável.

Assim, a melhor forma destes comunicarem é através da língua gestual. No entanto, a maioria da sociedade não tem conhecimentos sobre esta forma de comunicação, excluindo algumas exceções como profissionais da área da tradução.

3.1.3 Geração e enriquecimento de ideias

Este elemento refere-se ao nascimento, desenvolvimento e amadurecimento de uma ideia concreta (Belliveau et al., 2002).

A geração e o enriquecimento da ideia surgiram de eventos, congressos e encontros com a comunidade surda, onde estes expõem as suas dificuldades e problemas que sentem no seu dia-a-dia enquanto membro ativo da sociedade. Estas dificuldades foram analisadas e tentou-se identificar possíveis soluções com o intuito de atenuar os impactos destas.

3.1.4 Seleção de ideias

Neste elemento, são selecionadas as ideias identificadas anteriormente.

Considerando as ideias resultantes da análise dos problemas identificados pela comunidade surda, optou-se por uma ideia atrativa e que pudesse representar, no futuro, um impacto significativo para os intervenientes em questão. Um dos problemas identificados é na comunicação com a restante comunidade envolvente, sendo que muitas das vezes não são compreendidos pois a maioria das pessoas não possui conhecimentos de língua gestual.

3.1.5 Definição de conceito

Tal como referido anteriormente, o número de pessoas surdas tem aumentado, apresentando números significativos em relação ao total da população. As associações de surdos têm tido um impacto interessante na luta pelos seus direitos, mas, apesar de já começarem a existir soluções de suporte, infraestruturas e programas especialmente designados para estes, continuam a existir lacunas ao nível da igualdade e de oportunidades. Desta forma, tendo em conta a análise realizada nos elementos anteriores, percebeu-se que uma das maiores necessidades é auxiliar a comunidade surda no momento da comunicação. Assim, decidiu-se desenvolver uma aplicação capaz de traduzir automaticamente gestos língua gestual para texto, criando uma base para uma solução futura que promova a comunicação entre surdos e outras pessoas sem conhecimentos de língua gestual nos mais diversos contextos da sociedade.

O sistema a desenvolver deve ser capaz de interpretar gestos da língua gestual e retirar a informação da mesma. Para isso, terá de efetuar a leitura de um vídeo e transformá-lo em diversos *frames*, isto é, imagens. Cada imagem contém um gesto, representando parte da mensagem que a pessoa está a transmitir, que, através de ferramentas de visão computacional e de modelos de classificação, é interpretado e traduzido para mensagem. Obviamente, para um resultado eficaz, é importante o vídeo captar com clareza a mão do comunicador.

3.2 Valor para o cliente

O conceito de valor é muito subjetivo e o seu significado está dependente do contexto em que se enquadra. Se considerarmos o contexto do *marketing*, o valor pode ser identificado pela utilidade e benefícios para o cliente (Sidorchuk, 2015). De acordo com o Professor Art Weinstein, a simples definição de valor não é suficiente para os clientes, sendo que estes querem que os negócios os surpreendam, indo acima do comum para satisfazer as suas necessidades e desejos. O valor para o cliente refere-se à capacidade das empresas em criar e acrescentar valor aos bens e serviços e pode ser definido em quatro pontos-chave (Mcfarlane, 2013):

- **Serviço –** o valor oferecido aos clientes;
- **Qualidade –** a perceção dos clientes sobre como os produtos e serviços de uma empresa atendem às suas expectativas;
- **Imagem –** a perceção do cliente sobre a empresa;
- **Preço –** o preço que a empresa está disposta a fazer e que os clientes estão dispostos a pagar.

Neste projeto, a solução desenvolvida permitirá a criação de uma base futuramente a ser utilizada pelos elementos da comunidade surda ou por outras pessoas com interação com estas, permitindo-lhes serem facilmente interpretadas e de forma correta quando pretenderem comunicar com alguém que não possui conhecimentos de língua gestual portuguesa. Com isto, é possível reduzir algumas das dificuldades enfrentadas pelas pessoas surdas na sua forma primária de comunicação, promovendo a sua inclusão e abrindo novas oportunidades nas diversas áreas da sociedade.

3.2.1 Proposta de Valor

Uma solução tecnológica direcionada para a comunidade surda, criando uma base que lhes permita, no futuro, serem compreendidas sempre que comunicarem com recurso à Língua Gestual Portuguesa. Funciona através do recurso a vídeo e da captura das mãos, interpretando as diversas configurações de mão realizadas e convertendo-as para texto. A solução deve ser capaz de dar resposta em tempo-real e com a eficácia esperada. Com esta, espera-se que a comunidade surda seja capaz de ultrapassar as dificuldades sentidas durante o processo de comunicação, podendo ser interpretadas por pessoas sem conhecimentos de Língua Gestual Portuguesa e apoiando-as em situações do dia-a-dia, como consultas, compras ou emergências.

3.2.2 Modelo de negócio CANVAS

O modelo de negócio Canvas foi proposto por Alexander Osterwalder e Yves Pigneur, e atualmente conta com 45 adaptações para diferentes países (Osterwalder et al., 2010).

O modelo é representado por uma tabela que descreve a proposta de valor, a infraestrutura, os clientes e as finanças de uma empresa ou produto. Assim, este modelo é dividido em nove elementos:

- Parceiros Chave;
- Atividades Chave;
- Recursos Chave;
- Proposta de Valor;
- Relações com o Cliente;
- Canais;
- Segmentos de Clientes;
- Custos;
- Receitas.

A Tabela 2 representa o modelo Canvas para este projeto.

Tabela 2 - Modelo Canvas

<table>
<tr><th>Parceiros Chave</th><th colspan="2">Atividades Chave</th><th>Proposta de Valor</th><th>Relações com Cliente</th><th>Segmentos de Clientes</th></tr>
<tr><td rowspan="4">Associação Portuguesa de Surdos;

Federação Portuguesa das Associações de Surdos;

Surdos e tradutores e outros elementos que contribuam.</td><td colspan="2">Desenvolvimento de solução intuitiva e fácil de utilizar (tendo em conta a quem se direciona);

Constante melhoria do serviço tendo em conta o feedback dos utilizadores;

Expansão para outras línguas gestuais;

Implementação de novas funcionalidades adequadas.</td><td>Solução de apoio à comunicação entre elementos da comunidade surda e restantes comunicadores;

Melhorias na comunicação dos surdos, promovendo a sua integração na sociedade;

Redução da necessidade de recorrer a tradutores.</td><td>Formações para a utilização da solução desenvolvida;

Apoio personalizado ao utilizador.</td><td rowspan="4">Comunidade surdas;

Quem comunica com pessoas surdas.</td></tr>
<tr><td colspan="2">Recursos Chave</td><td>Canais</td></tr>
<tr><td>Programador (desenvolvimento da solução pretendida e potenciais melhorias no futuro);

Especialista de língua gestual portuguesa (para a interpretação de imagens e criação de datasets);</td><td>*Datasets*;

Hardware (necessário para implementação e testes).</td><td rowspan="2">Associação Portuguesa de Surdos;

Federação Portuguesa das Associações de Surdos;

Encontros, congressos e eventos de surdez;

Instituições de língua gestual (escolas, faculdades, institutos, etc.).</td></tr>
<tr><td colspan="3"></td></tr>
<tr><td colspan="4">Custos</td><td colspan="2">Receitas</td></tr>
<tr><td colspan="4">Contratação de especialistas para apoio à Língua Gestual Portuguesa.</td><td colspan="2">Licenças (por exemplo, uso em contexto profissional);

Expansão para outras línguas gestuais.</td></tr>
</table>

3.3 *Analythic Hierarchy Process* (AHP)

O *Analythic Hierarchy Process* (AHP) é um método matemático, desenvolvido por Thomas Saaty, que auxilia na tomada de decisão, através de critérios qualitativos e quantitativos.

O seu resultado são diversos valores cada um correspondente a uma alternativa e, aquele que tiver mais valor, representa a melhor alternativa de acordo com os critérios e pesos definidos (Saaty, 1990).

A primeira etapa para a aplicação deste método, é a definição do objetivo, critério e alternativas:

- **Objetivo:** Escolha da linguagem de programação adequada a utilizar no projeto;
- **Critérios:**
  - **Ferramentas e Bibliotecas –** Ferramentas e bibliotecas existentes para as linguagens de programação em questão, direcionadas para visão computacional e aprendizagem máquina;
  - **Facilidade de Aprendizagem –** Facilidade de aprendizagem para as linguagens de programação em questão, considerando também a experiência com utilização da mesma noutros projetos e áreas;
  - **Performance –** Tempo de resposta e utilização de recursos;
  - **Comunidade –** Comunidade existente, capaz de solucionar problemas ou suportar certas questões na área do projeto;
- **Alternativas:** Python, Java e C/C++.

A Figura 14 representa a árvore hierárquica de decisão de acordo com o objetivo, critérios e alternativas definidas anteriormente.

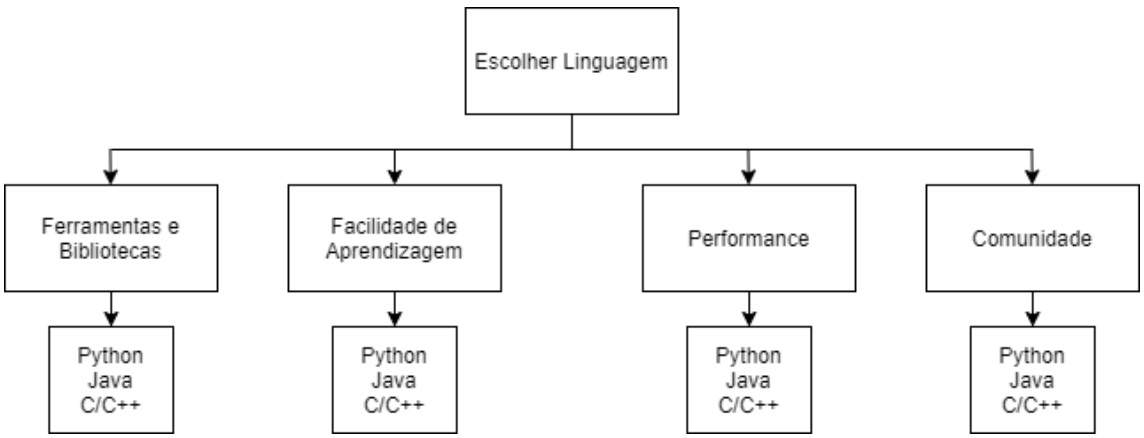

Figura 14 - Árvore Hierárquica de Decisão

Com base nos critérios definidos, foi criada a matriz de comparação dos mesmos (ver Tabela 3), estabelecendo prioridades entre os diversos elementos para cada nível.

Tabela 3 - Matriz de Comparação para os critérios

Critérios	Ferramentas e Bibliotecas	Facilidade de Aprendizagem	Performance	Comunidade
Ferramentas e Bibliotecas	1	6	3	4
Facilidade de Aprendizagem	$^1/_6$	1	$^1/_4$	$^1/_2$
Performance	$^1/_3$	4	1	3
Comunidade	$^1/_4$	2	$^1/_3$	1
Soma	$^7/_4$	13	$^{55}/_{12}$	$^{17}/_2$

Com a transformação da tabela para uma matriz, é possível obter a seguinte matriz de comparação:

$$A = \begin{bmatrix} 1 & 6 & 3 & 4 \\ 0{,}17 & 1 & 0{,}25 & 0{,}5 \\ 0{,}33 & 4 & 1 & 3 \\ 0{,}25 & 2 & 0{,}33 & 1 \end{bmatrix}$$

De seguida, foi calculada a prioridade relativa de cada um dos critérios, que permite identificar quais os critérios com maior peso entre si. Para isso, dividiu-se cada um dos valores da tabela anterior pela soma de cada uma das colunas. Posteriormente, calculou-se a média aritmética de cada uma das linhas, obtendo-se assim a respetiva prioridade relativa de cada um dos critérios (ver Tabela 4).

Tabela 4 - Matriz normalizada com prioridade relativa

Critérios	Ferramentas e Bibliotecas	Facilidade de Aprendizagem	Performance	Comunidade	Prioridade Relativa
Ferramentas e Bibliotecas	$^4/_7$	$^6/_{13}$	$^{36}/_{55}$	$^8/_{17}$	0,5395
Facilidade de Aprendizagem	$^2/_{21}$	$^1/_{13}$	$^3/_{55}$	$^1/_{17}$	0,0714
Performance	$^4/_{21}$	$^4/_{13}$	$^{12}/_{55}$	$^6/_{17}$	0,2673
Comunidade	$^1/_7$	$^2/_{13}$	$^4/_{55}$	$^2/_{17}$	0,1218

Com a análise da prioridade relativa, podemos concluir que, de acordo com os pesos inseridos, os critérios ordenados do mais para o menos importante são os seguintes: ferramentas e bibliotecas (0,54), performance (0,27), comunidade (0,12) e facilidade de aprendizagem (0,07). Assim, a matriz de comparação, a matiz normalizada e o vetor prioridades, ou vetor próprio, são os seguintes:

$$A = \begin{bmatrix} 1 & 6 & 3 & 4 \\ 0{,}17 & 1 & 0{,}25 & 0{,}5 \\ 0{,}33 & 4 & 1 & 3 \\ 0{,}25 & 2 & 0{,}33 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} 0{,}57 & 0{,}46 & 0{,}65 & 0{,}47 \\ 0{,}10 & 0{,}08 & 0{,}06 & 0{,}06 \\ 0{,}19 & 0{,}31 & 0{,}22 & 0{,}35 \\ 0{,}14 & 0{,}15 & 0{,}07 & 0{,}12 \end{bmatrix} \longrightarrow X = \begin{bmatrix} 0{,}54 \\ 0{,}07 \\ 0{,}27 \\ 0{,}12 \end{bmatrix}$$

Tendo sido calcular a prioridade relativa, é necessário avaliar a consistência das prioridades relativas. Para isso, é calculada a Razão de Consistência (RC), que permite medir quanto os julgamentos foram consistentes em relação a grandes amostras de juízos completamente aleatórios. Primeiro é necessário calcular o valor próprio (γ_{max}). Esta etapa pode ser representada através da seguinte fórmula:

$$A \times X = \gamma_{max} \begin{bmatrix} 0{,}54 \\ 0{,}07 \\ 0{,}27 \\ 0{,}12 \end{bmatrix}$$

Inicialmente, multiplica-se as matrizes A e X, definidas anteriormente, obtendo-se o seguinte resultado:

$$\begin{bmatrix} 2{,}25 \\ 0{,}29 \\ 1{,}09 \\ 0{,}48 \end{bmatrix} = \gamma_{max} \begin{bmatrix} 0{,}54 \\ 0{,}07 \\ 0{,}27 \\ 0{,}12 \end{bmatrix}$$

Com o resultado da multiplicação das matrizes A e X, efetuou-se a média da divisão de cada uma das linhas, obtendo-se o valor de γ_{max}:

$$\gamma_{max} = average\{2{,}25 \div 0{,}54;\ 0{,}29 \div 0{,}07; 1{,}09 \div 0{,}27; 0{,}48 \div 0{,}12\} = 4{,}09$$

Com o valor de γ_{max}, foi possível calcular o Índice de Consistência (IC):

$$IC = \frac{\gamma_{max} - n}{n - 1} = \frac{4{,}09 - 4}{4 - 1} = 0{,}03$$

Tabela 5 - Valores de IR para matrizes quadradas de ordem n

1	2	3	4
0	0	0,58	0,90

Com o valor de IC calculado e considerando com o valor da Tabela 5 para quatro alternativas (0,90), foi possível calcular o valor da Razão de Consistência através da seguinte fórmula:

$$RC = \frac{IC}{0{,}90} = \frac{0{,}03}{0{,}90} = 0{,}03$$

Como **0,03<0,1**, é possível concluir que os valores das prioridades relativas utilizados estão consistentes entre si.

Terminado o processo de avaliação da consistência das prioridades relativas, avançou-se para a etapa seguinte, que consiste na construção da matriz de comparação paritária para cada critério, considerando cada uma das alternativas selecionadas. As tabelas e os respetivos vetores de prioridade obtidos foram os seguintes:

Tabela 6 - Matriz de comparação, normalizada e prioridade relativa para o critério Ferramentas e Bibliotecas

Matriz de Comparação			
Ferramentas e Bibliotecas	Python	Java	C/C++
Python	1	5	3
Java	$^1/_5$	1	$^1/_3$
C/C++	$^1/_3$	3	1
Soma	$^{23}/_{15}$	9	$^{13}/_3$

Matriz Normalizada				
Ferramentas e Bibliotecas	Python	Java	C/C++	**Prioridade Relativa**
Python	$^{15}/_{23}$	$^5/_9$	$^9/_{13}$	0,6333
Java	$^3/_{23}$	$^1/_9$	$^1/_{13}$	0,1061
C/C++	$^5/_{23}$	$^1/_3$	$^3/_{13}$	0,2605

$$X_{ferramentas} = \begin{bmatrix} 0{,}63 \\ 0{,}11 \\ 0{,}26 \end{bmatrix}$$

Tabela 7 - Matriz de comparação, normalizada e prioridade relativa para o critério Facilidade de Aprendizagem

Matriz de Comparação			
Facilidade de Aprendizagem	Python	Java	C/C++
Python	1	3	5
Java	$^1/_3$	1	2
C/C++	$^1/_5$	$^1/_3$	1
Soma	$^{23}/_{15}$	$^{13}/_3$	8

Matriz Normalizada				
Facilidade de Aprendizagem	Python	Java	C/C++	**Prioridade Relativa**
Python	$^{15}/_{23}$	$^9/_{13}$	$^5/_8$	0,6565
Java	$^5/_{23}$	$^3/_{13}$	$^1/_4$	0,2327
C/C++	$^3/_{23}$	$^1/_{13}$	$^1/_8$	0,1108

$$X_{aprendizagem} = \begin{bmatrix} 0{,}66 \\ 0{,}23 \\ 0{,}11 \end{bmatrix}$$

Tabela 8 - Matriz de comparação, normalizada e prioridade relativa para o critério Performance

Matriz de Comparação			
Performance	Python	Java	C/C++
Python	1	$^1/_2$	$^1/_3$
Java	2	1	$^1/_2$
C/C++	3	2	1
Soma	6	$^7/_2$	$^{11}/_6$

Matriz Normalizada				
Performance	Python	Java	C/C++	**Prioridade Relativa**
Python	$^1/_6$	$^1/_7$	$^2/_{11}$	0,1634
Java	$^1/_3$	$^2/_7$	$^3/_{11}$	0,2972
C/C++	$^1/_2$	$^4/_7$	$^6/_{11}$	0,5390

$$X_{performance} = \begin{bmatrix} 0{,}16 \\ 0{,}30 \\ 0{,}54 \end{bmatrix}$$

Tabela 9 - Matriz de comparação, normalizada e prioridade relativa para o critério Comunidade

Matriz de Comparação			
Comunidade	Python	Java	C/C++
Python	1	5	3
Java	$^1/_5$	1	$^1/_3$
C/C++	$^1/_3$	3	1
Soma	$^{23}/_{15}$	9	$^{13}/_3$

Matriz Normalizada				
Comunidade	Python	Java	C/C++	**Prioridade Relativa**
Python	$^{15}/_{23}$	$^5/_9$	$^9/_{13}$	0,6333
Java	$^3/_{23}$	$^1/_9$	$^1/_{13}$	0,1062
C/C++	$^5/_{23}$	$^1/_3$	$^3/_{13}$	0,2605

$$X_{comunidade} = \begin{bmatrix} 0{,}63 \\ 0{,}11 \\ 0{,}26 \end{bmatrix}$$

Com os vetores de prioridades calculados, foi criada outra árvore de hierárquica de decisão:

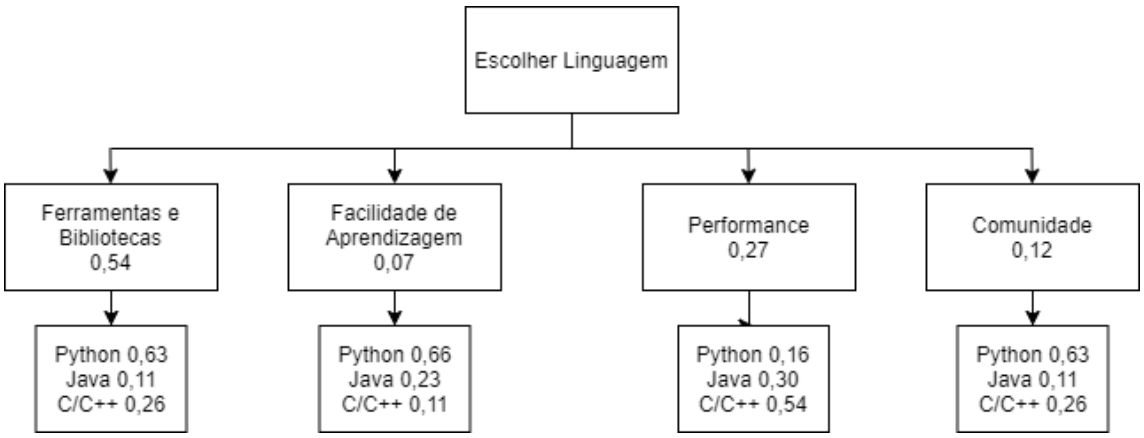

Figura 15 - Árvore Hierárquica de Decisão final

Posteriormente, multiplicou-se a matriz de prioridade com o vetor dos pesos dos critérios:

$$\begin{bmatrix} 0{,}63 & 0{,}66 & 0{,}16 & 0{,}63 \\ 0{,}11 & 0{,}23 & 0{,}30 & 0{,}11 \\ 0{,}26 & 0{,}11 & 0{,}54 & 0{,}26 \end{bmatrix} \times \begin{bmatrix} 0{,}54 \\ 0{,}07 \\ 0{,}27 \\ 0{,}12 \end{bmatrix} = \begin{bmatrix} 0{,}51 \\ 0{,}17 \\ 0{,}33 \end{bmatrix}$$

Finalmente, com o resultado obtido, foi possível concluir que a melhor alternativa é o Python (0,51), seguindo-se o C/C++ (0,33). Segundo os cálculos, o Java revelou-se a pior alternativa das três (0,17).

