SignPic: Sistema Móvel Para Deteção de Língua Gestual Utilizando *Machine Learning*

André Nogueira

Faculdade de Ciências e Tecnologia

Universidade Fernando Pessoa

Uma tese submetida para obtenção do grau de

Master of Science

2020

Resumo

O objetivo do trabalho proposto nesta Dissertação assenta na contribuição para colmatar a barreira de comunicação existente entre pessoas que comunicam utilizando a Língua Gestual Portuguesa e pessoas que comunicam utilizando a língua oral. Na língua gestual, a forma, o posicionamento e o movimento das mãos, bem como as expressões faciais e os movimentos corporais, desempenham papéis importantes para as pessoas comunicarem entre si. A motivação subjacente à escolha deste tema, centra-se na dificuldade existente na comunicação entre as pessoas que compreendem e utilizam a Língua Gestual Portuguesa e as pessoas que comunicam apenas em língua oral, impedindo que os gestos em Língua Gestual Portuguesa sejam corretamente compreendidos pelas pessoas ouvintes. Devido a não ter sido encontrado conjunto de dados referentes ao alfabeto da Língua Gestual Portuguesa para utilização no desenvolvimento do sistema, foi utilizado um conjunto de dados relativo ao alfabeto da Língua Gestual Americana. Neste contexto, optou-se por desenvolver uma aplicação que permite identificar e traduzir os gestos em Língua Gestual Americana, efetuados por uma pessoa utilizando uma câmara de vídeo convencional, presente na maioria dos *smartphones* atuais. Para atingir este objetivo, o *software* desenvolvido recorreu a técnicas de Inteligência Artificial designadas por *Deep Learning*. Para o treino do conjunto de dados foi utilizado um modelo pré treinado presente no *Zoo* de modelos disponibilizados pela biblioteca *PyTorch*, denominado *MobileNet v2*. A avaliação ao sistema proposto foi efetuada a partir de estatísticas guardadas ao longo dos treinos efetuados a vários modelos classificadores e também, a partir de testes, utilizando a câmara de um *iPhone* para obter as imagens que posteriormente foram classificadas no mesmo. Concluiu-se que com o sistema desenvolvido, apesar de o classificador ter atingido 99% de acurácia, durante a validação, ainda está longe de ser um sistema capaz de colmatar a barreira de comunicação entre pessoas que comunicam utilizando Língua Gestual Americana e as pessoas que comunicam usando língua oral.

Abstract

The objective of the work proposed in this Dissertation is to contribute to establishing a communication bridge between people who communicate using the Portuguese Sign Language and people who communicate using the oral language. In sign language, the shape, positioning and movement of the hands, as well as facial expressions and body movements, play an important role for people to communicate with each other. The motivation underlying the choice of this theme focuses on the difficulty existing in communication between people who understand and use Portuguese Sign Language and people who communicate only with oral language, preventing gestures in Portuguese Sign Language from being correctly understood by hearing people. Due to the lack of a dataset related to the alphabet of the Portuguese Sign Language to use in the development of the system, a dataset containing the alphabet of the American Sign Language was used. In this context, it was decided to develop an application that allows the identification and translation of gestures in American Sign Language, carried out by a person using a conventional video camera, present in most current *smartphones*. To achieve this goal, the *software* developed fall back on Artificial Intelligence techniques called *Deep Learning*. To train the dataset, a pre-trained model called *MobileNet v2* was used. This model is present in the *Zoo* of models made available by the library *PyTorch*. The evaluation of the proposed system was conducted based on statistics saved during the training carried out on several classifying models and also, based on tests, using a *iPhone* camera to obtain the images that were later classified on it. It was concluded that with the developed system, although the classifier reached 99% accuracy, during the validation, it is still far from being a system capable of overcoming the communication barrier between people who communicate using American Sign Language and the people who communicate using oral language.

Em primeiro lugar, gostaria de dedicar esta dissertação à minha família, em especial à minha mãe pelo seu apoio contínuo e força motivadora e, por me ter facultado as condições e oportunidade de, depois da licenciatura, conseguir obter e concluir o mestrado.

Agradecimentos

Um agradecimento aos Professores Doutor Christophe Soares (orientador) e Doutor Rui Moreira (co-orientador) pelo apoio científico, pela competência pedagógica e caráter humano.

Por detrás do trabalho pessoal realizado, existiram apoios, sugestões, comentários e críticas vindas de pessoas próximas. Foram também estas contribuições que me levaram a seguir em frente e concluir esta etapa da minha formação.

A todos o meu muito e sincero obrigado.

Índice

Índice de Figuras

Índice de Tabelas

Lista de Acrónimos

ANN Rede Neuronal Artificial (Artificial Neural Network)

API Interface de Programação de Aplicações (Application Programming Interface)

ASL Língua Gestual Americana (American Sign Language)

CLAP14 ChaLearn Looking at People 2014

CNN Rede Neuronal Convolucional (Convolutional Neural Network)

CUDA Compute Unified Device Architecture

DL Deep Learning

DNN Deep Neural Network

ESL Língua Gestual Inglesa (English Sign Language)

FPGA Field-programmable Gate Array

GMM-HMM Gaussian Mixture Model-Hidden Markob Model

HCI Interação Homem-Máquina (Human-Computer Interaction)

iOS Sistema operativo do iPhone

IoT Internet das Coisas (Internet of Things)

JS Javascript

LGP Língua Gestual Portuguesa

LP Língua Portuguesa

LSTM Long-Short Term Memory

ML Machine Learning

PIL Python Imaging Library

pip Gerenciador de dependências Python

ReLU Rectified Linear Units

RNN Rede Neuronal Recorrente (Recurrent Neural Network)

SGD Stochastic Gradiend Descent

SLR Reconhecimento de Língua Gestual (Sign Language Recognition)

TF Tensorflow

TF.js Tensorflow JS

UI Interface do utilizador (User Interface)

VRS Video Relay Service

Capítulo 1

Introdução

Durante a última década, a tecnologia tem vindo a evoluir a uma velocidade estonteante. Apesar dos avanços tecnológicos, a interação entre o ser humano e o computador não evoluiu à mesma velocidade. As técnicas de interação entre o ser humano e a máquina mantêm-se muito à base de um rato e de um teclado, incluindo comandos ou telas de toque.