4 Visão da Solução

Neste capítulo é introduzida a visão da solução, apresentado ao leitor as diversas tecnologias a utilizar e uma comparação entre elas, de forma a concluir quais serão utilizadas durante o desenvolvimento do projeto. São também apresentados e comparados diferentes *datasets* identificados. Por fim, é explicada a abordagem com que se pretende avançar.

4.1 Tecnologias

A escolha das tecnologias a utilizar num projeto é um passo fundamental, influenciando todos os passos do processo de desenvolvimento. Assim, é importante escolher as melhores tecnologias dentro das que estão disponíveis e tirar o melhor proveito delas. Só desta forma se podem garantir os melhores resultados em termos de produtividade.

De seguida são apresentadas algumas tecnologias no âmbito da visão computacional e processamento de imagem, e na área de aprendizagem máquina e criação de modelos de classificação.

4.1.1 Tecnologias de Visão Computacional e Processamento Imagem

OpenCV

O OpenCV (OpenCV, 2020a) é uma biblioteca *open source* de visão computacional e aprendizagem máquina. Esta contém mais de 2500 algoritmos otimizados, que permitem detetar faces, identificar objetos, classificar ações de humanos em vídeo, extrair modelo 3D de objetos, entre outras funcionalidades.

Para além dos seus recursos, contém uma comunidade com mais de 47 000 utilizadores e com um número estimado de *downloads* superior a 18 milhões. Atualmente é utilizado por empresas com a Google, Yahoo, Microsoft, Intel, IBM, Sony, entre outras, e está presente em projetos de videovigilância em Israel ou de monitorização de minas na China (OpenCV, 2020a).

Google MediaPipe

Em 2019, a empresa americana Google, anunciou o lançamento de um rastreador de mãos desenvolvido em MediaPipe – uma *framework opensource* que permite a criação de *pipelines* para processar dados percetuais de diferentes modalidades, tais como vídeo e áudio.

O sistema desenvolvido permite a identificar e acompanhar os movimentos das mãos e dos dedos, recorrendo a técnicas de aprendizagem máquina para inferir um total de 21 pontos 3D através da análise de um único *frame*. A profundidade é representada em tons de cinzento e pelo tamanho dos pontos detetados (ver Figura 16).

Para além desta funcionalidade, a *framework* MediaPipe já contem outros sistemas de deteção com recursos a técnicas de aprendizagem máquina, desenvolvidos em Android, como a deteção e rastreamento de faces e objetos e a segmentação do cabelo de uma pessoa (Bazarevsky & Zhang, 2020).

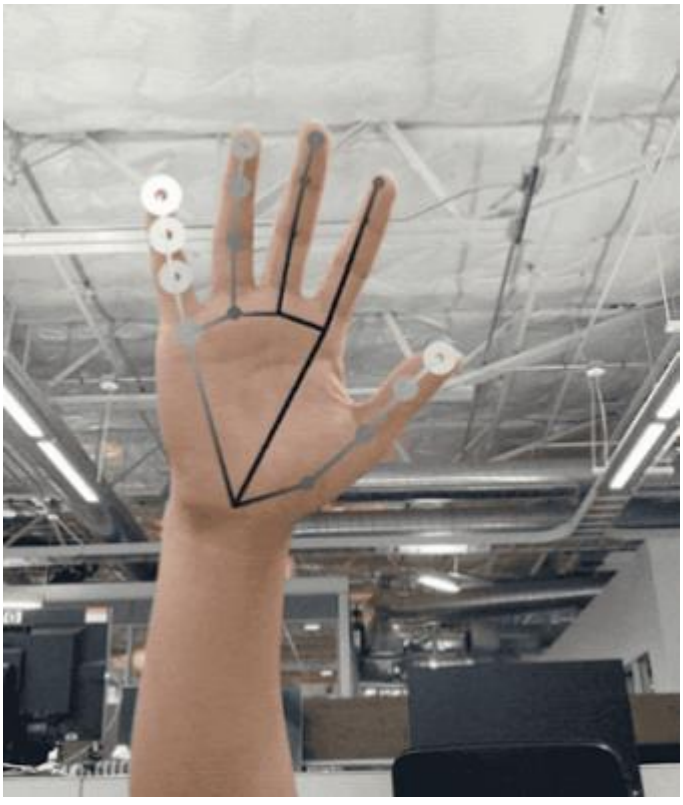

Figura 16 - Deteção 3D de uma mão utilizando MediaPipe

4.1.2 Tecnologias de Aprendizagem Máquina

TensorFlow e Tensorflow Object Detection API

TensorFlow (Tensorflow, 2020b) é um sistema de aprendizagem máquina baseado em redes neuronais, funcional em diversos ambientes e que é utilizada por múltiplos sistemas da Google. Devido à sua arquitetura, o TensorFlow permite que os programadores experimentem novas otimizações e algoritmos de treino (Abadi et al., 2016).

Os seus utilizadores podem criar grafos de fluxo de dados (*dataflow graphs*), que representam estruturas que descrevem como é que a informação se movimenta num grafo, ou uma série de nós de processamento. Cada nó do grafo representa uma operação matemática, e cada conexão entre nós é uma matriz de dados multidimensional (*tensor*) (Yegulalp, 2019). O TensorFlow mapeia os nós do grafo através de várias máquinas em *cluster*, ou através de múltiplos dispositivos computacionais como CPUs, GPUs ou unidades de processamento do Tensor (*Tensor Processing Units* – TPUs) (Abadi et al., 2016).

O Tensorflow Object Detection API é uma *framework open source* desenvolvido por cima do Tensorflow, para facilitar a construção, treino e implantação de modelos de deteção de objetos. Atualmente, esta API suporta ambas as versões 1 e 2 do Tensorflow e contém uma série de modelos pré-treinados que podem ser utilizados para treinos nas mais diversas áreas (Tensorflow, 2020b).

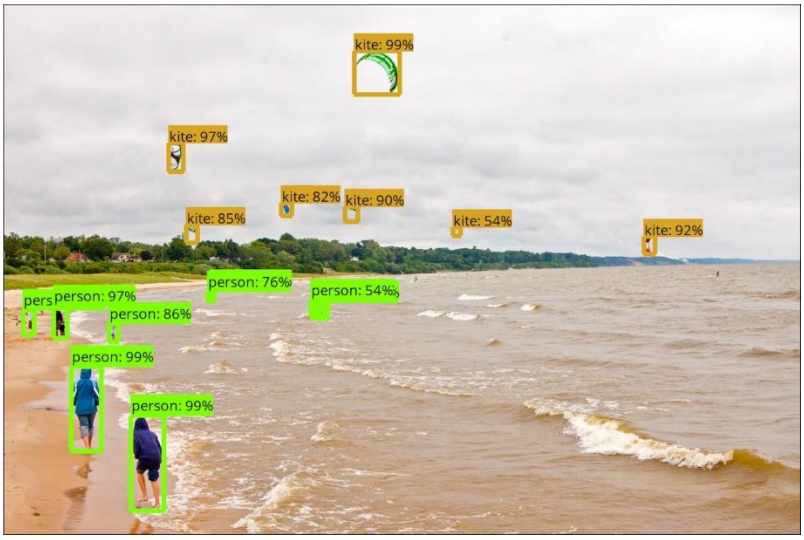

Figura 17 - Tensorflow e Tensorflow Object Detection API

Keras

O Keras é uma API de aprendizagem máquina e *deep learning* desenvolvida em Python, e que corre no topo do Tensorflow.

Assim, juntamente com o Tensorflow, o Keras torna-se uma API muito poderosa, altamente produtiva para problemas de aprendizagem máquina, em particular de *deep learning*. Esta contém abstrações e funcionalidades para o desenvolvimento e criação de soluções de aprendizagem máquina com alta velocidade de iteração (Chollet & &, 2020).

PyTorch

O PyTorch (Facebook, 2020) é uma *framework* de aprendizagem máquina, sendo uma alternativa ao Tensorflow. Este permite experiências rápidas e flexíveis experiências através de um *front-end* de fácil utilização, treino distribuído e um ecossistema extenso de ferramentas e bibliotecas. É também recomendado para *deep learning*, possibilitando a utilização tanto do CPU como GPU.

Scikit-Learn

O Scikit-Learn (Scikit-Learn, 2020b) é uma biblioteca de aprendizagem máquina *open source* desenvolvida para Python, que fornece várias ferramentas para a criação de modelos, pré-processamento de dados, seleção e avaliação de modelos e respetivas métricas, entre outras. Para além disso, suporta aprendizagem supervisionada e não supervisionada.

4.1.3 Outras tecnologias e ferramentas

Python

Python é uma linguagem de programação orientada a objetos e atualmente é considerada uma das linguagens de programação mais populares, devido à sua sintaxe e à sua reputação de produtividade (Venners, 2013).

São diversas as aplicações populares que recorrem às capacidades do Python para implementar as suas funcionalidades como o Instagram, Spotify, Dropbox, Uber, Reddit, entre outras (Django Starts, 2019).

Esta linguagem teve um grande crescimento nos últimos anos e, de acordo com dados de 2016, na área de aprendizagem máquina era a mais procurada, superando linguagens como Java, C ou C++ (ver Figura 18). Algumas das razões para este sucesso são as inúmeras bibliotecas disponíveis, a sua sintaxe simples e intuitiva, a sua flexibilidade ou a extensa comunidade (Luashchuk, 2019).

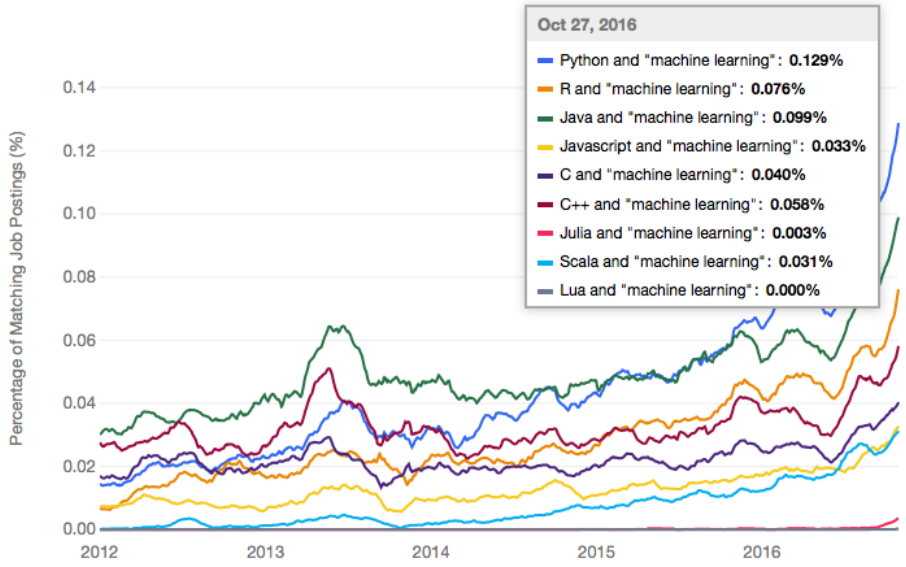

Figura 18 - Linguagens de Programação mais procuradas, de acordo com o Indeed (Puget, 2016)

4.1.4 Comparação das Tecnologias

Tal como analisado anteriormente (ver 3.3 *Analythic Hierarchy Process* (AHP)), a linguagem de programação Python revelou-se a mais interessante para ser utilizada neste projeto, considerando os parâmetros de ferramentas e bibliotecas, facilidade de aprendizagem, performance e comunidade. Uma das vantagens do Python é que atualmente é uma linguagem muito utilizada para projetos de inteligência artificial e aprendizagem máquina, possuindo um conjunto extenso de bibliotecas úteis que podem ser importadas.

Relativamente à visão computacional, o Google MediaPipe revela-se uma ferramenta muito interessante, possibilitando a leitura de vídeo e consequente identificação de mãos e dos seus respetivos pontos. No entanto, esta funcionalidade é relativamente recente e a comunidade com conhecimentos sobre esta ferramenta no geral ainda parece curta, não havendo muitos tópicos de discussão ou de aprendizagem. Por outro lado, o OpenCV possui uma extensa comunidade, uma documentação muito completa e um conjunto muito extenso de funcionalidades ao nível de processamento de imagem. Ao contrário do MediaPipe, o OpenCV não consegue identificar automaticamente mãos, mas integrado com outras

ferramentas de aprendizagem máquina e modelos de classificação, este tipo de deteção também pode ser possível. Como tal, o OpenCV revela-se a melhor solução neste aspeto e o MediaPipe, apesar de ser interessante, foi uma hipótese descartada.

Relativamente às tecnologias de aprendizagem máquina, existe uma concorrência interessente entre o Tensorflow e o PyTorch. Ambas apresentam uma boa comunidade e documentação, com provas dadas no campo da performance e treino de modelos. A decisão neste componente do projeto, acabou por recair no Tensorflow devido às diversas integrações que tem e que o tornam ainda uma ferramenta mais poderosa. Por exemplo, o Keras é uma API desenvolvida sob o Tensorflow e que permite a manipulação de algoritmos de *deep learning*, como redes neuronais. O Tensorflow é também uma tecnologia com mais tempo no mercado e que suporta diversas versões do Python, enquanto que o PyTorch apenas suporta Python a partir da versão 3. Por fim, o Tensorflow tem o Tensorflow Object Detection API, que permite a criação de modelos para a deteção de objetos e que vai permitir identificar a posição dos gestos no momento de captura do vídeo (ver Tabela 10).

Tabela 10 - Comparação de alternativas de tecnologias

Área de aplicação	Tecnologias	Funcionalidades	Decisão
Visão Computacional	**OpenCV**	+ inúmeras funcionalidades + extensa comunidade + documentação - deteção automática de objetos	**OpenCV**
	MediaPipe	+ deteção automática de mãos + deteção de pontos e profundidade - recursos inúteis ao projeto - comunidade pequena	
Aprendizagem Máquina	**Tensorflow**	+ excelente documentação oficial + integração com diversas bibliotecas + boa performance + Tensorflow Object Detection API	**Tensorflow**
	PyTorch	+ documentação oficial + funcionalidades - versões do Python suportadas	

4.2 Datasets

De modo a treinar modelos de classificação, é necessário recorrer a um conjunto de dados. Desde o início do projeto que se identificou como uma das restrições a existência de um *dataset* adequado para ser utilizado na classificação do gesto, devido à inexistência dos mesmos para a língua gestual portuguesa. No entanto, para a vertente de visão computacional deste projeto, ou seja, para a deteção da presença da mão e a sua extração através de um vídeo, foi possível identificar *datasets* úteis e passíveis de serem utilizados para a criação do modelo.

De seguida, são apresentados dois conjuntos de dados encontrados e é feita uma comparação entre ambos.

4.2.1 Egohands Dataset

O Egohands Dataset (Indiana University, 2020), foi criado pela Universidade de Indiana e é composto por um total de 48 vídeos representativos da interação entre duas pessoas e em diferentes cenários. Foram filmados com uns Google Glass, uns óculos inteligentes desenvolvidos pela empresa americana Google.

Este *dataset* está disponível de três formas distintas:

- **Dados rotulados (*labeled data*)** – constituído por 100 imagens rotuladas por cada um dos 48 vídeos, totalizando 4800 imagens com informação da localização das mãos.
- **Ficheiros de vídeo** – constituído por um total de 48 vídeos em formato MP4 (h264). Cada vídeo tem uma duração de 90 segundos e uma resolução de 720x1280px (píxeis) a 30fps (*frames* por segundo).
- **Todos os *frames*** – constituído por todos os frames, ou seja, 2700 *frames* por cada um dos 48 vídeos.

No contexto deste projeto, a primeira opção é a que faz mais sentido, proporcionando, num ficheiro MatLab, os dados já rotulados, isto é, com as informações relativas à localização da mão em cada uma das imagens, e prontos a serem usados para a criação do modelo.

Relativamente aos dados do *dataset*, este contém frames de quatro jogos diferentes de dois jogadores (cartas, xadrez, jenga e puzzle) em três localizações diferentes (pático, sala de estar e escritório). Cada combinação destas constitui uma pasta com 100 imagens diferentes e um ficheiro *.mat* (matlab) com a informação das mãos.

Este *dataset* já foi utilizado em projetos desenvolvidos para a área de visão computacional e, mais especificamente, de deteção de mãos, apresentando resultados positivos e promissores para este projeto em particular (Dibia, 2019).

A Figura 19 representa um exemplo de uma das imagens do *Egohands Dataset*, tirada durante um dos jogos de cartas no pátio. Devido ao facto de as fotos terem sido tiradas em diferentes ações e locais, estas podem apresentar diferentes condições de iluminação e ambiente.

Figura 19 - Exemplo de imagem do Egohands Dataset

4.2.2 Oxford Hand Dataset

O *Oxford Hand Dataset* (University of Oxford, 2014), desenvolvido pelo Visual Geometry Group da Universidade de Oxford, é outro conjunto de dados existem para problemas de deteção de mão, tendo um total de 13050 anotações de mãos.

Este *dataset* é constituído por três conjuntos de dados diferentes: treino, avaliação e teste. As imagens que compõem cada um destes conjuntos são retiradas de diferentes fontes, como de séries, ou de outros *datasets* já existentes. Por exemplo, são utilizadas imagens do *Inria pedestrian*, um *dataset* com imagens de pessoas.

Ao contrário do *dataset* anterior, que tinha um ficheiro *matlab* com as anotações para cada uma das ações e locais, o *Oxford Hand Dataset* contém um ficheiro *matlab* por imagem. Ou seja, o número de imagens e o de ficheiros de anotações será sempre o mesmo.

O conjunto de dados de treino contém cerca de 4069 imagens, o de teste tem 821 e, por fim, o de validação contém 738 imagens.

Figura 20 - Exemplo de imagem do Oxford Hands Dataset

4.2.3 Comparação dos Datasets

Os dois *datasets* apresentam um conjunto de dados significativo e estão devidamente anotados, evitando assim o processo de anotação, que consiste em selecionar para cada uma das imagens o local onde se encontra a mão e que, para uns *datasets* tão extensos, se revelaria uma tarefa demorada e trabalhosa.

Assim, uma das diferenças entre os dois *datasets* está na qualidade dos dados. Após a análise dos dois, detetou-se que o *Egohands Dataset* continha imagens com mais qualidade e resolução, com mais luminosidade e contraste. Devido à baixa dimensão e qualidade de algumas imagens do *Oxford Hands Dataset* nem sempre é fácil identificar a localização das mãos o que pode ter um impacto significativo no momento do treino dos modelos de deteção.

Para além disso, os cenários representados no *Egohands Dataset* assemelham-se mais com o contexto deste projeto comparativamente ao outro *dataset*, em que vários das imagens são de pessoas a uma distância considerável da câmara e, consequentemente, com as mãos muito pequenas. Com isto, em muitas das imagens a forma, contornos e caraterísticas das mãos são praticamente irreconhecíveis. Para além disto, também foram identificadas algumas

imagens desfocadas, principalmente na parte da mão. Desvantagens grandes tendo em conta que neste projeto a mão é a principal fonte de informação.

Concluindo, e após esta análise dos dados de ambos os *datasets*, concluiu-se que o *Egohands Dataset* é o melhor conjunto de dados para o projeto em questão, permitindo obter melhores resultados no momento do treino e, consequentemente, no momento da deteção.

Figura 21 - Lado a lado de imagens de ambos os Datasets

4.3 Abordagem

Inicialmente, foi necessário analisar o problema, compreendendo-o e identificando os objetivos e a possível abordagem para resolver o mesmo. Tratando-se de um problema com pouco destaque e que a maioria dos cidadãos comuns nunca pensou, foi necessário uma boa discussão e organização de ideias.

A análise de soluções existentes nos capítulos anteriores (ver 2.2 Estado da Arte), revelou que estas soluções de tradução de língua gestual para as pessoas surdas ainda estão pouco desenvolvidas. Em mercado, apenas foram identificadas duas soluções, SignAll e MyoSign, sendo que a primeira necessita de recorrer a três câmaras RGB e um Kinect, e a segunda recorre a uma pulseira para detetar gestos e movimentos. Foi também identificado o VirtualSign que, apesar de não se encontrar no mercado, já possui resultados interessantes. De resto, foram encontrados pequenos projetos realizados em universidades, desenvolvidos

apenas para pequenas experiências de tradução de simples gestos, como a numeração ou alfabeto, e não considerando toda a extensão da língua gestual.

Apesar da análise do estado da arte não ter revelado com grande detalhe possíveis soluções e abordagens para o problema em questão, permitiu identificar tecnologias e algoritmos que podem ser usadas para o desenvolvimento do projeto e que podem ser integradas entre si para um resultado interessante.

Com base nas fases anteriores, é possível identificar possíveis abordagens e planear as fases seguintes, mais propriamente a fase de implementação. Considerando a falta de experiência e de conhecimento com a linguagem de programação Python, é também necessário considerar algum tempo para exploração das suas funcionalidades e capacidades, para posteriormente ser utilizada com as tecnologias identificadas. Por outro lado, com falta de suporte para a Língua Gestual Portuguesa, é necessária a criação de um *dataset* com configurações de mão para posterior uso.

Posteriormente, a fase de implementação contempla todo o desenvolvimento de código, testando diferentes soluções e tecnologias com o objetivo de desenvolver o melhor código possível e com eficácia nas resoluções do problema identificado. Pode-se considerar duas fases distintas no projeto: a identificação das mãos, com recurso a tecnologias de visão computacional e modelos de deteção de objetos, e a de classificação, que consiste na tradução do gesto da Língua Gestual Portuguesa para texto.

Existem dois tipos de classificadores: binários ou multicritério. Os binários aplicam-se em casos que o resultado da classificação só pode conter dois resultados, enquanto que na classificação multicritério ou multi-classe o resultado pode ser um de múltiplas classes diferentes, como é o caso deste projeto.

Por fim, a solução desenvolvida deve ser testada, os resultados analisados e retiradas as necessárias conclusões, identificando se os objetivos inicialmente proposto foram cumpridos com sucesso.

5 Análise e Desenho

Neste capítulo são apresentados quais os requisitos funcionais e não funcionais que a solução desenvolvida deverá respeitar. A abordagem a tomar é aprofundada e é explicada ao leitor quais os processos que a solução deve efetuar para, no fim, conseguir traduzir o gesto da Língua Gestual Portuguesa (*input*) para texto (*output*).

5.1 Análise

A solução desenvolvida deve ser capaz de, através de um vídeo com conteúdo de Língua Gestual Portuguesa, traduzir o gesto para língua portuguesa escrita.

5.1.1 Requisitos

Requisitos funcionais

Relativamente aos requisitos funcionais, esse o principal caso de uso do sistema, aquele que acrescenta valor ao utilizador, é a tradução de gestos da Língua Gestual Portuguesa para texto. Para isso, é necessário tratar a informação inicial (*input*), de modo a ser corretamente classificada.

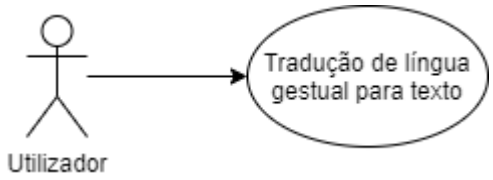

Figura 22 - Diagrama de Casos de Uso

Requisitos não funcionais

Relativamente aos requisitos não funcionais, é de extrema importância que os resultados apresentados tenham uma boa taxa de acerto, ou seja, eficácia. Caso contrário, considerando o utilizador alvo e o objetivo da solução, uma má tradução pode levar a males entendidos e à deturpação da mensagem transmitida pela pessoa surda. Também é importante que o tempo de resposta seja baixo, de forma a possibilitar no futuro a sua utilização durante diálogos.

5.2 Arquitetura

De acordo com o artigo *Visual Interpretation of Hand Gestures for Human-Computer Interaction* (Pavlovic et al., 1997), publicado em 1997, existem diferentes modelos de mão que podem ser utilizados para identificar a mesma configuração de mão:

- ***3D Textured volumetric model*** – representação 3D, com textura e sombras para representar diferentes profundidas (ver Figura 23a);
- ***3D wireframe volumetric model*** – representação 3D, com recurso ao modelo *wireframe* (ver Figura 23b);
- ***3D skeletal model*** – representação 3D baseada no esqueleto humano, mais propriamente nas mecânicas e morfologias da mão (ver Figura 23c);
- ***Binary silhouette*** – representação 2D binária (ver Figura 23d);
- ***Contour*** – representação 2D dos contornos da mão (ver Figura 23e).

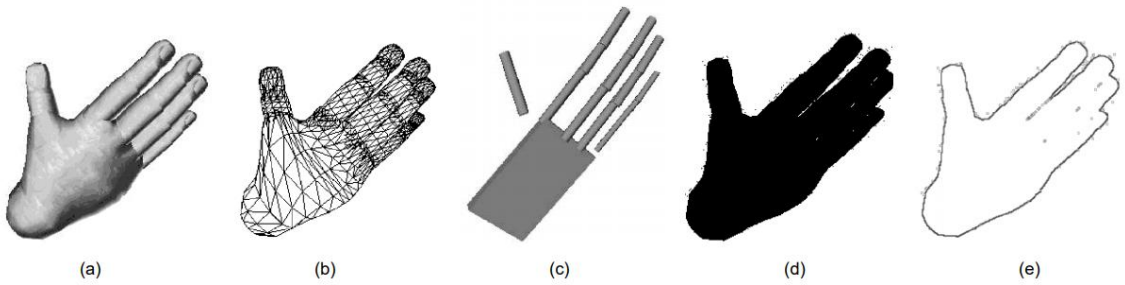

Figura 23 - Diferentes modelos de mão (Pavlovic et al., 1997)

No entanto, tal como analisado anteriormente, atualmente existem um conjunto de ferramentas, tecnologias ou bibliotecas que permitem efetuar a deteção de mão e a respetiva configuração dos dedos, através de modelos devidamente treinados para esse fim. Assim, um dos objetivos deste projeto é explorar diferentes modelos, criados através do uso de diversos algoritmos e analisar quais apresentam melhores resultados para os dados em questão.

A Figura 25 apresenta um esquema das diferentes etapas que a solução realizará a fim de apresentar ao utilizador a descrição do gesto. Tratando-se de um vídeo, o primeiro passo será extrair o *frame* para este ser processado e ser identificada a mão. Caso a mão seja identificada, será necessário transformar a imagem, de modo a facilitar a etapa de classificação e tornar os dados uniformes (ver Figura 24).

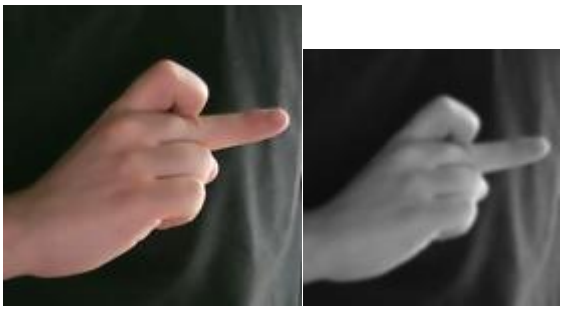

Figura 24 - Processo de tratamento e transformação

A Figura 24 demonstra as diferentes etapas de análise ao vídeo. Inicialmente, é necessário identificar a mão, através do Tensorflow Object Detection API e um modelo devidamente criado, e extraí-la num formato quadrado, permitindo a abstração dos restantes elementos da imagem. Depois, esta é processada por uma questão de uniformização, isto é, para tornar a imagem idêntica aos dados com que o modelo de classificação foi treinado. Por fim, com base em algoritmos de classificação, é apresentada ao utilizador a classificação do gesto.

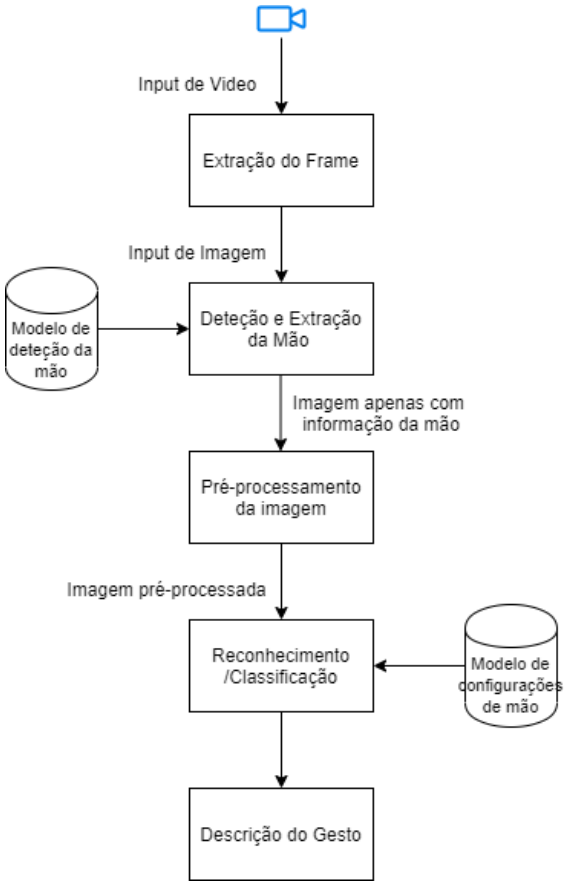

Figura 25 - Fluxo da solução desenvolvida

6 Construção da Solução

Neste capítulo é apresentado a componente prática do trabalho. Este está dividido pelas diferentes etapas de desenvolvimento desde a deteção da mão, à classificação do respetivo gesto.

6.1 Deteção da mão

A deteção da mão trata-se da etapa base, fundamental para o sucesso dos passos seguintes.

Recorrendo a um dispositivo de entrada, como uma *webcam*, é capturado um vídeo, que é constituído por vários *frames*, isto é, imagens. Para cada uma destas imagens, é necessário identificar a mão, ou até ambas as mãos, e recortar da restante imagem, mantendo desta forma apenas a informação necessária. A imagem com a mão é de seguida arquivada, com um nome apropriado. No caso de ambas as mãos serem identificadas, direita e esquerda, são arquivadas ambas com a referência de acordo com o lado a que pertencem.

De seguida são abordadas as diferentes fases do desenvolvimento desta funcionalidade de visão computacional como:

- **Procura de Dataset –** apresentação e comparação dos diferentes conjuntos de dados encontrados (*datasets*);
- **Processamento do Dataset –** transformação do *dataset* para ser interpretado pelos algoritmos de treino;
- **Criação dos Modelos –** utilização do Tensorflow Object Detection API para interpretar o dataset e criar o modelo de classificação;
- **Captação, Identificação e Resultado –** apresentação do resultado obtido com a criação dos modelos.

É importante mencionar que para o desenvolvimento desta funcionalidade utilizou-se como referência o artigo publicado pelo Victor Dibia (Dibia, 2019) sobre deteção de mãos usando o Tensorflow Object Detection API, e respetiva proposta de melhoria do J.K. Jung (Jung, 2020). Estes trabalhos, para além de terem o código aberto (*open source*) a todos os interessados, apresenta resultados interessantes na funcionalidade de deteção de mão.

6.1.1 Procura de Dataset

Para a deteção correta da mão, é fundamental a utilização de um *dataset* extenso e com qualidade, aumentando assim a eficácia dos modelos criados. O ideal seria a criação de um *dataset* próprio, de acordo com as necessidades do projeto e incluindo diferentes tons de pele, luminosidades e cenários, mas, considerando a extensão do *dataset* necessária e também as falta de condições para a criação do mesmo, esta hipótese foi descartada.

Tal como referido anteriormente, após alguma pesquisa, identificaram-se dois *datasets* diferentes, de acesso livre e com elevada extensão. No entanto, após a análise dos dados de ambos e respetiva comparação, identificou-se que o *Egohands Hands Dataset* da Universidade de Indiana (Indiana University, 2020) era o melhor para o projeto em questão.

6.1.2 Processamento do Dataset

Como primeiro passo, foi necessário organizar os dados do dataset para serem interpretados pelo *Tensorflow Object Detection API*. Esta API suporta várias formas de organização dos dados, como Kitti, Ava, Coco, Pascal, entre outros. Estes formatos são posteriormente convertidos para TFRecords, um formato simples do Tensorflow para armazenar uma sequência de registos binários (TensorFlow, 2020).

Neste caso, optou-se por converter os dados do ficheiro matlab para o formato Kitti (Geiger et al., 2012). Este formato é dividido em duas pastas:

- **labels** – constituída por um ficheiro de texto (.txt) para cada uma das imagens, fazendo um total de 4800 ficheiros. Cada um destes ficheiros tem como nome o mesmo da imagem a que se refere;
- **images** – constituída por todas as imagens do dataset (4800 imagens).

Cada um dos ficheiros de texto, contém a informação relativa à localização das mãos na imagem, retirada do ficheiro matLab. Estes dados são organizados num total de 16 colunas:

- 1ª coluna – descreve o tipo de entidade/objeto – neste caso, uma mão;
- 2ª coluna – 0 (*non-truncated*) ou 1 (*truncated*), em que 1 refere-se ao objeto a ultrapassar os limites da imagem;
- 3ª coluna – 0 (totalmente visível), 1 (parcialmente obstruído), 2 (maioritariamente obstruído) ou 3 (desconhecido), indicando o estado da oclusão;
- 4ª coluna – ângulo de observação do objeto;
- 5ª, 6ª, 7ª e 8ª coluna – coordenadas da *bounding box*, isto é, a caixa delimitadora, do objeto;
- 9ª, 10ª e 11ª coluna – dimensões 3D do objeto;

- 12ª, 13ª e 14ª coluna – localização 3D do objeto;
- 15ª coluna – rotação em torno do eixo do Y;
- 16ª coluna – apenas para resultados.

As coordenadas da *bounding box* são retiradas dos ficheiros matlab do *dataset* e, de resto, todas as colunas têm o valor 0 pois não se aplica nos dados em questão. A primeira coluna é representada através do valor *hand* (mão), por se tratar do "objeto" em questão.

Tendo o conhecimento destas informações, foi necessário iterar por cada uma das pastas do *dataset*, retirar as coordenadas do ficheiro matlab e criar um ficheiro de texto para cada uma das imagens com estes dados. O código representado na Figura 27, demonstra como são criados os ficheiros de texto com o nome das imagens e a criação do ficheiro de texto de acordo com o formato Kitti.

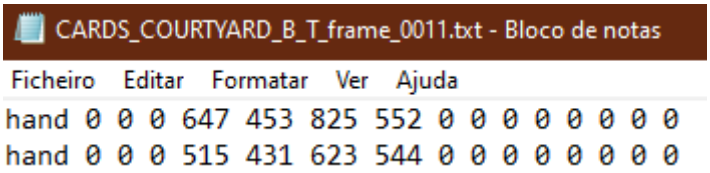

Figura 26 - Exemplo de ficheiro no formato Kitti

```
1.  def convert_folder(folder):
2.      print('Converting %s folder to Kitti format' % folder)
3.      folder_path = os.path.join(EGOHANDS_DOWNLOAD_DATA_DIR, folder)
4.      frames = [os.path.splitext(f)[0]
5.          for f in os.listdir(folder_path) if f.endswith('jpg')]
6.
7.      video = loadmat(os.path.join(folder_path, 'polygons.mat'))
8.      polygons = video['polygons'][0]
9.      for i, frame in enumerate(frames):
10.         # copy and rename jpg file to the 'converted' folder
11.         src_image = frame + '.jpg'
12.         dst_image = folder + '_' + src_image
13.         if not os.path.exists(os.path.join(PROCESSED_IMAGES_DIR, dst_image)):
14.             print('Copying file to %s' % dst_image)
15.             copyfile(os.path.join(folder_path, src_image),
16.                 os.path.join(PROCESSED_IMAGES_DIR, dst_image))
17.         dst_txt = folder + '_' + frame + '.txt'
18.         boxes = []
19.         with open(os.path.join(PROCESSED_LABELS_DIR, dst_txt), 'w') as f:
20.             for polygon in polygons[i]:
21.                 box = polygon_to_box(polygon)
22.                 if box:
23.                     boxes.append(box)
24.                     f.write(box_to_line(box) + '\n')
```

Figura 27 – Código que prepara o formato Kitti

O passo seguinte foi criar um ficheiro Protobuf, uma biblioteca eficaz na serialização de dados estruturados, com a informação sobre as classes. Neste caso apenas temos uma classe, a mão. O ficheiro tem a extensão *.pbtxt*.

```
item {
  id: 1
  name: 'hand'
}
```

Figura 28 - Ficheiro protobuf criado

Por fim, com os dados organizados no formato Kitti e o ficheiro protobuf criado, foram gerados os TFRecords, ficheiros com sequências de registos binários sobre os dados, necessários para a criação do modelo. Para isso utilizou-se um *script* próprio do Tensorflow Object Detection API, *create_kitti_tf_record.py*, que com base no formato Kitti e no ficheiro protobuf, cria os ficheiros TFRecords.

```
#!/bin/bash
TF_RECORD_SCRIPT_PATH="models/research/object_detection/dataset_tools"
mkdir data
python3 create_kitti_tf_record.py \
        --data_dir=egohands_kitti_formatted \
        --output_path=data2/egohands2 \
        --classes_to_use=hand \
        --label_map_path=data2/egohands_label_map.pbtxt \
        --validation_set_size=500
```

Figura 29 - Script criado para executar o create_kitti_tf_record.py

6.1.3 Criação dos modelos

Quando se treina uma rede neuronal, as camadas iniciais da rede identificam as propriedades mais simples, como, por exemplo, linhas retas ou oblíquas no caso da primeira camada. À medida que as camadas vão aprofundando, é possível identificar propriedades mais sofisticadas. Por exemplo, a camada 2 é capaz de identificar formas geométricas, como círculos ou quadrados. As camadas seguintes já são capazes de efetuar identificações de maior complexidade e, as camadas mais profundas, podem identificar objetos e caraterísticas, como cães, gatos ou, no caso concreto deste projeto, mãos. Assim, as camadas iniciais de um modelo, terão sempre de identificar linhas, formas geométricas ou padrões, independentemente do tipo

de modelo que se esteja a criar. O que distingue principalmente uns modelos dos outros são as camadas finais, que contêm a informação sobre o que se pretende identificar (Vasani, 2019).