Para Chao, os sistemas Homem-Máquina (HCI) referem-se ao sistema composto por humanos e máquinas para suprir algumas funções através da interação entre humanos e computadores (Chao, 2009).

Segundo Taskiran, a Organização Mundial da Saúde anunciou que o número de deficientes auditivos chegou recentemente a 400 milhões (Taskiran et al., 2018).

Pessoas com deficiências, tais como a surdez ou pessoas mudas, possuem uma enorme dificuldade em comunicar com a sociedade em geral, devido à ausência de meios para que a comunicação entre as pessoas que utilizam a Língua Gestual Portuguesa (LGP) e as pessoas que utilizam a língua oral seja feita de uma forma fluente.

Hoje em dia, com o desenvolvimento tecnológico, juntamente com o aumento do poder de processamento, é possível criar aplicações capazes de melhorar a vida destes indivíduos.

1.1 Problema

O problema, em geral, é a barreira existente na comunicação entre pessoas que utilizam a LGP e as pessoas que utilizam a língua oral como meio de comunicação, colocando, de certa forma, à parte, as pessoas que comunicam através de gestos. Estas pessoas têm dificuldade em transmitir os seus pensamentos às pessoas que utilizam a língua oral. E por sua vez, a língua oral nunca será percecionada de forma plena pela pessoa surda, uma vez que esta só poderá desenvolver as capacidades cognitivas proporcionadas por uma língua, quando exposta a uma língua espácio-visual.

O facto de se pretender desenvolver uma aplicação móvel onde todo o fluxo, presente na Figura 1.1, desde o conteúdo proveniente da câmara de um *smartphone*, passando pelo pré-processamento de imagem e reconhecimento, seja feito no próprio dispositivo, traz uma maior dificuldade computacional. Apesar da evolução da tecnologia, as técnicas que permitem que tal sistema seja desenvolvido ainda são bastante recentes e incipientes. Evidências disto são, por exemplo, a biblioteca *Mediapipe*, que apenas foi disponibilizada ao público em meados do mês de Agosto de 2019, bem como a versão 3 do *"Core ML"* da *Apple* que também permite o processamento no próprio dispositivo, mas esta está limitada ao ecossistema da *Apple*.

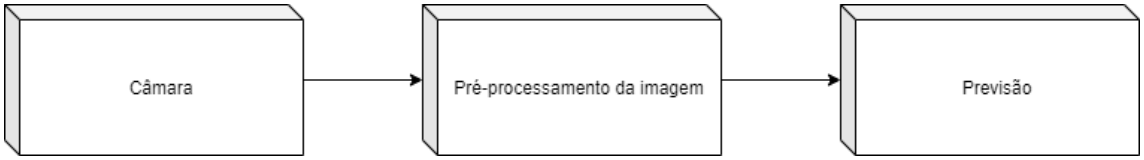

Figura 1.1: Fluxo dos dados até à etapa de reconhecimento do gesto

1.2 Objetivos

O objetivo a alcançar inclui o desenvolvimento de uma aplicação móvel, que permita que a comunicação entre pessoas que utilizem LGP e língua oral, seja efetuada de uma maneira simples e eficaz. Para isso, é necessário que a aplicação detete e reconheça os gestos efetuados por uma pessoa e os traduza para texto.

Para atingir o objetivo proposto anteriormente, a utilização da biblioteca *Mediapipe* pareceu ser uma opção viável devido ao facto de retirar a maior parte da complexidade na deteção dos gestos efetuados pelas mãos de um utilizador. Esta é apoiada e desenvolvida pela Google, direcionada para plataformas móveis, permitindo-nos que o processamento seja efetuado no dispositivo móvel. E é este facto que traz enormes benefícios, tais como não ser necessário estabelecer uma comunicação com a Internet para o processamento ser efetuado através da *cloud*.

Mas com o avançar da investigação, foi decidido alterar a biblioteca utilizada para desenvolver o produto final. Optou-se por uma biblioteca mais vocacionada para a investigação, que possuísse uma *API* mais amigável para o desenvolvedor, o *PyTorch*. A mesma possui código fonte aberto, e é utilizada por várias empresas como a *Udacity*, *Salesforce* e também em Universidades como *Stanford*. A par disto, a biblioteca também disponibilizou recentemente compatibilidade com plataformas móveis, tanto para *Android* como para *iOS*.

A entrada dos dados, como se pode verificar na Figura 1.1, ocorre utilizando os dados provenientes de uma câmara que posteriormente serão pré-processados, englobando

um redimensionamento da imagem, no caso de esta não possuir a resolução de 224x224 pixeis. Após o redimensionamento ser efetuado com sucesso, procede-se à respetiva interpretação para texto.