O Tensorflow Object Detection API providencia vários modelos pré-treinados em diferentes *datasets* – COCO, Kitti, AVA, entre outros (Tensorflow, 2020b). Um modelo pré-treinado já contém a informação necessária sobre as primeiras camadas, restando ser treinado para as camadas mais profundas. Sendo assim, optou-se por utilizar um destes modelos pré-treinados, eliminando a necessidade de treinar camadas mais superficiais e reduzindo ao tempo e à complexidade de treino.

Para cada um dos modelos pré-treinados do Tensorflow, é indicado a velocidade do modelo e a precisão média (mAP – *median Average Precision*). Como são disponibilizados dezenas de modelos, teve de se analisar estes indicares dos diversos modelos e optou-se por realizar o treino para três deles. Os dois primeiros modelos representados na Tabela 11 foram selecionados pela sua alta velocidade e por serem dos mais comuns em termos de utilização. O último modelo foi escolhido para efeitos de comparação e para perceber se sua lentidão teria impacto no âmbito deste projeto. Os dois primeiros acabaram por se revelar muito idênticos, enquanto que o terceiro demonstrou ser muito lento no momento da deteção.

Tabela 11 - Modelos pré-treinados escolhidos

Modelos pré-treinados	Velocidade (ms)	mAP[^1]
ssd_mobilenet_v1_coco	30	21
ssd_mobilenet_v2_coco	31	22
faster_rcnn_resnet50_coco	89	30

Com os estes selecionados e os respetivos ajustes de configurações de cada um, como caminhos de pastas, foram treinados os modelos. Para isso utilizou-se o Google Colab (Google Colab, 2020), um serviço de nuvem (*cloud*), próprio para computação de inteligência artificial e aprendizagem máquina. Este permite utilizar placas gráficas (GPUs) da Google, o que aumenta a velocidade de processamento comparativamente a um processador.

Para correr o treino dos modelos, executou-se o script *model_main.py* do Tensorflow.

```
python3 ./models/research/object_detection/model_main.py \
        --pipeline_config_path=${PIPELINE_CONFIG_PATH} \
        --model_dir=${MODEL_DIR} \
        --num_train_steps=${NUM_TRAIN_STEPS} \
        --sample_1_of_n_eval_samples=1 \
        --alsologtostderr
```

Figura 30 - Execução do model_main.py

O resultado dos treinos é um ficheiro (*frozen_graph_inference.ph*) que representa o modelo criado e que pode ser utilizado para, neste caso, identificar mãos.

6.1.4 Captação, Identificação e Resultado

De forma aplicar o modelo treinado, foi necessário criar um script em Python para, através de uma câmara *web*, carregar o modelo e aplicar a identificação na imagem captada.

O OpenCV (OpenCV, 2020a) é uma ferramenta com diversas utilidades de visão computacional, permitindo utilizar um vídeo como dado de entrada, dividi-lo em várias imagens, dependendo do número de frames (fps) da câmara em questão, e, por fim, processar cada uma dessas imagens.

Para além de se inicializar a captação do vídeo através da classe *VideoCapture(type)*, é necessário carregar os modelos gerados anteriormente, mais propriamente, o ficheiro *frozen_graph_inference.ph*. O código da Figura 31 demonstra o código utilizado.

```
cap = cv2.VideoCapture(args.video_source)

# load detection graph
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.GraphDef()
    with tf.io.gfile.GFile(PATH_TO_FROZEN_GRAPH, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')

with detection_graph.as_default():
    with tf.Session() as sess:

        while True:
            ret, frame = cap.read()
```

```
            img = detect_image(frame, args.output, detection_graph, sess)
            cv2.imshow('hand_detection', img)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

cap.release()
cv2.destroyAllWindows()
```

Figura 31 - Captação de vídeo e leitura do modelo

A função *cap.read()* permite a leitura do *frame* naquele momento, guardando-o numa *array* de NumPy.

No código da Figura 32, são carregados os diferentes *tensors* (Tensorflow, 2020a), isto é, matrizes multidimensionais com um tipo uniforme, do modelo criado. Depois é executado o método *sess.run()*, obtendo três valores:

- ***boxes*** **–** coordenadas das *bouding boxes*, ou seja, da localização das mãos identificadas;
- ***scores*** **–** percentagem de certeza da identificação da classe;
- ***classes*** **–** classe identificada. Neste caso, a mão é a única classe disponível.

```
# define input/output tensors
image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
detection_classes = detection_graph.get_tensor_by_name('detection_classes:0')
num_detections = detection_graph.get_tensor_by_name('num_detections:0')
# run inference
boxes, scores, classes, _ = sess.run(
    [detection_boxes, detection_scores, detection_classes, num_detections],
    feed_dict={image_tensor: np.expand_dims(img, 0)})
```

Figura 32 - Carregar tensors e fazer a previsão

Com as coordernadas das *bounding boxes* detetadas é possível retirar da imagem apenas a parte da mão para posteriormente ser identificado o gesto. A distinção entre a imagem esquerda e a mão direita é feita também recorrendo a estas coordenadas, sendo que a *bounding box* com os menores valores do eixo do x considera-se ser a mão direita. Por fim, a imagem recortada, contendo apenas a mão, é guardada numa pasta destino, tendo como nomenclatura uma referência temporal e a mão referente – direita (RGT), esquerda (LFT) ou vazia se apenas tiver sido detetada uma mão.

Figura 33 - Exemplo do resultado obtido

6.2 Classificação do gesto

Com a extração da mão do vídeo inicial, é possível avançar para a etapa seguinte, que consiste na classificação do gesto. Para isso, é necessário treinar um novo modelo, com base num novo *dataset* com o diferente vocabulário pretendido, e aplicar uma previsão do gesto em causa.

De seguida são abordadas as diferentes fases do desenvolvimento desta funcionalidade de classificação computacional como:

- **Criação do *dataset* –** processo de criação do *dataset* utilizado para o treino dos modelos;
- **Pré-processamento do *dataset* –** diferentes processamentos efetuados às imagens do *dataset*, para consequentemente serem interpretados pelo algoritmo de treino;
- **Algoritmos de classificação –** os algoritmos e respetivas configurações utilizadas para a criação dos modelos;
- ***Cross Validation* e treino dos modelos –** processo de utilização dos algoritmos anteriormente referidos, para avaliar e criar os modelos de classificação;
- **Exportação dos resultados –** apresentação dos resultados da avaliação dos algoritmos;
- **Previsão** – utilização dos modelos criados para efetuar a previsão de um gesto.

6.2.1 Criação do Dataset

Tal como referido em capítulos anteriores, a língua gestual varia de acordo com o país, não sendo uma linguagem universal. Considerando isto, identificou-se desde início que uma das restrições deste projeto seria a utilização de um *dataset* de língua gestual portuguesa já existente e com dados suficientes.

Como tal, foi necessário criar de raiz um *dataset* capaz de servir de base para esta etapa de classificação. Assim, com o intuito de criar um conjunto de dados sólido e significativo, tiveram de ser considerados alguns critérios, recursos e parâmetros, como:

- **Câmara para captação das imagens –** LarmTek 1080p de 2 megapixéis (2mpx);
- **Vocabulário –** 26 letras do alfabeto: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z;
- **Tamanho do *dataset* por letra –** captação de 150 imagens por letra;
- **Tamanho do *dataset* total –** total de 1300 imagens (150 * 26);
- **Factores de mudança entre imagens da mesma letra –** com o intuito de as imagens não se tornarem demasiado idênticas entre si na captura de cada um dos gestos, utilizaram-se as seguintes distinções:
    - Ligeira rotação da mão entre diferentes capturas;
    - Planos de fundo com diferentes cores, como o azul, o branco, o castanho, o laranja e o preto;
    - Luminosidade.

Para a captura das imagens do *dataset*, utilizou-se parte do trabalho desenvolvido na etapa anterior, da captura da mão, permitindo capturar constantemente diferentes imagens, recortar os gestos e guardá-los localmente. Devido ao trabalho e lentidão que envolve a criação de um *dataset*, optou-se por apenas considerar o alfabeto, composto por 26 letras. No entanto, uma das vantagens da língua gestual é que certas expressões ou palavras envolvem letras do alfabeto, podendo aumentar assim a abrangência deste *dataset*.

Assim, para facilitar todo o processo de criação do *dataset*, criou-se um script, denominado *create_hands_dataset.py*, com duas funcionalidades principais: a captação de imagens para o *dataset* e a renomeação das imagens capturadas para uma nomenclatura uniforme. Como tal, para a execução do *script* Python, criaram-se três argumentos de entrada:

- ***Type* –** funcionalidade desejada pelo utilizador. Caso o argumento seja 0, é iniciada a captura de imagens e extração de gestos. Se o argumento for 1, são renomeadas as imagens anteriormente captadas pelo utilizador;
- ***Symbol* –** identificador da letra em questão. Por exemplo, a letra A tem o identificador 001, a letra B é representada pelo 002 e continuando a incrementação para as restantes letras. Este campo apenas é necessário caso a funcionalidade seja a de incrementar o *dataset*;
- ***Author* –** 3 letras representantes do autor das imagens. Este campo apenas é necessário caso a funcionalidade seja a de incrementar o *dataset*;
- ***Color* –** cor do fundo utilizada: BLK para preto, BLU para azul, BRO para castanho, ORA para laranja e WHT para branco.

Relativamente à primeira funcionalidade, tal como referido anteriormente, aproveitou-se a maioria do trabalho desenvolvido na etapa da captura da mão/gesto, tendo apenas sido feitas adaptações no nome destino da imagem, de forma a considerar os argumentos de entrada fornecidos pelo utilizador. Assim, no caso de o utilizador captar imagens, estas são guardadas com a nomenclatura "LGP_*symbol_author_color_timestr*_.jpg", sendo que as variáveis *symbol, author* e *color* representam os argumentos de entrada, e a variável *timestr* representa uma informação temporal no formato %Y%m%d-%H%M%S%f, isto é, ano-mês-dia-hora-minuto-segundo-milésimas de segundo (e.g. 20201231-211342324). As iniciais "LGP" (Língua Gestual Portuguesa) são apenas um indicador da linguagem em causa.

```
1.  timestr = datetime.utcnow().strftime("%Y%m%d-%H%M%S%f")[:-3]
2.
3.  (...)
4.
5.  cv2.imwrite(os.path.join(output_path, 'LGP_' + symbol + '_' + author + "_" +
     timestr + "_" + ".jpg"), save_img)
```

Figura 34 - Nomenclatura para a funcionalidade de captação

No final do processo de captação, o utilizador pode escolher quais as imagens que pretende manter no *dataset*, removendo aquelas que não são do seu interesse e, por fim, pode renomear as imagens para a nomenclatura final recorrendo à segunda funcionalidade. Esta funcionalidade vai iterar cada uma das imagens armazenadas e, para aquelas cujo nome termina por "_", vai alterar para a nomenclatura final: LGP_*symbol_author_color_id*.jpg, em que *id* representa um identificador que incrementa para a cada uma das imagens da letra, ou seja, de 1 até 150.

Figura 35 - Exemplo de capturas para a letra A, com diferentes fundos, após a funcionalidade de renomear

6.2.2 Pré processamento do Dataset

De forma a retirar o máximo de informação do *dataset* criado, aumentando assim a eficácia do treino a realizar, foram utilizados um conjunto de 8 diferentes tipos de pré-processamento, como:

- Imagens iguais às capturadas, apenas com um redimensionamento de 128x128 píxeis (ver Figura 36 alínea 1a);
- Aplicação de um desfoque gaussiano e conversão das imagens para preto, assim como um redimensionamento de 128x128 píxeis (ver Figura 36 alínea 1b);
- Deteção dos limites da imagem, sem suavização do fundo, assim como um redimensionamento de 128x128 píxeis (ver Figura 36 alínea 1c);
- Deteção dos limites da imagem, com suavização do fundo, assim como um redimensionamento de 128x128 píxeis (ver Figura 36 alínea 1c).

Para além destes, foi também aplicado um aumento de contraste antes de cada um dos pré-processamento referidos nos pontos anteriores, perfazendo um total de 8 conjuntos de dados com distinção entre si (ver Figura 36 alínea 2).

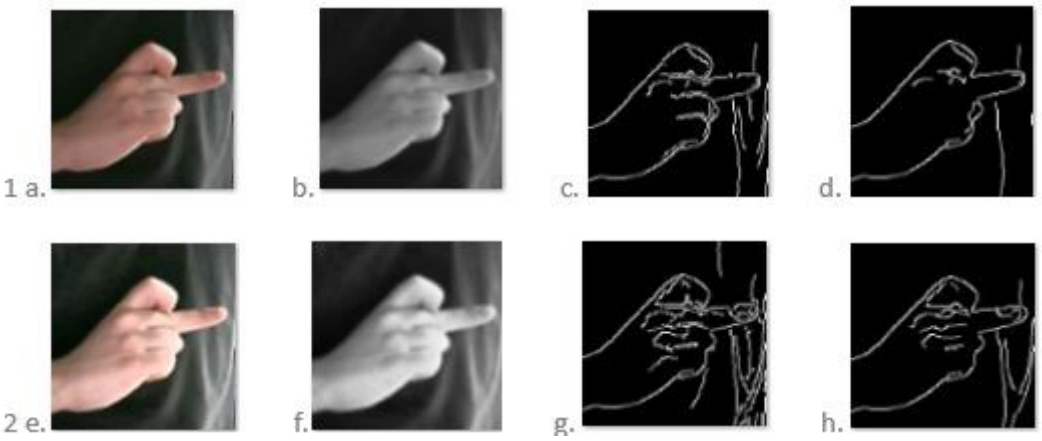

Figura 36 - Resultado dos diferentes tipos de pré-processamento aplicados

6.2.3 Configuração dos Algoritmos de Classificação

Com a criação do *dataset* e respetivo pré-processamento do mesmo, a seguinte etapa foi a de implementação e preparação dos algoritmos para a consequente criação dos modelos. Considerando a inovação do projeto e a componente de investigação incluída neste desde o início, optou-se por não limitar a solução a apenas um algoritmo, mas experimentar múltiplos com o intuito de descobrir qual apresenta melhores resultados para o *dataset* criado e para cada um dos pré-processamentos aplicados.

Assim, com base na investigação efetuada anteriormente (ver 2.2.5 Aprendizagem Máquina – Algoritmos de Classificação), optou-se por utilizar os seguintes algoritmos: *Convolutional Neural Network* (CNN), *K-Nearest Neighbors* (KNN), *Support Vector Machine* (SVM), *Multilayer Perceptron* (MLP), *Decision Tree* e *Naïve Bayes*.

As seguintes seções descrevem os parâmetros utilizados para cada um destes e as suas implementações. À semelhança das etapas anteriores, foi utilizado Python e bibliotecas como o Keras e o Sklearn.

Convolutional Neural Network (CNN)

Uma rede neuronal convolucional é constituída por três partes principais: a camada convolucional, que extrai informações da imagem inicial; uma camada de *pooling*, que reduz a dimensionalidade da imagem sem perder as caraterísticas e padrões; e uma camada totalmente ligada, também conhecida como camada densa, na qual os resultados das camadas convolutivas são alimentados através de uma ou mais camadas neuronais para gerar uma previsão (Missing Link, 2020). Existem dois tipos de implementação possíveis para uma CNN: sequencial e paralela. Enquanto que a sequencial permite definir a rede camada a camada, a paralela apenas define uma única camada com diferentes tamanhos de filtro. A implementação sequencial é a abordagem mais simples de implementar uma CNN, sendo suportada diretamente pelo Keras, ao contrário da paralela.

Decidido o tipo de implementação, foi necessário definir as diferentes camadas da rede e optou-se por uma implementação com 3 camadas convolucionais (Chollet & &, 2020):

- **1ª camada convolucional –** na primeira camada, adicionou-se:
    - uma camada convolucional 2D, *Conv2D*, que representa as camadas que vão interpretar as imagens do *dataset*. O primeiro parâmetro (32) é o número de nós de cada uma das camadas. O segundo parâmetro (3, 3) é o *kernel_size* e, neste caso, significa que a camada terá uma

matriz de filtro 3x3. O *input_shape* representa o tamanho das imagens a serem lidas;

- a ativação, *Activation*, que significa a função de ativação da camada. Existem diferentes tipos – *relu*, *sigmoid*, *softmax*, *softplus*, *softsign*, *tanh*, *selu*, *elu* e exponencial – mas, neste caso, optou-se pela *relu* por ser uma função indicada para redes neuronais;
- o *MaxPooling2D*, que recebe cada saída do mapa de características da camada convolucional e prepara um mapa de características condensadas.

- **2ª camada convolucional –** igual à primeira camada;
- **3ª camada convolucional –** idêntica às duas camadas anteriores, mas com 64 nós em vez de 32.

Após a adição destas 3 camadas ao modelo sequencial, utilizou-se a função de *Flatten()*, necessária antes de utilizar o *Dense()* e que transforma uma matriz bidimensional de características num vetor. As camadas *Dense* utilizadas servem para alimentar o último tensor de saída da base convolucional e realizar a classificação. A primeira utilização desta camada serve para nivelar a saída 3D para 1D e a segunda camada superior a esta representa o número de classes que se pretende classificar – neste caso, 26. Pelo meio, ainda é adicionada uma camada de Droupout que previne o chamado *overfitting*, ou seja, quando o modelo se ajusta demasiado bem ao conjunto de dados em questão. Na camada de saída, é utilizado *softmax* como *Activation* porque é a ativação ideal para classificações multi-classe.