O desenvolvimento do projeto será feito de uma forma incremental, isto é, primeiramente, pretende-se implementar a deteção de um gesto único e estático, como, por exemplo, a letra A. Após isso, pretende-se proceder à adição de mais gestos ao classificador. Para a adição dos gestos adicionais será utilizado um conjunto de dados proveniente da plataforma *Kaggle*, que já os possui dividido por classes, específicos da Língua Gestual Americana (ASL); e não à realidade pretendida nos objetivos desta dissertação. Apesar deste constrangimento, alguns gestos são partilhados entre os restantes Idiomas. O conjunto de dados em questão apenas possui o alfabeto, neste caso americano.

1.3 Justificação

O tema escolhido para esta Dissertação de Mestrado tem como propósito resolver um problema de comunicação que afeta membros da nossa sociedade e cuja resolução se revela de extrema importância.

A solução atual prende-se com a utilização de um *Video Relay Service* (VRS), que consiste na introdução de uma pessoa mediadora na comunicação para a tradução da conversa entre as partes. Apesar de ser a mais viável em termos de tradução, não é muito conveniente, visto ser necessário estabelecer uma vídeo chamada com a pessoa mediadora, e essa também necessita de estabelecer uma chamada com o destinatário. Este tipo de serviço, para além de ser custoso devido ao tempo, que é perdido no estabelecimento das duas chamadas e à devida tradução por parte do mediador, também arrecada custos.

1.4 Estrutura do trabalho

O presente trabalho é composto por 6 capítulos. No Capítulo 1 é introduzida a temática abordada, o problema para o qual se pretende encontrar solução, o objetivo que se pretende atingir, assim como a justificação da escolha do tema.

Durante o Capítulo 2, é apresentado o estado da arte. Aqui é apresentada a língua gestual (LG), diversas técnicas utilizadas para a deteção de gestos presentes nesta língua, assim como vários, dispositivos, ferramentas e bibliotecas utilizadas para atingir os objetivos referidos na Secção 1.2.

Segue-se o Capítulo 3, onde é especificada, a forma como se vai proceder ao desenvolvimento do trabalho, assim como a arquitetura referente ao classificador e à aplicação móvel, como a proposta do modelo desenvolvido. Adicionalmente são apresentados os requisitos funcionais e não funcionais, e é apresentado todo o desenvolvimento do sis-

tema, desde a sua criação, treino e teste do modelo para classificação dos mais variados gestos presentes no alfabeto *ASL*, até ao desenvolvimento da aplicação para dispositivos *Apple*. Neste caso, para o *iPhone*, onde é utilizado o modelo classificador referenciado anteriormente, para a previsão dos gestos capturados pela própria câmara do telemóvel.

Prosseguindo, no Capítulo 4, encontramos os testes e resultados do progresso efetuado no Capítulo 3, onde é feita uma análise aos dados estatísticos guardados.

Por fim, no Capítulo 5, são apresentadas as conclusões referentes ao desenvolvimento do modelo proposto, bem como da aplicação móvel que utiliza o modelo classificador para a previsão do gesto efetuado.

Capítulo 2

Reconhecimento de Língua Gestual

Neste capítulo é apresentado o conceito de LG e são referenciados trabalhos efetuados na área da *Sign Language Recognition* (SLR). Relativamente à SLR são mencionadas várias técnicas e propostas que permitem atingir um objetivo comum, a deteção de gestos.

Serão também apresentados diferentes tipos de redes neuronais, diversas bibliotecas para tratamento de imagem e vídeo, bem como bibliotecas vocacionadas para execução de modelos classificadores no próprio dispositivo.

2.1 Língua Gestual

Desde as primeiras manifestações de vida humana que houve a necessidade do homem comunicar. Esta necessidade inata de interagir e relacionar-se com elementos da sua comunidade, aliada à necessidade das trocas comerciais entre povos, foram determinantes para o aparecimento de diferentes tipos de comunicação. Se inicialmente a comunicação era exercida através de sons, sinais ou figuras desenhadas, esta sofreu uma evolução natural, motivada pelo desenvolvimento cerebral do indivíduo, pelas suas movimentações e pelos contactos sociais que estabelecia (Stokoe, 2006).

Comunicar pressupõe que exista partilha de experiências de vida entre sujeitos que possam ou não conhecer-se, que possam ter ou não uma identidade comum, que façam parte de um ambiente multicultural ou intercultural e onde possam ou não existir limitações na forma de comunicar, as quais podem ser motivadas por problemas *"de natureza sensorial, cognitiva, neurogénica, intelectual, psíquica, motora e outras"* (Guerreiro, 2012).

A linguagem tem a capacidade de organizar a forma de pensar, de modo a que este possa ser transmitido através da língua falada e suportando-se pela construção de conteúdos com carácter criativo e linguístico (Chiari, 2014) e assume duas funções: o intercâmbio social e o pensamento generalizador (Vygotsky, 1993).

A linguagem é formada por dois aspetos: a língua e a fala. A língua é a parte social

da linguagem, algo que o sujeito partilha com os membros da sua comunidade e que pode ser estudada. Já a fala compreende as características individuais que cada sujeito utiliza quando formula o seu discurso na interação com os outros (Saussure, 1978).

Sendo a linguagem o meio através do qual o ser humano comunica as suas ideias e sentimentos, esta assume um papel determinante na comunicação, podendo ser concretizada através da fala, da escrita ou na utilização de sinais.

Para Sim-Sim, a produção de sons de uma língua, a aprendizagem de novas palavras e a utilização e compreensão das regras da gramática são insuficientes para permitir a aquisição e progressão da linguagem. Estes autores referem-se a ela como *"um processo complexo e fascinante em que a criança, através da interacção com os outros, (re)constrói, natural e intuitivamente, o sistema linguístico da comunidade onde está inserida, i.e., apropria-se da sua língua materna."* (Sim-Sim et al., 2008). Esta é adquirida de uma forma natural e nos primeiros anos de vida (Sim-Sim et al., 1997).