Por fim, é compilado o modelo onde são definidos três parâmetros: *optimizer*, *loss* e *metrics*. O *optimizer* representa a taxa de aprendizagem, ou seja, determina a rapidez com que são calculados os pesos ideais para o modelo. Neste caso, utilizou-se o *adam* por se adaptar ao longo do treino e ser muito versátil. A *loss*, ou perda, escolhido foi o *categorical_crossentropy*, por ser o mais utilizado para a classificação. Por fim, as *metrics* são as métricas pretendidas e neste caso escolheu-se a *accuracy*, ou seja, a exatidão.

```
1.  def construct_model():
2.      # Initialising
3.      cnn_classifier = Sequential()
4.
5.      # 1st conv. layer
6.      cnn_classifier.add(Conv2D(32, (3, 3), input_shape = (IMG_SIZE, IMG_SIZE,
    3)))
7.      cnn_classifier.add(Activation('relu'))
8.      cnn_classifier.add(MaxPooling2D(pool_size = (2, 2)))
9.
10.     # # 2nd conv. layer
11.     cnn_classifier.add(Conv2D(32, (3, 3)))
12.     cnn_classifier.add(Activation('relu'))
```

```
    cnn_classifier.add(MaxPooling2D(pool_size = (2, 2)))

    # # 3nd conv. layer
    cnn_classifier.add(Conv2D(64, (3, 3)))
    cnn_classifier.add(Activation('relu'))
    cnn_classifier.add(MaxPooling2D(pool_size = (2, 2)))

    # Flattening
    cnn_classifier.add(Flatten())

    # Full connection
    cnn_classifier.add(Dense(units = 64))
    cnn_classifier.add(Activation('relu'))
    # dropout prevents overfitting
    cnn_classifier.add(Dropout(0.5))

    # units must be the number of the classes
    cnn_classifier.add(Dense(units = 26))
    cnn_classifier.add(Activation('softmax'))

    # Compiling the CNN
    cnn_classifier.compile(optimizer = 'adam', # 'adam'/rmsprop'
                           loss = 'categorical_crossentropy',
                           metrics = ['accuracy'])

    return cnn_classifier
```

Figura 37 - Implementação do algoritmo CNN

K-Nearest Neighbors (KNN)

Para a utilização do algoritmo *K-Nearest Neighbors* (KNN) utilizou-se a biblioteca Sklearn que já possui toda a implementação deste. Assim, a única necessidade é a definição dos parâmetros, que neste caso é apenas o número de vizinhos (*n_neighbors*), que representa o *k*. A escolha de um *k* ideal não é fácil, podendo muitas das vezes ser obtido através de múltiplas tentativas e erros. No entanto, geralmente o *k* otimizado é obtido pela fórmula $\sqrt{n}$, isto é, pela raiz quadrada do número de amostras (Band, 2020). Como neste caso o nosso *dataset* tem um total de 3900 imagens, à partida, o número ideal de *k* será de 62.

```
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors=62)
```

Figura 38 - Implementação do algoritmo KNN

Support Vector Machine (SVM)

A implementação do *Support Vector Machine* (SVM) também está incluída na biblioteca Sklearn (Scikit-Learn, 2020a), o que facilita a configuração do modelo. Para este algoritmo foram definidos três parâmetros: a *gamma*, com um valor de 0.01, a *probability*, a *True* porque queremos utilizar o modelo para fazer previsões, e o *verbose*, também a *True* para ser apresentado na consola o progresso de treino do algoritmo. Relativamente à *gamma*, que tem de ser obrigatoriamente um valor entre 0 e 1, foi escolhido um valor de 0.01 porque, geralmente, valores superiores tendem a resultar em *overfitting*.

```
1. from sklearn import svm
2.
3. classifier = svm.SVC(gamma=0.01, probability=True, verbose=True)
```

Figura 39 - Implementação do algoritmo SVM

Multilayer Perceptron (MLP)

No *Multilayer Perceptron* (MLP) também se recorreu à biblioteca Sklearn mas a escolha dos parâmetros foi mais complexa. O MLP possui vários parâmetros possíveis, com múltiplas possibilidades, e, portanto, a escolha ideal destes pode ser um processo complicado. Assim, optou-se por utilizar o *GridSearchCV*, também da Sklearn, que permite, através de um conjunto de parâmetros, indicar quais os ideais para o algoritmo especificado e para o conjunto de dados em causa. Como se pode ver na Figura 40, a variável *parameter_space* contém todos os valores a analisar para os múltiplos parâmetros do *MLPClassifier(max_iter=300)*. O *max_iter* representa o número máximo de interações e, como um valor de 200 revelou-se insuficiente para o treino, optou-se por utilizar 300. Por fim, a função *fit()* inicia o treino do modelo e o *print(gscv.best_params_)* imprime na consola os melhores valores obtidos.

```
1. mlp = MLPClassifier(max_iter=300)
2. parameter_space = {
3.       'activation': ['tanh', 'relu'],
4.       'hidden_layer_sizes': [(70,60), (50,100,50), (100,)],
5.       'solver': ['sgd', 'adam'],
6.       'alpha': [0.01, 0.0001, 0.05],
7.       'learning_rate': ['constant','adaptive'],
8. }
9. (…)
10. gscv = GridSearchCV(mlp, parameter_space, verbose=100, cv=2, n_jobs=-1)
11. gscv.fit(np.array(train_data), np.array(train_labels))
12.
13. print(gscv.best_params_)
```

Figura 40 - Implementação do algoritmo GridSearchCV

O resultado obtido foi o seguinte:

```
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (100,), 'learning_rate'
: 'constant', 'solver': 'sgd'}
```

Com este resultado, foi possível definir o algoritmo recorrendo novamente à biblioteca Sklearn, como representado na Figura 41.

```
1. from sklearn.neural_network import MLPClassifier
2.
3. classifier = MLPClassifier(activation='tanh', hidden_layer_sizes=(100,), max
   _iter=300, alpha=0.01, learning_rate='constant', solver='sgd', verbose=True,
    random_state=21, tol=0.0001)
```

Figura 41 - Implementação do algoritmo MLP

Decision Tree

O algoritmo *Decision Tree* cria um algoritmo que prevê um valor através da aprendizagem de regras de decisão simples inferidas a partir das características dos dados, à semelhança de uma tabela de decisão.

À semelhança dos algoritmos anteriores, utilizou-se a biblioteca Sklearn e a implementação *DecisionTreeClassifier()*. Os parâmetros definidos foram o *random_state*, utilizado em algoritmos que implementam mecanismos de aleatoriedade, que serve para controlar o gerador de número aleatórios e pode ter um valor entre 0 e 42 que representa a *seed*, o que garante que mesmo em diferentes execuções teremos sempre os mesmos números aleatórios gerados. O outro parâmetro utilizado foi o *criterion*, que define a função para medir a qualidade da divisão. Neste caso, escolheu-se *entropy* porque é a ideal para ganho de informação.

```
1. from sklearn.tree import DecisionTreeClassifier
2.
3. classifier = DecisionTreeClassifier(random_state=0, criterion="entropy")
```

Figura 42 - Implementação do algoritmo Decision Tree

Naïve Bayes

O algoritmo *Naïve Bayes* é também implementado pela biblioteca Sklearn, mais propriamente o *Gaussian Naïve Bayes*, representado pela seguinte fórmula:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Figura 43 - Fórmula para o Gaussian Naïve Bayes

Ao contrário dos algoritmos anteriores, neste não se definiu qualquer parâmetro, tal como demonstrado na Figura 44.

```
1. from sklearn.naive_bayes import GaussianNB
2.
3. classifier = GaussianNB()
```

Figura 44 - Implementação do algoritmo GaussianNB

6.2.4 Cross Validation e Treino dos Modelos

Após a definição dos algoritmos e dos respetivos parâmetros, foi necessário preparar os dados para o treino e criação dos modelos. Neste aspeto o código foi dividido em duas partes: *crossvalidation* e criação dos modelos.

Cross-validation

O *cross-validation* foi implementado com a intenção de analisar os diferentes pré-processamentos efetuados, estimando valores experimentalmente e reduzindo a variabilidade. Com este conhecimento é assim possível criar posteriormente os modelos para o(s) melhor(es) algoritmo(s).

No *cross-validation* o *dataset* é dividido num número *k* de blocos e parte é utilizada para treino e outra para teste (ver 7.4 Metodologia de Avaliação) (Ren et al., 2019). No caso deste projeto, utilizou-se 10 como *k*, ou seja, 10 blocos. Nestes blocos, diferentes dados de treino são utilizados para treinar o modelo e, de seguida, os dados de testes são aplicados no modelo e os resultados registados.

A primeira fase do método de *cross-validation* é a divisão dos dados. O *dataset* em questão contém 26 classes, que representam o alfabeto, e cada classe contém 150 imagens. Utilizou-se 90% dos dados para treino (3510 imagens) e 10% para teste (390 imagens), num total de 3900 imagens. As classes foram divididas uniformemente.

O código da Figura 45 representa a implementação para dividir as imagens do *dataset* pelo número de blocos, neste caso, 10, e adicioná-los a diferentes listas. Existem bibliotecas com recursos para fazer esta divisão de forma mais simplificada, no entanto, optou-se por um processo manual para dividir os dados das diferentes classes uniformemente.

O resultado deste método são duas matrizes, uma com a informação das imagens e outra com as respetivas *labels*. Cada uma das matrizes é composta por 10 listas com 390 entradas, que representam as imagens a serem usadas para a fase de teste em cada uma das iterações. Por exemplo, na primeira iteração a primeira lista de cada uma dessas matrizes são utilizadas como dados de teste e as restantes listas representam os dados de treino. Esta divisão é aplicada para todos os algoritmos.

```
1.  def load_data_into_x_folds(folds, dir, size):
2.      train_data_folds = [[] for i in range(folds)]
3.      train_label_folds = [[] for i in range(folds)]
4.
5.      folders = os.listdir(dir)
6.      for n_folder, folder in enumerate(folders):
7.
8.          files = os.listdir(dir + folders[n_folder])
9.
10.         #calculate how many images in one fold
11.         fold_size = int(len(files) / folds)
12.
13.         random.shuffle(files)
14.         n_fold = 0
15.         #iterate through images divided in chunks and add each chunk into an
    other list(fold)
16.         for i in range(0, len(files), fold_size):
17.             fold_files = files[i:i+fold_size]
18.
19.             for j,file in enumerate(fold_files):
20.                 train_data_folds[n_fold].append(transform_image(dir + folder
     + os.sep + fold_files[j], size))
21.                 label = fold_files[j][4:-16]
22.                 label = int(label) - 1
23.                 train_label_folds[n_fold].append(label)
24.
25.             #change to the next fold
26.             n_fold += 1
27.
28.     return train_data_folds, train_label_folds
```

Figura 45 - Método para dividir os dados

Treino dos Modelos

Com os dados devidamente divididos e prontos a ser utilizados, a próxima etapa é a de treino e avaliação.

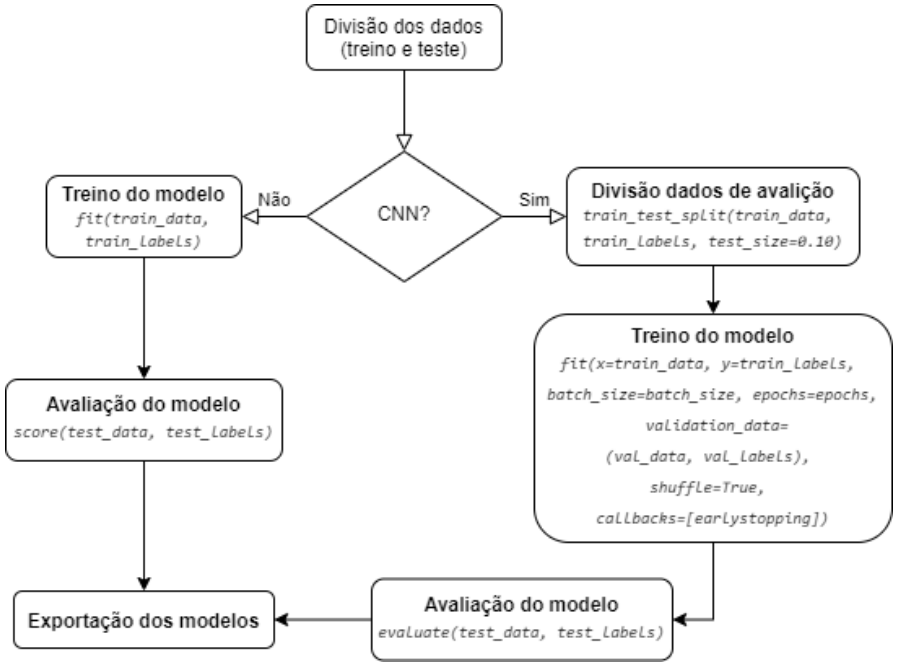

Figura 46 - Diagrama de Atividades do fluxo do processo de treino

À exceção do algoritmo CNN, que utiliza a biblioteca Keras em vez da Sklearn, todos os restantes utilizam a mesma implementação para treinar o modelo (ver Figura 47). O método *fit()* é utilizado para fazer o treino do modelo, que neste caso é representado pela variável *classifier*, passando como argumentos os dados de treino: as imagens e as respetivas *labels*. Após a finalização do treino, é feito a avaliação do mesmo através da função *score()*, que recebe por parâmetro os dados de teste. Por fim, este retorna a exatidão média (*accuracy*) com que os dados de teste foram classificados, que, neste caso, é guardada na variável *score*.

```
1.  # treino do modelo
2.  history = classifier.fit(train_data, train_labels)
3.
4.  # avaliação do modelo
5.  score = classifier.score(test_data, test_labels)
```

Figura 47 - Implementação para treinar e avaliar os modelos

Relativamente ao algoritmo CNN, a implementação é ligeiramente diferente. Numa rede neuronal sequencial os dados são divididos em três conjuntos diferentes: os dados de treino, os dados de teste e os dados de avaliação. Os dois primeiros são idênticos aos restantes algoritmos e divididos da mesma forma. Os dados de avaliação representam uma parte dos dados de treino e servem para ajustar e afinar o modelo com os resultados das métricas que obtém do seu conjunto de dados.

O método *train_test_split()* pertence à biblioteca Sklearn e faz esta divisão para o conjunto de dados de avaliação. No caso da Figura 48, utiliza-se um *test_size* de 0.10, o que significa que 10% dos dados de treino serão utilizados para avaliação.

```
1.  # divide os dados de treino em dados de treino e dados de validaçao
2.  train_data, val_data, train_labels, val_labels = train_test_split(train_data
    , train_labels, test_size=0.10)
```

Figura 48 - Divisão para os dados de avaliação

Numa rede neuronal CNN, passar apenas uma única vez todo o *dataset* pela rede não é suficiente. Assim, quando são definidos os parâmetros do treino, é necessário referir os *epochs*, que é o número de vezes que queremos aplicar o *dataset* na rede neuronal criada.

Como a escolha ideal do número de *epochs* não é um processo fácil, utilizou-se a biblioteca Keras para definir uma *callback* para detetar quando parar o treino, ou seja, quando o valor aproximadamente ideal de *epochs* fosse atingido.

No código representado na Figura 49, é criado uma *callback* do tipo *EarlyStopping*, que monitoriza o índice de perda (*monitor="val_loss"*) e que, quando deteta que o valor mínimo (*mode="min"*) desse índice não é alterado durante 3 *epochs* consecutivos, termina o treino pois significa que o modelo parou de obter informação do *dataset*.

```
1.  earlystopping = callbacks.EarlyStopping(monitor="val_loss",
2.                                          mode="min", patience=3,
3.                                          restore_best_weights=True)
```

Figura 49 - EarlyStopping para definir nº de epochs

Para iniciar o treino do modelo, é utilizada, à semelhança dos outros algoritmos, a função *fit()*. No entanto, existem parâmetros adicionais necessários para além dos dados das imagens de treino (*train_data*) e das respetivas *labels* (*train_labels*). Assim, são também definidos o *batch_size*, que representa o número de elementos de treino que estão presentes num *batch*, os *epochs*, a *validation_data*, que contém os dados de validação criados anteriormente, e o *shuffle*, que simplesmente distribui aleatoriamente os dados de treino.

De recordar que, como foi definida a função de *callback* para controlar a pausa do treino, o número de *epochs* não é relevante, tendo-se escolhido um número aleatório suficientemente alto para permitir o treino do modelo até à função de *callback* atuar.

```
1. # treino do modelo
2. history = classifier.fit(x=train_data, y=train_labels, batch_size=batch_size
   , epochs=epochs, validation_data=(val_data, val_labels), shuffle=True, callb
   acks=[earlystopping])
3.
4. # avaliação do modelo
5. scores = classifier.evaluate(test_data, test_labels, verbose=1)
```

Figura 50 - Treino e avaliação do modelo CNN

A função *evaluate()* permite obter os resultados do treino, como a exatidão (*accuracy*) e a perda (*loss*).

Por fim, após o treino dos modelos é necessário exportar os modelos para serem utilizados para futuras previsões ou para continuar o treino em caso de alterações no *dataset*.

```
1. classifier.save_weights(folderToSave + os.sep + 'classifier_weights.h5')
2. classifier.save(folderToSave + os.sep +'classifier_model.h5')
```

Figura 51 - Exportação dos modelos

6.2.5 Exportação dos resultados

Tanto no processo de treino criado para testar os vários algoritmos, com recurso ao *cross-validation*, como na criação dos modelos, foi importante registar os resultados dos treinos e das avaliações para se entender se os treinos correram de acordo com o esperado e quais os melhores modelos criados.

Assim, para além das métricas falados anteriormente, como a exatidão (*accuracy*) e a perda (*loss*), foram obtidos mais alguns índices, recorrendo à biblioteca Sklearn e exportando-os para um ficheiro de texto.

Para exportar os dados (ver Figura 52), recorrendo ao modelo treinado, fez-se uma previsão com os dados de teste (*test_data*) e, com isso, analisou-se a exatidão (*accuracy*), a precisão (*precision*), o *recall* e a *Area Under Curve* (AUC). Relativamente à precisão e ao *recall* foram registados três tipos diferentes de valores:

- **Micro –** calcula a métrica com base na contribuição de cada uma das classes, ou seja, tem em consideração a quantidade de dados para cada uma das classes;
- **Macro –** calcula a métrica de forma independente para cada uma das classes, ou seja, tratando cada uma das classes de forma igual;
- ***Weighted* –** calcula a métrica de forma independente para cada uma das classes, mas quando as soma usa um peso que depende do número de *labels* verdadeiras de cada classe;

Para a curva AUC, também foram registados quatro valores diferentes – *macro OvR*, *weighted OvR*, *macro OvO* e *weighted OvO*. Como a curva ROC e o respetivo AUC, não podem ser calculados diretamente através de dados multi-classe, é necessário converter os dados como se fosse um problema binário. O OvR (*One-Vs-Rest*) e OvO (*One-Vs-One*) são diferentes configurações para classificações multi-classe, que tratam dessa transformação dos dados.

Aquando do desenvolvimento desta parte do código, optou por se guardar todas estas métricas por uma questão de análise, apesar de não serem todas necessárias. Por exemplo, em classificações multi-classe, como é o caso deste projeto, o *recall* micro e macro são sempre iguais. No entanto, optou por se incluir na mesma a exportação destes dois valores individualmente para validar se os resultados eram efetivamente os esperados. Também é expectável que todos os resultados da AUC calculados – *macro OvR*, *weighted OvR, macro OvO, weighted OvO* – tenham os mesmos valores, visto que as configurações apenas significam diferentes formas de processar os dados para classificação binária. Os resultados entre a AUC *macro* e *weighted* também devem ser iguais porque estamos a balancear os dados entre classes no momento da divisão do *cross-validation*.

A análise aos resultados de cada uma destas métricas está presente no capítulo de Avaliação de resultados.

```
# accuracy: (tp + tn) / (p + n)
accuracy_scr = accuracy_score(label_data, pred_classes)

# precision tp / (tp + fp)
precision_micro = precision_score(label_data, pred_classes, average='micro')

precision_macro = precision_score(label_data, pred_classes, average='macro')

precision_weighted = precision_score(label_data, pred_classes, average='weighted')

# recall: tp / (tp + fn)
recall_micro = recall_score(label_data, pred_classes, average='micro')
recall_macro = recall_score(label_data, pred_classes, average='macro')
recall_weighted = recall_score(label_data, pred_classes, average='weighted')


# ROC AUC
macro_roc_auc_ovo = roc_auc_score(label_data_categorical, probs, multi_class="ovo", average="macro")
weighted_roc_auc_ovo = roc_auc_score(label_data_categorical, probs, multi_class="ovo", average="weighted")
macro_roc_auc_ovr = roc_auc_score(label_data_categorical, probs, multi_class="ovr", average="macro")
weighted_roc_auc_ovr = roc_auc_score(label_data_categorical, probs, multi_class="ovr", average="weighted")
```

Figura 52 - Utilização da biblioteca Sklearn para cálculo das métricas

Além da exportação das métricas, também se criou uma função para gerar as matrizes de confusão e as curvas ROC para cada um dos modelos criados (ver 7.3 Identificação dos indicadores e fontes de informação). Utilizou-se a biblioteca Matplotlib, que permite criar diferentes tipos de diagramas e gráficos.

A Figura 53 e a Figura 54 representam um exemplo das matrizes de confusão e dos gráficos ROC criados.

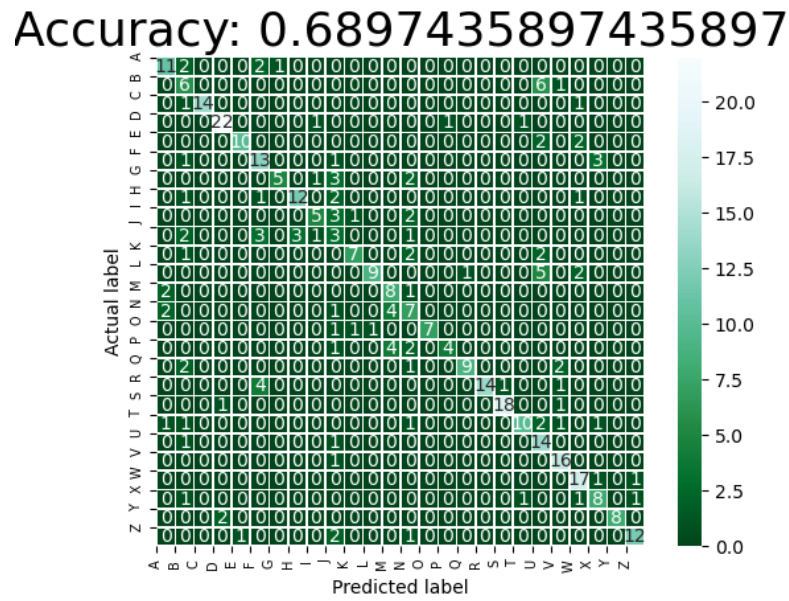

Figura 53 - Exemplo de uma matriz de confusão criada

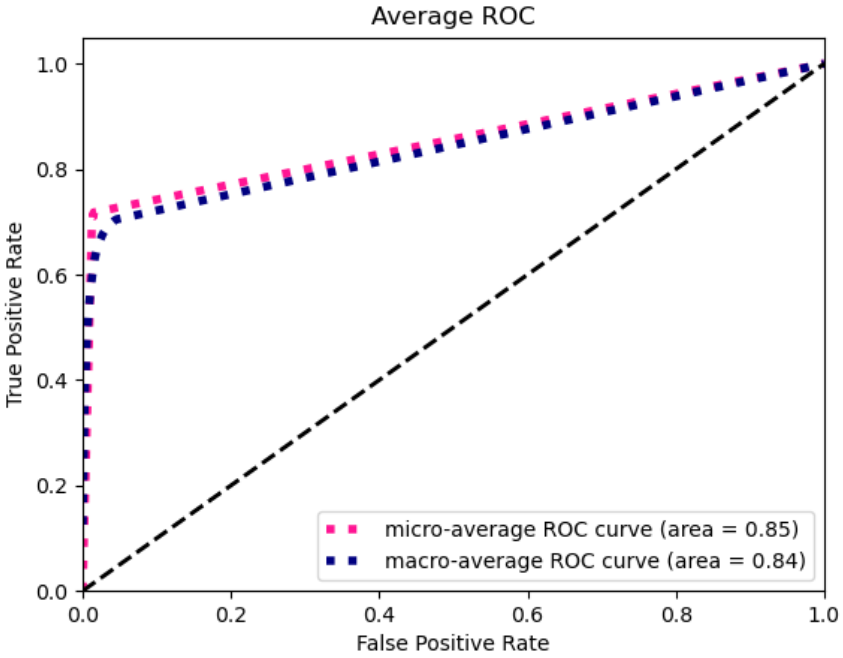

Figura 54 - Exemplo de uma curva ROC criada

6.2.6 Previsão

Com os modelos criados, a etapa final consiste na previsão. Através da importação do modelo, a previsão consiste na classificação final de uns determinados dados de entrada, ou seja, neste caso consiste na previsão de qual a letra do alfabeto em questão.

Antes de efetuar a previsão, é necessário aplicar o pré-processamento adequado à imagem a classificar, de forma a facilitar a interpretação que o modelo faz desta. De seguida, é importado o modelo criado anteriormente, e, por fim, é feita a previsão da imagem transformada anteriormente.

```
1.  def predict_sign(img_path):
2.      size = (IMG_SIZE, IMG_SIZE)
3.
4.      # loads and ransforms the image to classify
5.      transformed_img = transform_image(img_path, size)
6.
7.      # loads model of the classifier
8.      cnn_classifier = keras.models.load_model(PATH_TO_MODEL)
9.
10.     # predicts the class
11.     result = cnn_classifier.predict(transformed_img)
```

Figura 55 - Exemplo de previsão para um modelo CNN

7 Avaliação de resultados

Neste capítulo é apresentado o processo de avaliação, identificando as grandezas, as hipóteses, os indicadores e fontes de informação, quais as melhores metodologias de avaliação e as experiências realizadas e respetivos resultados.

Tal como referido ao longo dos capítulos anteriores, o problema de origem deste projeto foi identificado pela comunidade surda, que demonstrou dificuldades no momento da comunicação com pessoas sem conhecimentos de Língua Gestual Portuguesa. Os objetivos traçados inicialmente pretendiam o desenvolvimento de uma solução base que no futuro seja capaz de atenuar este problema, suportando a comunidade surda no seu dia-a-dia e apelando a uma maior integração social.

Assim, para garantir que os objetivos foram cumpridos com sucesso, é necessário um processo de avaliação, capaz de considerar o que se pretendia no início e o que se conseguiu no fim do projeto.

7.1 Grandezas

Desde o início que foram identificadas grandezas que teriam de ser consideradas durante todo o processo de desenvolvimento do projeto, como na escolha de tecnologias e decisões de abordagens, de forma a que a solução final fosse a melhor possível.

7.1.1 Exatidão

Tratando-se de um sistema de tradução durante um diálogo ou transmissão de uma mensagem, é fundamental que o sistema apresente uma elevada exatidão na sua funcionalidade. Caso contrário, baixa exatidão pode originar classificações erradas e, consequentemente, más interpretações, deturpações da verdade ou uma má compreensão do destinatário da mensagem. Igualmente, uma taxa reduzida ou inferior à expectável pode resultar na falha do principal objetivo do projeto – o auxílio à comunidade surda nos diversos processos da sociedade.

7.1.2 Tempo

Tratando-se de uma solução que pode ser utilizada em situações de diálogo, é importante que esta apresente um tempo de resposta consideravelmente reduzido, de forma a proporcionar uma comunicação fluída ou uma resposta rápida.

7.2 Hipótese

Com base no problema e nos objetivos que se pretendem alcançar com este projeto, a hipótese que se define é que possível reconhecer as configurações de ambas as mãos para efeitos de tradução automática de língua gestual de forma não invasiva, recorrendo a tecnologia de visão computacional.

7.3 Identificação dos indicadores e fontes de informação

Tratando-se de um projeto com poucas experiências passadas, foi importante que o seu desenvolvimento considerasse diferentes classificadores de forma a identificar o mais apropriado para o problema em questão.

Existem dois tipos de classificadores: binários ou multicritério. Os binários aplicam-se em casos que o resultado da classificação só pode conter dois resultados, como por exemplo **1 e 0** ou **Sim e Não**. No caso deste projeto, a classificação é multicritério ou multi-classe porque o resultado pode ser qualquer uma das configurações de mão ou gestos que o sistema conhece.

Um dos melhores métodos para avaliar a performance de um classificador é através da matriz de confusões, que permite calcular métricas como a precisão (*accuracy*), exatidão (*precision*), o *recall*, a curva ROC (*Receiver Operating Characteristic Curve*) e o respetivo AUC (*Area Under Curve*).

Matriz de Confusão

Considerando um classificador e uma instância, existem quatro possíveis resultados. Se a instância for positiva e for classificada como positiva, é considerada um verdadeiro positivo (TP – *true positive*); se for classificada como negativa, representa um falso positivo (FP – *false positive*). Se a instância for negativa e for classificada como positiva, é considerada um verdadeiro negativo (TN – *true negative*); se for classificada como negativa, representa um falso

negativo (FN – *false negative*). Assim, com um determinado classificador e um conjunto de instâncias, isto é, dados de teste, é possível construir uma matriz de confusões que represente as disposições do conjunto de instâncias (ver Figura 56). A partir desta, é então possível, tal como referido anteriormente, calcular um variado conjunto de métricas (Fawcett, 2006).

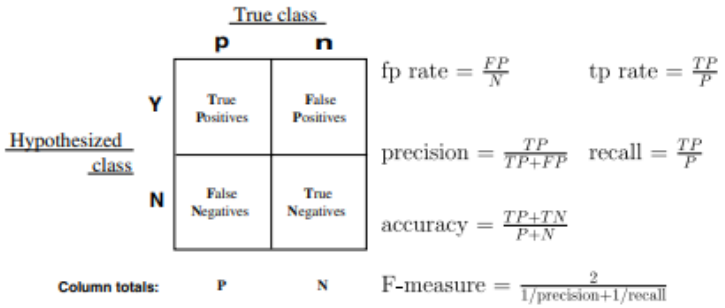

Figura 56 - Matriz de confusão e métricas associadas (Fawcett, 2006)

Exatidão (*accuracy*)

A exatidão representa o quanto se previu corretamente considerando o total de classes e é calculada através da fórmula:

$$accuracy = \frac{TP + TN}{TP + FN + FP + TN}$$

Precisão (*precision*)

A precisão representa a proporção dos dados que o modelo classifica como relevantes e que realmente o foram. É calculada através de:

$$precision = \frac{TP}{TP + FP}$$

Recall

Por fim, a *recall* representa a capacidade de encontrar todas as instâncias relevantes no conjunto de dados e é representada pela seguinte fórmula:

$$recall = \frac{TP}{TP + FN}$$

Receiver Operating Characteristic – ROC

O ROC (*Receiver Operating Characteristic*) é um gráfico que representa uma técnica para visualizar, organizar e selecionar classificadores com base no seu desempenho. O ROC é um gráfico bidimensional em que a taxa de positivos verdadeiros (*TP Rate* – ver Figura 56) representa o eixo do Y e a taxa de falsos positivos (*FP Rate* – ver Figura 56) representa o eixo do Y (Fawcett, 2006).

A Figura 57 representa um exemplo de um gráfico ROC com cinco classificadores diferentes.

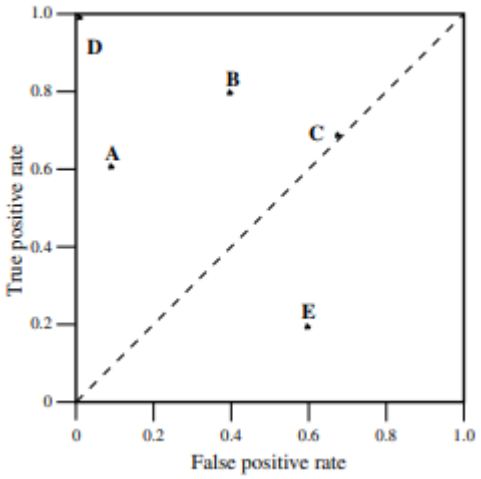

Figura 57 - Exemplo de um gráfico ROC

Area Under Curve – AUC

A AUC (*Area Under Curve*) está diretamente relacionada ao ROC (Fawcett, 2006). Tal como referido anteriormente, um gráfico ROC é uma representação bidimensional do desempenho de um classificador. Para efetuar a comparação entre diferentes classificadores, é mais apropriado reduzir o desempenho ROC a um único valor representativo da performance expectável. Assim, a AUC é o método mais comum para isso, representando a área abaixo da curva ROC.

A AUC é uma porção da área total do gráfico ROC e cada um dos seus eixos são valores entre 0 e 1.0, o valor da AUC será também sempre um valor entre 0 e 1.0. Esta representa uma probabilidade estatística importante: a AUC de um classificador é equivalente à probabilidade de o classificador classificar uma instância positiva escolhida aleatoriamente mais alta do que uma instância negativa também escolhida aleatoriamente (Fawcett, 2006).

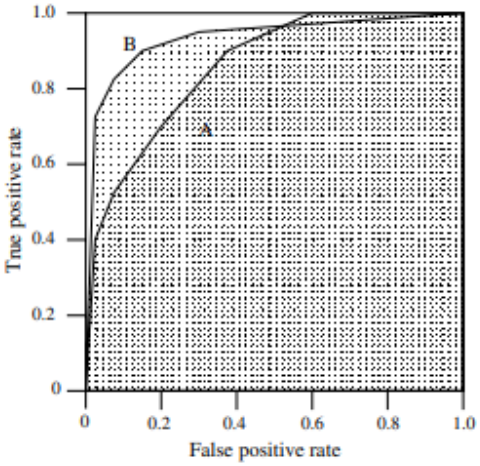

Figura 58 - Exemplo de gráfico ROC e respetiva AUC para dois classificadores (A e B)

7.4 Metodologia de Avaliação

Aplicar a técnica de estimação *K-fold cross validation* será interessante para avaliar a estabilidade do modelo definido, garantido que este tem a maioria dos padrões dos dados corretamente definidos. Nesta técnica o *dataset* é dividido em *k* número de blocos e por cada iteração um bloco diferente é reservado para teste e os restantes são utilizados para treinar os classificadores (ver Figura 59). Para cada uma das iterações, podem ser calculadas as respetivas matrizes de confusão e os respetivos índices referidos anteriormente.

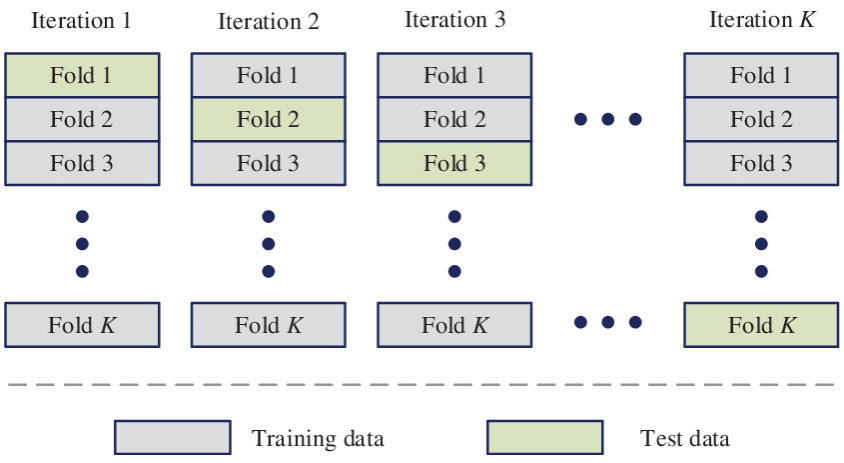

Figura 59 - K-Fold Cross Validation (Ren et al., 2019)

7.5 Preparação de experiências

Um dos objetivos deste projeto era a exploração de diferentes algoritmos de classificação, identificando aqueles que apresentam melhores resultados. Estes resultados estão sempre depende dos *dataset* e da qualidade dos dados, portanto, os resultados obtidos neste projeto não devem ser globalizados. O que significa que, no futuro, em caso de alteração do *dataset* ou extensão do mesmo, o ideal é voltar a avaliar os diversos algoritmos para garantir que os resultados obtidos anteriormente ainda se mantêm.

Nas experiências realizadas no âmbito deste projeto, existem duas variáveis que têm impacto nos respetivos resultados: o pré-processamento e os algoritmos utilizados. Tal como referido anteriormente, foram utilizados diferentes tipos de pré-processamento nas imagens do *dataset*, o que influência a maneira como os algoritmos interpretam os dados e retiram informação dos mesmos. Igualmente, também foram utilizados diferentes algoritmos, apropriados para diversos contextos e com potencialidades distintas.

As seguintes secções e capítulos descrevem as diferentes experiências realizadas e os seus parâmetros, e são apresentados os respetivos resultados.

7.5.1 Experiências realizadas

Recordando informação referida em capítulos anteriores, para este projeto e respetivos testes foram usados os seguintes dados:

- **Tamanho total do *dataset* –** 3900 imagens;
- **Tamanho do dados de treino –** 90% do *dataset*: 3510 imagens;
- **Tamanho dos dados de teste –** 10% do *dataset:* 390 imagens;
- **Algoritmos de classificação –** 6 algoritmos no total:
    - *Convolutional Neural Network (CNN*);
    - *K-Nearest Neighbors* (k-NN);
    - *Support Vector Machine* (SVM);
    - *Multilayer Perceptron* (MLP);
    - *Decision Tree*;
    - *Naïve Bayes*;
- **Tipos de pré-processamento –** 8 pré-processamentos no total:
    - Redimensionamento (128x128);
    - Redimensionamento (128x128) e preto e branco;
    - Redimensionamento (128x128) e extração de limites;
    - Redimensionamento (128x128) e extração de limites com suavização do plano de fundo;
    - Redimensionamento (128x128) com alto contraste;

- Redimensionamento (128x128) e preto e branco, com alto contraste;
- Redimensionamento (128x128) e extração de limites, com alto contraste;
- Redimensionamento (128x128) e extração de limites com suavização do plano de fundo, com alto contraste.

Figura 60 - Diversos pré-processamentos utilizados

A Tabela 12 representa o conjunto de experiências realizado com base no tipo de pré-processamento.

Tabela 12 - Diferentes experiências realizadas considerando os tipos pré-processamento

	Redim. (128x128)	P&B	Limites	Limites c/ suavização	Contraste alto
Experiência 1	✓	✗	✗	✗	✗
Experiência 2	✓	✓	✗	✗	✗
Experiência 3	✓	✗	✓	✗	✗
Experiência 4	✓	✗	✗	✓	✗
Experiência 5	✓	✗	✗	✗	✓
Experiência 6	✓	✓	✗	✗	✓
Experiência 7	✓	✗	✓	✗	✓
Experiência 8	✓	✗	✗	✓	✓

Este conjunto de 8 experiências, foi realizado para cada um dos algoritmos, resultando num total de 48 experiências. Para cada uma destas, foi utilizado o método de *k cross-validation*, descrito anteriormente, considerando um *k* de 10. Ou seja, para cada experiência foram realizadas 10 divisões diferentes dos dados e o respetivo treino e avaliação do modelo com esse conjunto de dados.

As métricas foram registadas num ficheiro de texto com os resultados globais das iterações – exatidão média (*average accuracy*) – e os dados individuais para a respetiva iteração – exatidão (*accuracy*), precisão (*precision micro*, *macro* e *weighted*), *recall* (*micro, macro* e *weighed*) e AUC (*macro OvR*, *weighted OvR, macro OvO, weighted OvO*) (ver 6.2.5 Exportação dos resultados).

7.6 Resultados obtidos

Nas seguintes secções são apresentados os resultados obtidos para cada uma das experiências com os algoritmos e respetivas configurações descritas anteriormente. Para a obtenção destes resultados, foi utilizado um cross-validation com 10 iterações (*k*) e calculadas as médias de cada uma das métricas. Apesar de terem sido exportadas outras, aquelas que são apresentadas nas seguintes tabelas são: exatidão (*accuracy*) e respetivo desvio padrão, precisão (*precision*) micro, precisão macro/weighted, *recall* e a AUC. Estes valores foram arredondados para quatro casas decimais.

De notar que, em classificações multi-classe, os valores de exatidão, precisão micro e *recall* são sempre os mesmos, tal como é possível verificar nos valores apresentados. Isto valida os cálculos efetuados.

Convolutional Neural Network (CNN)

O algoritmo CNN, tratando-se de um algoritmo de *deep learning*, é indicado para problemas de processamento de imagem e os resultados seguintes comprovam-no. Apresentou sempre resultados superiores a 96% na exatidão, sendo que a experiência que apresentou melhores resultados foi a Experiência 3. No entanto, qualquer uma das experiências teve resultados muito positivos.

A Tabela 13 apresenta os resultados obtidos.

Tabela 13 - Resultados das experiências realizadas com o algoritmo CNN

	$\bar{X}_{ACCURACY}$	Desvio Padrão	$\bar{X}_{PRECISION\ (MICRO)}$	$\bar{X}_{PRECISION\ (MACRO)}$	$\bar{X}_{RECALL}$	$\bar{X}_{AUC}$
Experiência 1	± 0.9715	± 0.0096	± 0.9715	± 0.9739	± 0.9715	± 0.9995
Experiência 2	± 0.9700	± 0.0093	± 0.9700	± 0.9721	± 0.9700	± 0.9997
Experiência 3	± 0.9746	± 0.0047	± 0.9746	± 0.9765	± 0.9746	± 0.9997
Experiência 4	± 0.9682	± 0.0107	± 0.9682	± 0.9708	± 0.9682	± 0.9997
Experiência 5	± 0.9722	± 0.0082	± 0.9722	± 0.9751	± 0.9722	± 0.9997
Experiência 6	± 0.9692	± 0.0090	± 0.9692	± 0.9714	± 0.9692	± 0.9994
Experiência 7	± 0.9651	± 0.0103	± 0.9651	± 0.9679	± 0.9651	± 0.9995
Experiência 8	± 0.9651	± 0.0141	± 0.9651	± 0.9678	± 0.9651	± 0.9996

K-Nearest Neighbors (KNN)

O algoritmo KNN apresentou resultados menos positivos que o CNN, sendo a melhor média de exatidão foi na Experiência 5, com cerca de 76%. Este algoritmo demonstrou mais dificuldade nas experiências 3, 4, 7 e 8, isto é, aquelas em que são extraídos da imagem os limites dos objetos.