Ao aprenderem a língua materna, as crianças na interação com outras pessoas vão conseguir desenvolver as suas competências para comunicar e, à medida que vão adquirindo a linguagem, vão construindo o seu próprio conhecimento, num processo holístico onde os diferentes aspetos que constituem a linguagem (função, forma e significado) vão ser percebidas em simultâneo (Sim-Sim et al., 2008).

Cada país tem a sua língua de adoção, a língua falada e esta é partilhada pelos membros falantes de uma comunidade que a utiliza e que representa o aspeto social da linguagem (Saussure, 1978). No entanto, existem pessoas que por serem surdas, deficientes auditivos, ou com deficiências ao nível da sua oralização, não sabem (porque nunca aprenderam), não conseguem (porque têm problemas na oralização) ou têm dificuldades em utilizá-la (porque não a praticam). Estes indivíduos, à semelhança de todos os outros, têm necessidade de comunicar.

A LG, é a língua utilizada pela Comunidade Surda e por muitos indivíduos que com eles interagem, como familiares, amigos, professores, educadores, técnicos, intérpretes, entre outros. A LG é reproduzida através de gestos, como forma de estes indivíduos poderem efetuar transmissão de sentimentos e pensamentos, de forma visual, e serem percebidos. É utilizada a ação das mãos, braços, tronco, face e cabeça. A diferença para a língua oral é que esta utiliza palavras e são produzidos sons, que são percebidos através dos ouvidos (Liddell, 2003).

Não existe uma data exata para o seu aparecimento, mas Armstrong e Wilcox referem que a origem da língua de gestos ocorreu simultaneamente com a origem das línguas humanas (Armstrong and Wilcox, 2007).

Esta língua de gestos, em Portugal, é denominada de LGP. Não é uma língua de utilização universal, cada país tem uma LG própria que espelha a cultura e o modo de vida da comunidade surda desse país.

A Constituição da República Portuguesa reconheceu a LGP *"enquanto expressão cul-*

tural e instrumento de acesso à educação e da igualdade de oportunidades" a 15 de novembro de 1997 (LC 1/97, Artigo 74o, Alínea h), data em que se assinala o Dia Nacional da LGP. A partir de então, o cidadão surdo teve a vida mais facilitada por via da lei.

No entanto, nem sempre foi assim. Almeida refere que existiram vários momentos na comunicação da Comunidade Surda: o oralismo, a comunicação total e o bilinguismo (Almeida, 2014). O oralismo correspondeu a uma fase em que os defensores desta teoria proibiram a utilização da LG porque acreditavam que a linguagem se resumia à língua oral. Esta era a única forma de comunicação aceite e por esse motivo esta língua era introduzida, desde muito cedo, às crianças surdas para que pudessem aprender e desenvolvê-la. Com a evolução dos tempos, seguiu-se a fase denominada de comunicação total, que compreendia a adoção de estratégias envolvendo a audição, as mãos e a oralização e que incentivavam a comunicação entre indivíduos surdos e entre estes e indivíduos ouvintes.

Com a implementação do bilinguismo, o indivíduo surdo passou a utilizar a LG, a sua primeira língua, para poder comunicar, motivada pela legislação que promove a inclusão do surdo na sociedade onde pertence, e a língua oficial do país, a língua escrita e eventualmente oral, como segunda língua.

A LG não deve ser considerada como menos legítima do que a língua oral e o indivíduo surdo não deve sentir-se inferiorizado em utilizá-la. Trata-se de uma língua reconhecida pela lei em Portugal à semelhança do Mirandês e da língua oficial do país, a Língua Portuguesa (LP).

Esta língua possui um léxico e uma gramática própria, que a distingue da LP. Através da forma da mão, do lugar ocupado por esta, da sua orientação, pela direção e velocidade do movimento executado, pelas expressões faciais e pela forma como movimentam o corpo, conseguem criar uma imagem precisa da ideia exprimida.

Relativamente à organização de frases, verifica-se com frequência que na LGP surge em primeiro lugar o Objeto, seguido do Sujeito e no final o Verbo (O-S-V) enquanto na LP as frases são compostas primeiro pelo Sujeito, seguido do Verbo e no final o Objeto (S-V-O) (Amaral et al., 1994).

Pese embora a valorização da LGP e do seu reconhecimento à luz da lei, os surdos continuam a ser "esquecidos" a nível da sua educação pelo Estado Português. Apesar de existirem escolas que dão aulas em LGP a indivíduos desta comunidade, outras não o fazem permanecendo a dúvida se a LGP será vista como dispensável ou se haverá uma pressão para que estes indivíduos sejam oralizados, de modo a que aprendam uma segunda língua, neste caso a língua oficial do país.

De acordo com Souza, os indivíduos surdos que conseguem aprender, através da terapia da fala, a língua oral, apesar de se sentirem mais integrados na sociedade fazem uma interpretação do mundo a partir da prática adquirida anteriormente da língua de gestos (Souza, 1997).

Skliar evidencia que nem todos os surdos dominam a LG, pelo que vêm a sua identidade desagregada, uma vez que não se sentem integrados na sua plenitude, nem na comunidade surda nem na comunidade dos ouvintes. Este autor partilha da opinião que existindo as duas línguas gestual e oral *"cada uma poderia continuar correspondendo a dois grupos de pessoas diferentes e a duas ou mais imagens do mundo."*, e que o sistema de educação deveria ser revisto afim de que esta comunidade pudesse *"enfrentar a escola, com a existência das diferenças, de outras formas e processos de identidade de linguagem e de cognição"* (Skliar, 1997).

O indivíduo surdo, regularmente denominado de *"pessoa portadora de deficiência"* tem vindo a manifestar o seu descontentamento face a esta conotação. Consideram-se pessoas diferentes, com uma cultura e língua próprias e que tem o direito de recusar a utilização do discurso oral ou escrito, se assim o pretender. Reivindicam uma sociedade onde haja tolerância, solidariedade, multiculturalismo, igualdade de oportunidades e não discriminação da língua que utilizam, a LG à semelhança do que o artigo 13º da Constituição da República Portuguesa refere de *"que todos os cidadãos têm a mesma dignidade social e são iguais perante a lei"*, não podendo ninguém *"ser privilegiado, beneficiado, prejudicado, privado de qualquer direito ou isento de qualquer dever em razão da língua"*.

Infelizmente, os direitos das pessoas surdas são frequentemente esquecidos ou negados, devido a preconceitos sociais e a presunções erradas, especialmente em países em desenvolvimento (World Federation of the Deaf, s.d.).

Nos Estados Unidos da América, a língua gestual adotada (ASL) não é reconhecida legalmente em todos os seus 50 Estados, apenas o sendo em 45. Estados como *Hawai*, *Mississippi*, *Missouri*, *New Hampshire* e *Wyoming*, ainda não conferiram à língua de gestos legitimidade para o seu uso, por via legal (National Association of the Deaf, 2016).

À semelhança de vários países, também os Estados Unidos da América possuem um património cultural próprio, do qual faz parte a ASL. A razão pela qual é denominada de *"American Sign Language"* é pelo facto de ser nativa dos Estados Unidos da América e de ser uma língua utilizada pelos habitantes da sua Comunidade Surda.

Como nos Estados Unidos da América se fala o Inglês, poderia supor-se que todos os países do mundo onde o Inglês fosse língua oficial, utilizariam a mesma língua de gestos. Este pensamento está errado. As línguas gestuais não dependem das línguas dos seus países. Por exemplo, a ASL apresenta mais semelhanças com a Língua Gestual Francesa do que com a Língua Gestual Britânica (Associação de Surdos do Porto, s.d.).

Distinta da LGP, a organização básica de frases em ASL, inicia-se com o Sujeito, seguido do Verbo e por fim o Objeto, numa ordem básica (S-V-O) (Sandler and Lillo-Martin, 2006).

De modo idêntico, o alfabeto é representado de forma diferente nestas duas línguas, como se pode verificar nas Figuras 2.1 e 2.2.

Figura 2.1: Alfabeto LGP (Wikipédia, s.d.)

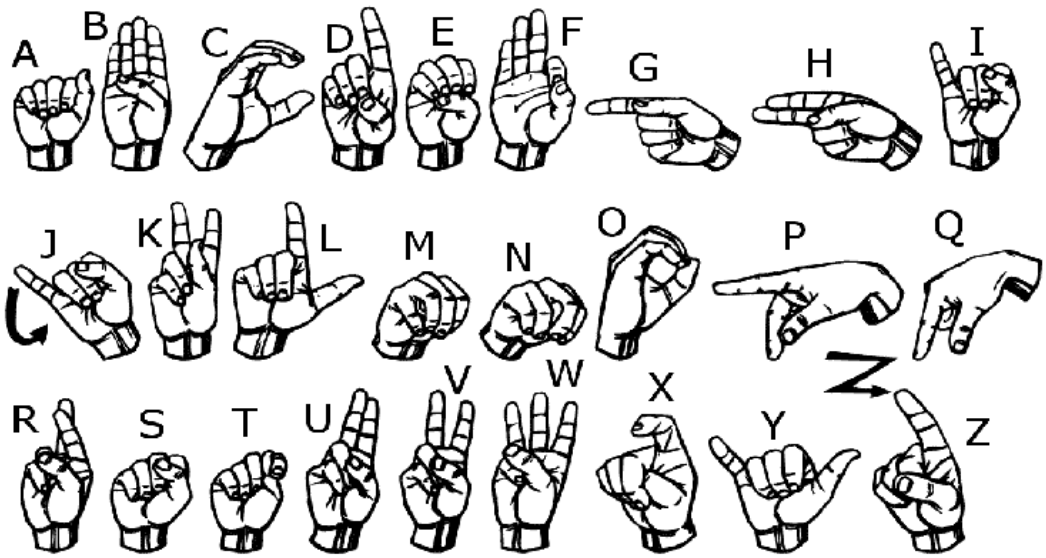

Figura 2.2: Alfabeto ASL (Chong and Lee, 2018)

2.2 *Sign Language Recognition*

Investigadores têm vindo a trabalhar na área de SLR ao longo dos anos, visto esta ser uma forma de contornar a barreira na comunicação entre cidadãos que utilizam a LG e os que utilizam a língua oral. A automatização de SLR assume uma importância considerável, na medida em que explora soluções tecnológicas para eliminar ou mitigar barreiras comunicacionais. Esta é uma área de investigação inclusiva porque procura melhorar a qualidade de vida dos cidadãos, através do aperfeiçoamento e facilitação dos meios de comunicação.

Nos últimos anos foram-se desenvolvendo vários projetos e tecnologias, cuja aplicação pode eliminar ou suavizar diversos obstáculos inerentes às dificuldades de comunicação através de LG.