Tabela 14 - Resultados das experiências realizadas com o algoritmo KNN

	$\bar{X}_{ACCURACY}$	Desvio Padrão	$\bar{X}_{PRECISION\ (MICRO)}$	$\bar{X}_{PRECISION\ (MACRO)}$	$\bar{X}_{RECALL}$	$\bar{X}_{AUC}$
Experiência 1	± 0.7364	± 0.0349	± 0.7364	± 0.7857	± 0.7364	± 0.9895
Experiência 2	± 0.7272	± 0.0250	± 0.7272	± 0.7864	± 0.7272	± 0.9874
Experiência 3	± 0.2095	± 0.0186	± 0.2095	± 0.3740	± 0.2095	± 0.8580
Experiência 4	± 0.0985	± 0.0162	± 0.0985	± 0.1487	± 0.0985	± 0.8188
Experiência 5	± 0.7631	± 0.0162	± 0.7631	± 0.7996	± 0.7631	± 0.9905
Experiência 6	± 0.7359	± 0.0237	± 0.7359	± 0.7781	± 0.7359	± 0.9880
Experiência 7	± 0.1103	± 0.0101	± 0.1103	± 0.2124	± 0.1103	± 0.8651
Experiência 8	± 0.0856	± 0.0058	± 0.0856	± 0.1198	± 0.0856	± 0.8767

Support Vector Machine (SVM)

À semelhança do algoritmo anterior, o SVM também teve melhores resultados nas experiências com apenas redimensionamento ou com o *dataset* a preto e branco. As experiências com o *dataset* a preto e branco foram as que apresentaram melhores resultados, principalmente a Experiência 2, ou seja, sem alto contraste.

O SVM é um algoritmo significativamente lento e de elevado processamento. Como tal, para este algoritmo no *cross-validation* foi utilizado um *k* de 4 iterações pois verificou-se que, com os recursos disponíveis – processador do computador onde se efetuaram os treinos ou Google Colab –, um treino com *k* de 10 demorava mais de um dia inteiro a processar. Como tal, reduziu-se o número de iterações, mas mantendo a divisão do *dataset* – 90% para treino e 10% para teste.

Tabela 15 - Resultados das experiências realizadas com o algoritmo SVM

	$\overline{X}_{ACCURACY}$	Desvio Padrão	$\overline{X}_{PRECISION\ (MICRO)}$	$\overline{X}_{PRECISION\ (MACRO)}$	$\overline{X}_{RECALL}$	$\overline{X}_{AUC}$
Experiência 1	± 0.6103	± 0.0241	± 0.6103	± 0.9650	± 0.6106	± 0.9989
Experiência 2	± 0.6692	± 0.0258	± 0.6692	± 0.9645	± 0.6692	± 0.9986
Experiência 3	0.1521	± 0.0094	0.1521	0.6244	0.1521	0.1529
Experiência 4	± 0.1724	± 0.0264	± 0.1724	± 0.6341	± 0.1724	± 0.2825
Experiência 5	± 0.4148	± 0.0330	± 0.4148	± 0.9639	± 0.4148	± 0.9994
Experiência 6	± 0.4929	± 0.0100	± 0.4929	± 0.9643	± 0.4929	± 0.9996
Experiência 7	± 0.1160	± 0.0111	± 0.1160	± 0.5585	± 0.1160	± 0.2471
Experiência 8	± 0.1562	± 0.0088	± 0.1562	± 0.6422	± 0.1562	± 0.1375

Multilayer Perceptron (MLP)

Tal como o CNN, o MLP também é um algoritmo de *deep learning*, ideal para classificações de imagens. Os melhores resultados foram a Experiência 1 com uma exatidão média de 97,12%. No entanto, todas as experiências apresentam resultados perto dos 90%, o que demonstra a capacidade de aprendizagem deste algoritmo e do CNN.

Tabela 16 - Resultados das experiências realizadas com o algoritmo MLP

	$\bar{X}_{ACCURACY}$	Desvio Padrão	$\bar{X}_{PRECISION\ (MICRO)}$	$\bar{X}_{PRECISION\ (MACRO)}$	$\bar{X}_{RECALL}$	$\bar{X}_{AUC}$
Experiência 1	± 0.9712	± 0.0056	± 0.9712	± 0.9729	± 0.9712	± 0.9995
Experiência 2	± 0.9528	± 0.0098	± 0.9528	± 0.9554	± 0.9528	± 0.9988
Experiência 3	± 0.8918	± 0.0091	± 0.8918	± 0.8974	± 0.8918	± 0.9954
Experiência 4	± 0.8826	± 0.0167	± 0.8826	± 0.8879	± 0.8826	± 0.9945
Experiência 5	± 0.97	± 0.0079	± 0.97	± 0.9719	± 0.97	± 0.9993
Experiência 6	± 0.9544	± 0.0058	± 0.9544	± 0.9571	± 0.9544	± 0.9987
Experiência 7	± 0.8649	± 0.0131	± 0.8649	± 0.8696	± 0.8649	± 0.9932
Experiência 8	± 0.8815	± 0.0175	± 0.8815	± 0.8871	± 0.8815	± 0.9945

Decision Tree

O algoritmo *Decision Tree* apresentou resultados idênticos ao KNN, com cerca de 70% para as experiências que apenas redimensionam os dados ou transformam as imagens para preto e branco.

Tabela 17 - Resultados das experiências realizadas com o algoritmo Decision Tree

	$\bar{X}_{ACCURACY}$	Desvio Padrão	$\bar{X}_{PRECISION\ (MICRO)}$	$\bar{X}_{PRECISION\ (MACRO)}$	$\bar{X}_{RECALL}$	$\bar{X}_{AUC}$
Experiência 1	± 0.7844	± 0.0224	± 0.7844	± 0.7947	± 0.7844	± 0.8879
Experiência 2	± 0.7744	± 0.0125	± 0.7744	± 0.7873	± 0.7744	± 0.8827
Experiência 3	± 0.3521	± 0.0120	± 0.3521	± 0.3592	± 0.3521	± 0.6631
Experiência 4	± 0.3231	± 0.0182	± 0.3231	± 0.3311	± 0.3231	± 0.648
Experiência 5	± 0.7379	± 0.0221	± 0.7379	± 0.7459	± 0.7379	± 0.8637
Experiência 6	± 0.7203	± 0.0171	± 0.7203	± 0.7319	± 0.7203	± 0.8545
Experiência 7	± 0.2295	± 0.0255	± 0.2295	± 0.2358	± 0.2295	± 0.5993
Experiência 8	± 0.2795	± 0.0211	± 0.2795	± 0.2866	± 0.2795	± 0.6253

Naïve Bayes

Por fim, o algoritmo Naïve Bayes apresentou resultados relativamente equilibrados entre as diferentes experiências realizadas, no entanto, a Experiência 1 foi a que obteve melhores resultados, mas não chegando aos 70% de exatidão.

Tabela 18 - Resultados das experiências realizadas com o algoritmo Naïve Bayes

	$\overline{X}_{ACCURACY}$	Desvio Padrão	$\overline{X}_{PRECISION\ (MICRO)}$	$\overline{X}_{PRECISION\ (MACRO)}$	$\overline{X}_{RECALL}$	$\overline{X}_{AUC}$
Experiência 1	± 0.6844	± 0.0167	± 0.6844	± 0.7328	± 0.6844	± 0.8466
Experiência 2	± 0.6503	± 0.0232	± 0.6503	± 0.6992	± 0.6503	± 0.8264
Experiência 3	± 0.5903	± 0.0279	± 0.5903	± 0.7276	± 0.5903	± 0.7875
Experiência 4	± 0.6208	± 0.0259	± 0.6208	± 0.7322	± 0.6208	± 0.8031
Experiência 5	± 0.6282	± 0.0267	± 0.6382	± 0.6998	± 0.6382	± 0.8284
Experiência 6	± 0.5769	± 0.0175	± 0.5769	± 0.6128	± 0.5769	± 0.7980
Experiência 7	± 0.5295	± 0.0185	± 0.5295	± 0.6380	± 0.5295	± 0.7624
Experiência 8	± 0.5185	± 0.0226	± 0.5185	± 0.6735	± 0.5185	± 0.7509

7.6.1 Sumário de experiências

A Tabela 19 representa um sumário das várias experiências realizadas, considerando as médias da exatidão e os respetivos desvios padrão. Para cada uma da experiência é apresentada a melhor exatidão no total dos testes efetuados e o algoritmo com que esse resultado foi obtido.

Assim, através da sua análise, podemos verificar que, como seria de prever, os algoritmos que obtiveram melhores resultados foram o CNN e o MLP. O CNN destacou-se em todas as experiências realizadas e apresentou sempre excelentes resultados, sendo que a experiência com maior exatidão foi a Experiência 3, com cerca de 97,46%. Nesta experiência, o desvio padrão foi de 0.0043, o valor mais baixo de todas as experiências. Isto demonstra que houve um baixo desvio da média entre as várias iterações do *cross-validation*.

O MLP, apesar de não ter conseguido resultados tão altos nas médias da exatidão, na Experiência 1 e Experiência 6 foram considerados os melhores devido ao baixo desvio padrão, comparativamente aos valores do CNN. De forma geral, as várias experiências apresentaram

resultados muito semelhantes entre si, o que significa que independentemente do pré-processamento escolhido, os resultados com estes algoritmos serão muito positivos. A Figura 61 representa um diagrama de superfície que relaciona os algoritmos e as experiências realizadas, considerando a exatidão. Como é possível analisar, o CNN é o algoritmo com valores de exatidão mais próximos de 1, mas o MLP também se destaca.

Tabela 19 - Sumário das Experiências ao nível da Exatidão (accuracy) e respetivo algoritmo

	Melhor $\bar{X}_{ACCURACY}$	Desvio Padrão	Melhor Algoritmo
Experiência 1	0.9712	± 0.0056	MLP
Experiência 2	0.9700	± 0.0093	CNN
Experiência 3	0.9746	± 0.0047	CNN
Experiência 4	0.9682	± 0.0107	CNN
Experiência 5	0.9722	± 0.0082	CNN
Experiência 6	0.9544	± 0.0058	MLP
Experiência 7	0.9651	± 0.0103	CNN
Experiência 8	0.9651	± 0.0141	CNN

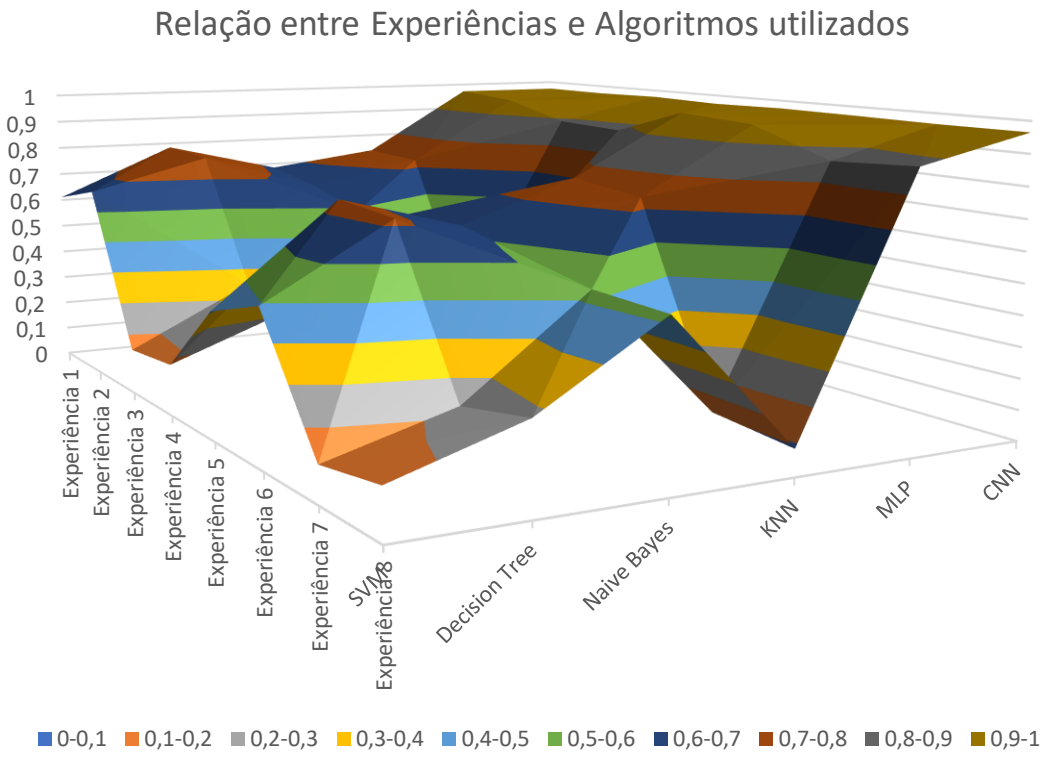

Figura 61 - Gráfico de Superfície

7.6.2 Conclusões sobre as experiências

Com os resultados obtidos, pode-se concluir que para o *dataset* criado, o algoritmo CNN apresenta resultados muito promissores. O algoritmo MLP também aparenta tratar-se de uma solução muito viável.

Os resultados de uma classificação estão sempre dependentes dos dados de entrada e o pré-processamento aplicado pode ter resultados variáveis conforme as condições da imagem captada, como, por exemplo, a luminosidade. No entanto, é possível verificar que, independentemente do pré-processamento aplicado, os resultados para as métricas registadas são sempre positivos e com valores altos. Assim, conforme as diferentes utilizações da solução desenvolvida, podem ser utilizados diferentes algoritmos – entre MLP e CNN – classificando dados com diferentes tipos de pré-processamento, desde que este processo seja consistente. Ou seja, desde que os modelos tenham sido treinados com o mesmo pré-processamento que está a ser aplicado aos dados a serem processados. Apenas desta forma, serão garantidos os melhores resultados.

Considerando estes resultados, é também possível concluir que a hipótese anteriormente definida é válida, ou seja, é possível reconhecer as configurações de ambas as mãos para efeitos de tradução automática de língua gestual de forma não invasiva, recorrendo a tecnologia de visão computacional.

7.7 Criação do Modelo final

Tal como referido anteriormente, com base no sumário das experiências, qualquer modelo criado para um determinado pré-processamento e com recurso ao respetivo algoritmo, principalmente CNN e MLP, resultaria num modelo de classificação com bons índices.

Tendo em conta que o melhor resultado foi para a Experiência 3, usando o algoritmo CNN, optou-se por criar um modelo final para este. Assim, utilizando o *dataset* com um pré-processamento de extração de limites e sem alto contraste, treinou-se e exportou-se um modelo CNN para ser futuramente utilizado.

Com a implementação descrita anteriormente (ver 6.2.3 Configuração dos Algoritmos de Classificação), o algoritmo detetou automaticamente quando parar o treino, ou seja, quando a perda de informação da avaliação não atingia o mínimo já registado há mais de três *epochs*. Assim, para este treino, o número de *epochs* foi de 14 e o valor da exatidão foi de

aproximadamente 0.9846 ou 98,46%, ou seja, um valor ainda superior ao obtido durante o processo de avaliação com recurso a *cross-validation*.

A Figura 62 demonstra a matriz de confusões criada e quais as *labels* que foram incorretamente classificadas para os 390 dados de teste (10% do *dataset*). Diretamente relacionada, a Figura 63 representa o gráfico ROC gerado e demonstra porquê que a AUC calculada apresenta um valor de 0.99.

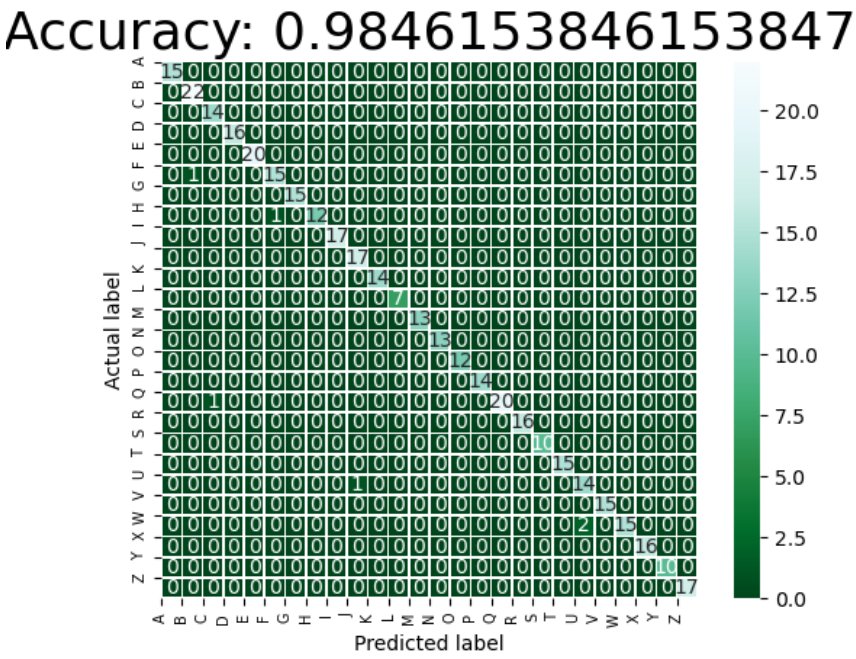

Figura 62 - Matriz de Confusão para o modelo criado

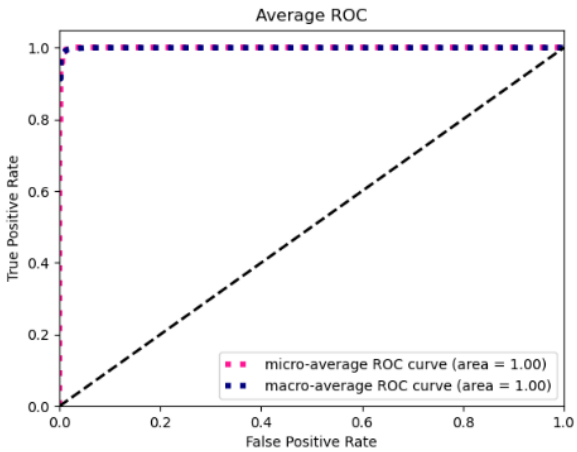

Figura 63 - Curva ROC

Com este modelo é então possível fazer a previsão de imagens, recorrendo ao código descrito anteriormente (ver 6.2.6 Previsão). A imagem na qual vai ser aplicada a previsão tem de ser inicialmente transformada para o pré-processamento esperado pelo modelo.

O código da Figura 64, realiza a previsão para a imagem inicialmente recebida e consequentemente pré-processada, e organiza o resultado por ordem crescente – da *label* com maior probabilidade de ser a representa na imagem, para a *label* com menor probabilidade.

```
1.  def predict_sign(img_path):
2.      (...)
3.
4.      # predicts the class
5.      result = cnn_classifier.predict(transformed_img)
6.
7.      zipped = zip(LABELS, result[0])
8.      print(sorted(zipped, key = lambda x: x[1], reverse=True))
```

Figura 64 - Código para fazer previsão e organizar os dados

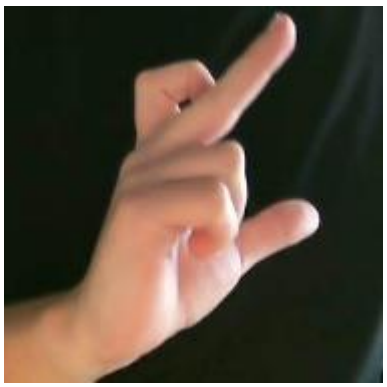

Figura 65 - Imagem a ser classificada

Ou seja, para uma imagem capturada como a da Figura 65, um excerto do resultado da previsão é:

```
1.  [('R', 0.9998611), ('J', 8.987424e-05), ('X', 4.622938e-05), (…)
```

Figura 66 - Excerto do resultado da previsão

Neste caso, o modelo identificou corretamente, com uma certeza de 99%, a Figura 65 como sendo o gesto para a letra R.