Das mais variadas abordagens para automatizar a identificação de sinais gestuais, destacam-se duas. Uma, baseada na utilização de equipamentos como *Data Gloves*, que utilizam vários tipos de sensores incorporados na própria luva e que guardam informações provenientes dos sensores (e.g. orientação da mão, forma da mão, velocidade com que a mão se desloca, etc.), tornando-se numa abordagem, tipicamente, mais dispendiosa. A outra abordagem, mais comum devido ao seu menor custo, utiliza técnicas *state of the art* de visão computacional para segmentação, rastreamento e extração de *features* para a posterior identificação de gestos característicos da LG.

As abordagens baseadas em visão, para além de mais económicas do que as primeiras, proporcionam uma forma mais natural de interação entre o homem e a máquina. Apesar destas vantagens, utilizar imagens ou vídeos para classificar gestos, não é uma tarefa fácil, porque existem múltiplos obstáculos que são necessários ultrapassar. Por exemplo, o facto de nem todos todos os gestos serem estáticos, uma vez que existem também gestos dinâmicos. Nesses, as mãos podem-se sobrepor e dificultar o seu processamento. Existem ainda diferentes cores de pele e variações de luminosidade nos mais variados ambientes, que colocam problemas à análise automática de imagens para a classificação de LG.

No projeto realizado por Park, recolheram-se dados provenientes de uma *Data Glove*, posteriormente analisados por um módulo FPGA. Este módulo permite identificar os dados referentes aos diferentes gestos. O resultado do reconhecimento é exibido visualmente num ecrã. Os autores concluem na sua avaliação, que nos dezassete tipos diferentes de gestos testados, conseguem atingir uma taxa de acurácia média de 94% (Park et al., 2008).

Os autores Krishnam e Balasubramanian utilizam um conjunto de dados provenientes da plataforma *Kaggle* (Rasband, 2018), referente à Língua Gestual Inglesa (ESL). Estes dados correspondem a imagens que representam os diferentes gestos do alfabeto da ESL, e a alguns símbolos. Deste conjunto de dados, 810 imagens foram utilizadas para treinar uma *Deep Neural Network* (DNN). Cada imagem possui inicialmente dimensões de 200

por 200 pixeis e foram posteriormente redimensionadas para 128 por 128 pixeis (Krishnan and Balasubramanian, 2019). A DNN proposta pode ser visualizada na Figura 2.3.

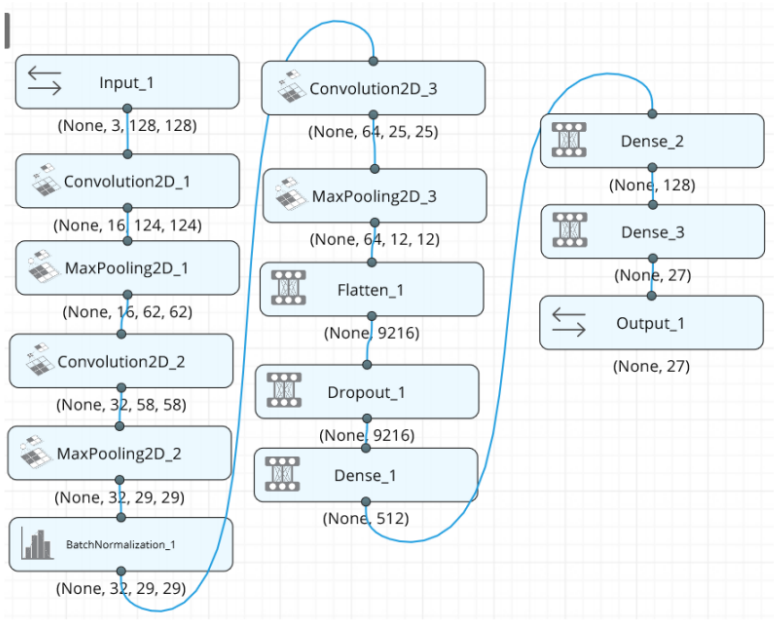

Figura 2.3: DNN (Krishnan and Balasubramanian, 2019)

Neste trabalho utilizaram uma rede neuronal convolucional (CNN) 2D para reconhecimento de gestos efetuados pelas mãos do utilizador. A camada de entrada possui a dimensão (3, 128, 128). A primeira convolução consiste em 16 filtros de tamanho (5,5), seguindo-se uma camada de *Max-polling* de tamanho (2,2). Ocorre novamente uma convolução, seguida de outro *Max-polling*. Segue-se uma camada denominada de *Batch normalization*, que é utilizada para normalizar os pesos entre as camadas convolucionais. Após nova convolução e *Max-polling*, ocorre *flatenning* e, de seguida os pesos são decrementados num fator de 0.4. Por fim, são utilizadas três blocos de camadas densas. Para o treino deste sistema de reconhecimento de gestos das mãos, foram utilizadas 20 épocas com um tamanho de *batch* de 4. Como função de perda, foi utilizada a *Cross-Entropy*, e como função de otimização utilizou-se *Stochastic Gradiend Descent* (SGD) com uma taxa de aprendizagem de 0.01 e um *momentum* de 0.5. Os autores concluem que foi possível atingir uma acurácia de teste de 70% (Krishnan and Balasubramanian, 2019).

A proposta de Taskiran, consiste numa CNN com uma camada de entrada, duas camadas de convolução 2D, seguida de *pooling* e *flattening*, e por fim duas camadas densas, como se pode visualizar na Figura 2.4. O conjunto de dados foi submetido a um pré-processamento de redimensionamento das imagens para 28 por 28 pixeis, e posteriormente convertidas para escala de cinzento (Taskiran et al., 2018).

Os autores também referem que efetuam uma deteção da cor da pele para seccionar a região onde está presente o gesto, utilizando o *OpenCV*, mais concretamente a função

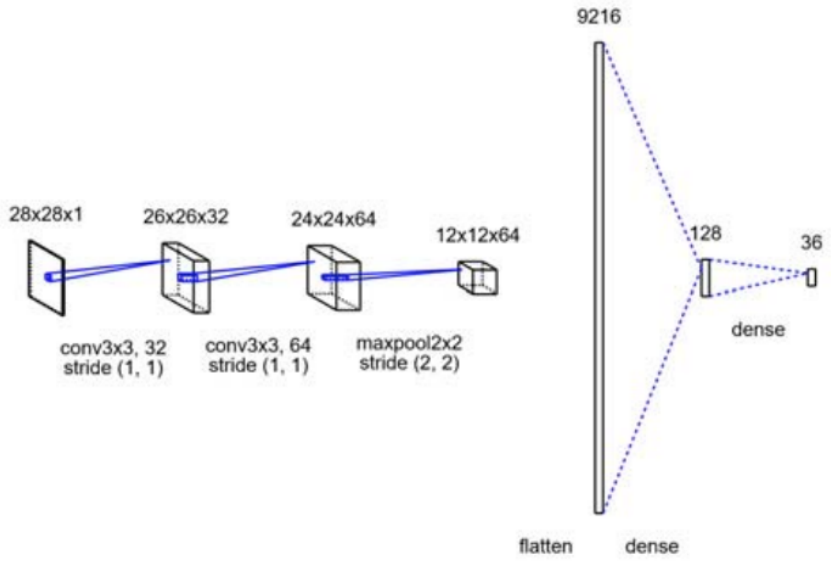

Figura 2.4: Rede neuronal convolucional 2D (Taskiran et al., 2018)

Convex Hull. Por fim, a imagem resultante é redimensionada para se adaptar ao tamanho que a CNN aceita na entrada. Os resultados referidos reportam em média 98.05% de acurácia, nos cinco testes realizados (Taskiran et al., 2018).

O lançamento do *Kinect*, por parte da *Microsoft*, em Novembro de 2010, foi direcionado para a consola *Microsoft XBox 360*. Era uma alternativa ao comando tradicional, disponibilizando uma experiência inovadora recorrendo a visão computacional nos jogos. O *Kinect* possui uma câmara RGB com resolução de 640 por 480 pixeis e um sensor de profundidade de 320 por 240 pixeis. Consegue rastrear o movimento do corpo do utilizador. Estas características levaram a comunidade científica a adotar o *Kinect* em inúmeros projetos de visão por computador.

Um exemplo da sua utilização no contexto de SLR foi proposto por Jie Huang. Este projeto propõe a utilização de uma CNN 3D, num conjunto de dados criado pelo mesmo, onde utilizaram o *Kinect* para a sua recolha. Este conjunto de dados identifica 25 classes distintas. Para cada classe recolheram 27 vídeos, totalizando 675 vídeos em todas as classes (Jie Huang et al., 2015).

A CNN proposta consiste em 8 camadas e está presente na Figura 2.5. Após a camada de entrada, seguem-se a camada convolucional (C1), *sub-samplings* (S1), camada convolucional (C2), *sub-samplings* (S2) e a camada convolucional (C3). Por fim, estão presentes duas camadas totalmente conectadas. Com esta proposta, conseguiram atingir 94,2% de acurácia, aquando do uso da CNN 3D com *Multi-channels*. Os mesmos autores utilizaram ainda *Gaussian Mixture Model-Hidden Markob Model* (GMM-HMM), como referência, tendo atingido uma acurácia média de 90,8%. Concluíram que com os testes

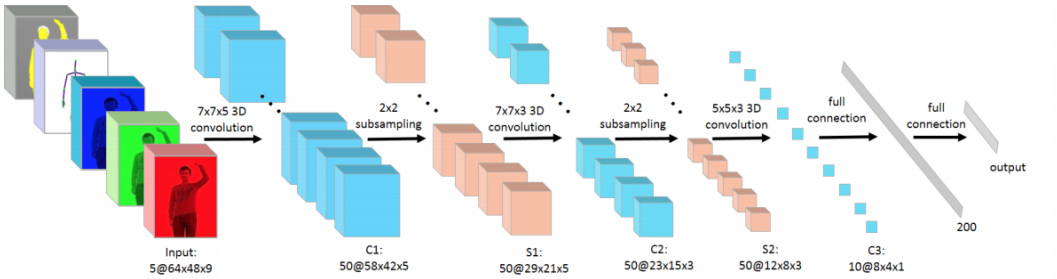

Figura 2.5: Rede neuronal convolucional 3D (Jie Huang et al., 2015)

realizados permitiram assim demonstrar a eficácia do modelo proposto (Jie Huang et al., 2015).

Pigou reporta a utilização de dados provenientes do *Kinect* para treinar o modelo proposto. O conjunto de dados escolhido pelo autor foi o *ChaLearn Looking at People 2014* (CLAP14) que se refere à Língua Gestual Italiana. Este conjunto de dados foi recolhido por 27 pessoas, com diferentes roupas, em diferentes locais, com diferentes níveis de luminosidade e diferentes tipos de movimentos. Dos 6600 vídeos presentes no conjunto de dados, 4600 foram utilizados para treino e 2000 para o conjunto de dados para validação. No pré-processamento recortou a imagem para obter apenas a mão e a parte superior do corpo. Cada uma destas imagens contendo a sua referência de profundidade e a imagem em escala de cinzento (Pigou et al., 2015).