8 SignToText

O SignToText é uma aplicação desenvolvida por estudantes no Instituto Superior de Engenharia no Porto, que, tal como o nome indica, pretende transformar os gestos da língua gestual portuguesa em texto. Esta aplicação ainda está em fase de desenvolvimento, não se encontrando disponível para utilização.

A aplicação consiste numa plataforma *Web*, desenvolvida em Python e JavaScript. Através de uma câmara, como uma *webcam*, capta um vídeo durante 5 segundos onde são detetados os vários movimentos do sujeito e recolhidos os diversos pontos do corpo como ombros, cotovelos, pulsos, entre outros. Para isso, é utilizado *PoseNet*, um modelo de visão computacional capaz de detetar a pose de uma pessoa numa imagem ou vídeo, estimando onde se encontram as articulações-chave do corpo e registando-as através de pontos numéricos.

8.1 Integração na aplicação

Considerando que o objetivo principal deste projeto era a tradução de gestos da Língua Gestual Portuguesa para texto, foi proposto integrar com a aplicação o código desenvolvido. Assim, o objetivo seria extrair os gestos detetados durante a captação de vídeo e dos diversos *frames* que constituem o mesmo, classificá-los e registar os três resultados com maior probabilidade.

Desta forma, seria possível registar os três gestos com maior probabilidade juntamente com a informação do corpo retirada através do *PoseNet*, como movimento das mãos e braços. Consequentemente, com o conjunto destes dados e através do treino de um modelo apropriado para este fim, é possível classificar palavras ou até expressões como “bom dia” ou “boa noite”.

A integração do código nesta aplicação foi simples devido à semelhança entre o objetivo e o trabalho que já tinha sido realizado, não havendo necessidade de grandes adaptações. O modelo utilizado foi o criado anteriormente (ver 7.7 Criação do Modelo final), ou seja, um modelo CNN com um pré-processamento aplicado às imagens de deteção de limites. Desta forma, todas as imagens capturadas pela *webcam* são redimensionadas e é aplicado um filtro para detetar os limites existentes na imagem. De seguida, esta imagem é classificada e os três melhores resultados (por exemplo, letra A com 70%, letra V com 15% e letra X com 10%) são registados e exportados para um ficheiro.

9 Conclusões

Neste capítulo são apresentadas as diversas conclusões que se podem retirar de todo o processo de desenvolvimentos deste projeto. São apresentados os diversos objetivos realizados, as possíveis melhorias e trabalho futuro relativamente à solução desenvolvida e uma apreciação final e geral do projeto.

9.1 Objetivos realizados

Todos os objetivos definidos na fase inicial do projeto foram cumpridos com sucesso.

O trabalho de pesquisa desenvolvimento numa fase inicial permitiu identificar diferentes abordagens atualmente existentes e possibilitou a descoberta de ferramentas e tecnologias utilizadas para visão computacional. Estas abordagens serviram de inspiração para as fases de desenvolvimento, enquanto que exploração de tecnologias permitiu identificar quais as mais apropriadas para a obtenção de uma solução funcional. A reflexão realizada posteriormente e a comparação de alternativas, também possibilitou uma maior preparação e segurança no momento da implementação.

Tratando-se de tecnologias novas e sem grande experiência de trabalho, foi importante explorá-las e começar a realizar algumas experiências recorrendo à sua utilização. Estas foram sendo adaptadas ao contexto do projeto, progredindo com o desenvolvimento e implementação do mesmo. O processo de desenvolvimento foi sempre iterativo, começando naquelas que eram as primeiras etapas já identificadas, como a deteção e extração da mão, às etapas finais como a classificação do gesto. Também foi criado um *dataset* de raiz com o alfabeto da Língua Gestual Portuguesa, o que foi um processo demoroso e resultante em 3900 imagens distintas e devidamente nomeadas de acordo com a letra, autor e fundo.

O projeto não se limitou à criação de um único modelo de classificação, mas também explorou diferentes algoritmos de classificação existentes, indicados para diferentes áreas. Foram também explorados diferentes tipos de pré-processamento de imagem e os seus impactos no momento do treino e respetiva classificação. Com esta relação entre diferentes algoritmos e tipos de pré-processamento, foram efetuadas cerca de 48 avaliações de forma a encontrar o resultado ideal. Este componente de exploração foi muito demoroso, devido à extensão do *dataset*, do elevado tempo de processamento de alguns destes algoritmos e dos

recursos *hardware* disponíveis. No entanto, esta etapa foi muito importante pois permitiu identificar as alternativas com maior potencial.

O resultado foi uma solução capaz de, através de vídeo, extrair a informação relativamente à mão, com recurso a um modelo de visão computacional e de deteção de objetos, pré processá-la e consequentemente classificá-la através de um modelo de classificação.

Concluindo, esta solução é passo importante em direção ao objetivo futuro global, que é contribuir para a integração da comunidade surda na sociedade, atenuando os seus problemas de comunicação.

9.2 Melhorias e Trabalho Futuro

Apesar de os objetivos terem sido atingidos com sucesso, é importante concluir que certos aspetos podem ser melhorados.

Relativamente ao *dataset* criado para a classificação dos gestos, existe algum trabalho a fazer. Visto que o *dataset* atual apenas contém o alfabeto, seria importante expandir a gramática deste, ou seja, incluir outros conceitos da Língua Gestual Portuguesa, como números e palavras. Na procura de uma maior versatilidade, seria também importante refazer parte ou expandir o alfabeto já criado de forma a incluir diferentes autores, com diferentes padrões nas mãos, como rugas, ou tons de pele. De referir que uma alteração do *dataset*, implica sempre uma atualização ou recriação dos modelos já existentes.

Relativamente aos classificadores, alguns deles permitem uma grande personalização de parâmetros, possibilitando diferentes valores na procura do melhor modelo. Por exemplo, o classificador CNN, que foi um dos que apresentou melhores resultados, é constituído por diferentes camadas. Como tal, seria interessante experimentar outras estruturas e arquiteturas para este algoritmo e encontrar soluções que pudessem melhorar ainda mais as métricas deste algoritmo.

Por fim, também é possível melhorar os modelos de deteção de objetos (mãos). A criação destes modelos exige uma grande carga computacional, podendo levar dias para serem treinados. Este tempo de treino pode ser reduzido com recurso a placas gráficas NVIDIA, permitindo aumentar a velocidade com que os dados são processados. Com os recursos ao dispor para este projeto e sem uma placa gráfica, foi necessário utilizar o Google Colab que, apesar de disponibilizar um serviço *cloud* com placas gráficas, tem um tempo limite contínuo de

treino de 12 horas. Como tal, seria importante treinar os modelos criados para a deteção de mãos, utilizando idealmente um computador com placa gráfica potente, permitindo o tempo de treino necessário.

9.3 Apreciação Final

A apreciação final deste projeto é completamente positiva, pois proporcionou o desenvolvimento de um projeto inovador e com vista a um futuro melhor para a comunidade surda e todos os seus intervenientes.

Sem experiência na área da inteligência artificial e aprendizagem máquina, foi necessária uma grande investigação sobre as soluções existentes, o que permitiu adquirir os conhecimentos necessários para o desenvolvimento das diferentes fases deste projeto. Também o acompanhamento do orientador deste projeto foi um passo importante pois permitiu esclarecer possíveis dúvidas e questão, mantendo o seu desenvolvimento num bom caminho. Isto permitiu adquirir conhecimento na área da inteligência artificial, que está em grande evolução e que terá um papel extremamente importante na sociedade num futuro próximo.

Por fim, a solução criada e todo o estudo efetuado antes, durante e após o desenvolvimento da mesma, representa uma base muito importante para o objetivo que é eliminar algumas das dificuldade e barreiras que a comunidade surda sente no seu dia-a-dia e na sua vida em comunidade. Esta solução pode ser integrada com outras ferramentas, tal como se fez com o SignToText, possibilitando a existência de ferramentas mais completas que considerem não só os gestos, mas também os movimentos e expressões faciais do interlocutor.

10 Referências

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., … Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*, 265–283. https://tensorflow.org.

Alpaydin, E. (2014). *Introduction to machine learning*. 640. https://doi.org/10.4018/978-1-7998-0414-7.ch003

Associação Portuguesa de Surdos. (2012). *Associação Portuguesa de Surdos*. Associação Portuguesa de Surdos. http://apsurdos.org.pt/?page_id=3614

Ballard, D. H., & Brown, C. M. (1982). *Computer Vision* (p. 513).

Band, A. (2020). *How to find the optimal value of K in KNN?* Towards Data Science. https://towardsdatascience.com/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb

Bazarevsky, V., & Zhang, F. (2020). *Google AI Blog: On-Device, Real-Time Hand Tracking with MediaPipe*. https://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html

Belliveau, P., Griffin, A., & Somermeyer, S. (2002). *The PDMA ToolBook for New Product Development*. John Wiley & Sons, Inc.

blender.org. (2020). https://www.blender.org/

Canuma, P. (2018). *Image Pre-processing*. Towards Data Science. https://towardsdatascience.com/image-pre-processing-c1aec0be3edf

Chen, L. (2019). *Support Vector Machine — Simply Explained*. 1–12. https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496

Chollet, F., & & O. (2020). *Keras: the Python deep learning API*. Keras: the Python deep learning API. https://keras.io/

Coldewey, D. (2018). *SignAll is slowly but surely building a sign language translation platform | TechCrunch*. https://techcrunch.com/2018/02/14/signall-is-slowly-but-surely-building-a-sign-language-translation-platform/

Cunningham, P., & Delany, S. J. (2020). *k-Nearest Neighbour Classifiers: 2nd Edition (with Python examples)*. http://arxiv.org/abs/2004.04523

Dan. (2019). *Edge Detection in Opencv*. Sicara. https://www.sicara.ai/blog/2019-03-12-edge-detection-in-opencv

Diário de Notícias. (2016). Surdez - Comunidade surda quer fazer-se ouvir. *DN Sociedade*. https://www.dn.pt/sociedade/comunidade-surda-quer-fazer-se-ouvir-5407954.html

Dibia, V. (2019). *How to Build a Real-time Hand-Detector using Neural Networks (SSD) on Tensorflow*. Medium. https://medium.com/@victor.dibia/how-to-build-a-real-time-hand-detector-using-neural-networks-ssd-on-tensorflow-d6bac0e4b2ce

Django Starts. (2019). *Top Seven Apps Built With Python*. https://djangostars.com/blog/top-seven-apps-built-python/?utm_source=medium&utm_medium=hackernoon.com&utm_campaign=7appspython&utm_content=appsbuiltwithpython

Escudeiro, P., Escudeiro, N., Reis, R., Rodrigues, P., Lopes, J., Norberto, M., Bela Baltasar, A., Barbosa, M., & Bidarra, J. (2015). *Real Time Bidirectional Translator of Portuguese Sign Language*.

Facebook. (2020). *PyTorch*. https://pytorch.org/

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861–874. https://doi.org/10.1016/j.patrec.2005.10.010

Federação Portuguesa das Associações de Surdos. (2020). *Associações Filiadas | FPAS*. http://www.fpasurdos.pt/associacoes-filiadas/

Geiger, A., Lenz, P., & Urtasun, R. (2012). Are we ready for autonomous driving? the KITTI vision benchmark suite. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 3354–3361. https://doi.org/10.1109/CVPR.2012.6248074

GILT. (2020). http://gilt.isep.ipp.pt/

Gogul, I., & Kumar, V. S. (2017, Outubro 26). Flower species recognition system using convolution neural networks and transfer learning. *2017 4th International Conference on Signal Processing, Communication and Networking, ICSCN 2017*. https://doi.org/10.1109/ICSCN.2017.8085675

Gomes, J. F. (2019). *Governo lança aplicação que permite aos surdos chamar o 112 sem ajuda*. Observador. https://observador.pt/2019/07/15/governo-lanca-aplicacao-que-permite-

aos-surdos-chamar-o-112-sem-ajuda/

Google Colab. (2020). *Welcome to Colaboratory - Colaboratory*. Getting Started - Introduction. https://colab.research.google.com/notebooks/intro.ipynb#recent=true

Harrison, O. (2018). *Machine Learning Basics with the K-Nearest Neighbors Algorithm*. Towards Data Science. https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

Huang, T. S. (1997). Computer Vision: Evolution and Promise. *Report*.

Indiana University. (2020). *EgoHands: A Dataset for Hands in Complex Egocentric Interactions | IU Computer Vision Lab*. http://vision.soic.indiana.edu/projects/egohands/

Isabel Ferreira Batista, S., Maria João Barroso Pena, D., & Auxiliar Convidada, P. (2013). *A (In) Exclusão das pessoas surdas Mestre em Serviço Social Orientador(a)*.

Jacovi, A., Sar Shalom, O., & Goldberg, Y. (2019). *Understanding Convolutional Neural Networks for Text Classification*. 56–65. https://doi.org/10.18653/v1/w18-5408

Jung, J. K. (2020). *Training a Hand Detector with TensorFlow Object Detection API*. https://jkjung-avt.github.io/hand-detection-tutorial/

Kahn, K. B., Kay, S. E., Slotegraaf, R. J., & Uban, S. (2013). *The PDMA Handbook of new Product Development - 3rd Edition* (K. B. Kahn (Ed.); 3rd ed.). John Wiley & Sons, Inc.

Knocklein, O. (2019). Classification Using Neural Networks - Towards Data Science. Em *Towards Data Science* (pp. 1–3). https://towardsdatascience.com/classification-using-neural-networks-b8e98f3a904f

Luashchuk, A. (2019). 8 Reasons Why Python is Good for Artificial Intelligence and Machine Learning. Em *Djangostars*. https://djangostars.com/blog/why-python-is-good-for-artificial-intelligence-and-machine-learning/

Martins, N. L. (2005). *Classificação e partição de polígonos simples*. 145.

Mcfarlane, D. A. (2013). The strategic importance of customer value. *Atlantic Marketing Journal*, *2*(1), 62–75. http://digitalcommons.kennesaw.edu/amj%0Ahttp://digitalcommons.kennesaw.edu/amj/vol2/iss1/5

Mini Som. (2014). *Deficiência auditiva afeta um milhão em Portugal*. Mini Som. https://www.minisom.pt/artigos/deficiencia-auditiva-afeta-um-milhao-em-portugal

Missing Link. (2020). *Using the Keras Flatten Operation in CNN Models with Code Examples - MissingLink.ai*. https://missinglink.ai/guides/keras/using-keras-flatten-operation-cnn-models-code-examples/

MySQL. (2020). https://www.mysql.com/

Nikam, A. S., & Ambekar, A. G. (2017a). Bilingual sign recognition using image based hand gesture technique for hearing and speech impaired people. *Proceedings - 2nd International Conference on Computing, Communication, Control and Automation, ICCUBEA 2016*, 1–6. https://doi.org/10.1109/ICCUBEA.2016.7860057

Nikam, A. S., & Ambekar, A. G. (2017b). Sign language recognition using image based hand gesture recognition techniques. *Proceedings of 2016 Online International Conference on Green Engineering and Technologies, IC-GET 2016*, 1–5. https://doi.org/10.1109/GET.2016.7916786

OpenCV. (2020a). *About*. https://opencv.org/about/

OpenCV. (2020b). *OpenCV: Smoothing Images*. https://docs.opencv.org/3.4/d4/d13/tutorial_py_filtering.html

Ostertagová, E., Ostertag, O., & Kováč, J. (2014). Methodology and application of the Kruskal-Wallis test. *Applied Mechanics and Materials*, *611*(August 2014), 115–120. https://doi.org/10.4028/www.scientific.net/AMM.611.115

Osterwalder, A., Pigneur, Y., Smith, A., & Movement, T. (2010). *Business Model Generation*.

Pavlovic, V. I., Sharma, R., & Huang, T. S. (1997). Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19*(7), 677–695. https://doi.org/10.1109/34.598226

Puget, J. F. (2016). *The Most Popular Language For Machine Learning Is ... (IT Best Kept Secret Is Optimization)*. https://www.ibm.com/developerworks/community/blogs/jfp/entry/What_Language_Is_Best_For_Machine_Learning_And_Data_Science?lang=en

Ren, Q., Li, M., & Han, S. (2019). Tectonic discrimination of olivine in basalt using data mining techniques based on major elements: a comparative study from multiple perspectives. *Big Earth Data*, *3*(1), 8–25. https://doi.org/10.1080/20964471.2019.1572452

Ribeiro, M., & Gomes, A. (2010). Adaptação de Cor para Dicromatas na Visualização de Imagens. *Encontro Português de Computação Grafica*, *October 2012*.

Rish, I. (2014). *An Empirical Study of the Naïve Bayes Classifier*. *January 2001*, 41–46.

Rokach, L., & Maimon, O. (2006). Decision Trees. Em *Data Mining and Knowledge Discovery Handbook* (pp. 165–192). Springer-Verlag. https://doi.org/10.1007/0-387-25465-x_9

Saaty, T. L. (1990). How to make a decision: The analytic hierarchy process. *European Journal of Operational Research*, *48*(1), 9–26. https://doi.org/10.1016/0377-2217(90)90057-I

Scikit-Learn. (2020a). *Neural network models*. https://scikit-learn.org/stable/modules/neural_networks_supervised.html

Scikit-Learn. (2020b). *Scikit-learn: machine learning in Python*. https://scikit-learn.org/stable/

Sebe, N., Cohen, I., Garg, A., & Huang, T. S. (2005). Machine Learning in Computer Vision. Em *Machine Learning in Computer Vision*. Springer-Verlag. https://doi.org/10.1007/1-4020-3275-7

Sidana, M. (2017). Intro to types of classification algorithms in Machine Learning. *Sifium*, 2–5. https://medium.com/sifium/machine-learning-types-of-classification-9497bd4f2e14

Sidorchuk, R. (2015). The concept of "value" in the theory of marketing. *Asian Social Science*, *11*(9), 320–325. https://doi.org/10.5539/ass.v11n9p320

Surdos Oralizados Portugueses. (2020). *Escolas de Referência - Surdos Oralizados Portugueses*. http://surdosoralizadosportugueses.eu/escolas-de-referencia/

Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. http://szeliski.org/Book/.

Tensorflow. (2020a). *Introduction to Tensors*. https://doi.org/10.1007/3-540-27066-3_2

Tensorflow. (2020b). *TensorFlow*. https://www.tensorflow.org/

TensorFlow. (2020). *TFRecord and tf.Example*. https://www.tensorflow.org/tutorials/load_data/tfrecord?hl=en

Trigueiros, P., Ribeiro, F., & Reis, L. P. (2012). A comparison of machine learning algorithms applied to hand gesture recognition. *Iberian Conference on Information Systems and Technologies, CISTI*, *January 2014*.

Unity Real-Time Development Platform | 3D, 2D VR & AR Engine. (2020). Unity Technologies. https://unity.com/

University of Oxford. (2014). *Hand Dataset*. http://www.robots.ox.ac.uk/~vgg/data/hands/

Vasani, D. (2019). *How do pre-trained models work?* Towards Data Science. https://towardsdatascience.com/how-do-pretrained-models-work-11fe2f64eaa2

Venners, B. (2013). *The Making of Python: Part I*. A Conversation with Guido van Rossum, Part I. https://www.artima.com/intv/python.html

Verma, K. (2010). A Theory Based on Conversion of RGB image to Gray image. *Article in International Journal of Computer Applications*. https://doi.org/10.5120/1140-1493

VIRTUALSIGN. (2020). http://193.136.60.223/virtualsign/pt/index.php

wikimedia. (2020). *File:Hempharvesting2.jpg - Wikimedia Commons*. https://commons.wikimedia.org/wiki/File:Map1NNReducedDataSet.png

Yegulalp, S. (2019). What is TensorFlow? The machine learning library explained. *InfoWorld*. https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html

Zhang, Q., Zhao, R., Wang, D., & Yu, Y. (2019). MyoSign: Enabling end-to-end sign language recognition with wearables. *International Conference on Intelligent User Interfaces, Proceedings IUI*, *Part F1476*(March), 650–660. https://doi.org/10.1145/3301275.3302296

Zhang, Z. (2019). *Naive Bayes Explained - Towards Data Science*. https://towardsdatascience.com/naive-bayes-explained-9d2b96f4a9c0