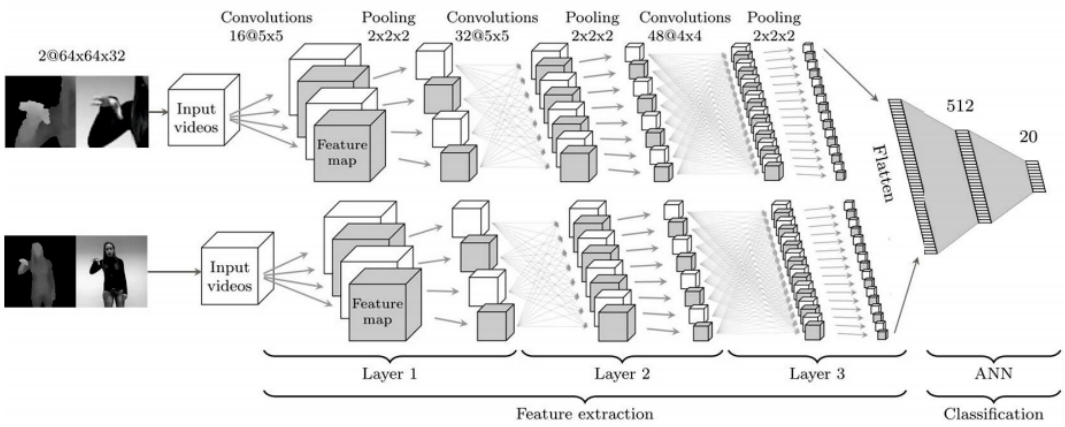

Figura 2.6: Rede neuronal convolucional (Pigou et al., 2015)

O autor refere que a utilização de redes neuronais convolucionais 2D resultaram numa melhor acurácia na fase de validação do que utilizando redes neuronais convolucionais 3D. A arquitetura proposta consiste em duas CNN, uma utilizada para extrair *features* da mão e a outra para extrair *features* da parte superior do corpo. Cada CNN está dividida em 3 camadas, como se pode visualizar na Figura 2.6, a camada 1, camada 2 e camada

3, que é onde ocorre a extração das *features*. Após estas três camadas está presente uma *Artificial Neural Network* (ANN) onde é concatenado o resultante das duas CNN. O treino é realizado utilizando o GPU, onde é utilizado *Nesterov's accelerated gradient descent* com um *momentum* fixo de 0.9 e com uma taxa de aprendizagem inicial de 0.003, sendo a mesma reduzida após cada época em 5%. Com esta proposta, o autor foi capaz de atingir cerca de 91.70% de acurácia no conjunto de validação e 95.68% no conjunto de dados de teste.

Bantupalli e Xie apresentam uma abordagem para o reconhecimento da ASL, utilizando visão computacional. Em relação ao conjunto de dados, os autores optaram pelo desenvolvimento de um com 100 gestos distintos, referentes à ASL. Este conjunto de dados é constituído por vídeos, cuja resolução é de 720p. Posteriormente cada vídeo foi dividido em imagens (Bantupalli and Xie, 2018).

Nesta proposta, utilizaram o modelo *Inception* para extrair *features* espaciais e de seguida uma rede neuronal recorrente (RNN), para extração de *features* temporais. Após o pré-processamento das imagens, estas são enviadas para a CNN para que sejam extraídas *features*, que são enviadas para uma *Long-Short Term Memory* (LSTM) e, por fim classificadas. Referem ainda que, tanto a CNN como a RNN, foram treinadas independentemente, e ambas utilizaram tamanho de *batch* de 64 e 10 épocas. Vários problemas foram observados pelos autores. A utilização de cores de pele não presentes no conjunto de dados fez com que a acurácia do modelo durante os testes diminuísse. E também, devido ao conjunto de dados possuir diversos vídeos que continham faces de pessoas diferentes, o modelo treinou *features* incorretas. Foi reportada uma acurácia de 93% aquando da utilização de 100 gestos (Bantupalli and Xie, 2018).

2.3 Redes neuronais

Um dos componentes de uma rede neuronal é o neurónio, sendo ele uma função que aceita um valor de entrada e produz um valor de saída. Uma rede neuronal (RN) é uma rede de neurónios que estão organizados em diferentes camadas.

Para além de neurónios, uma RN é constituída por conexões entre as várias camadas, em que cada conexão fornece a saída de um neurónio como uma entrada no próximo neurónio. Cada conexão tem associada a si um peso. Estes pesos, ao longo do treino da rede neuronal, vão sendo atualizados com o auxílio de algoritmos optimizadores, para que a acurácia da rede seja melhorada.

Dependendo da complexidade da rede neuronal, um neurónio pode possuir um ou mais neurónios, tanto de entrada como de saída.

Também faz parte de uma RN funções de ativação, que são algoritmos matemáticos que são ativados caso os requerimentos sejam atingidos, ou seja, se o valor resultante do algoritmo for superior ao valor limite, então o valor é calculado e passado ao próximo neu-

rónio. Exemplos destas funções são a função *Sigmoid*, a *Rectified Linear Units* (ReLU) e também o *Softmax*.

Outra noção muito importante em RN é o conceito de *Back Propagation*. Tendo o conceito sido introduzido por Rumelhart, *Back Propagation*, é o ato de, repetidamente, afinar os pesos das conexões da RN, tendo como base a perda gerada pela função de perda, obtida da época anterior, com o objetivo de minimizar a diferença do valor atual de saída e o valor esperado de saída (Rumelhart et al., 1988).

Redes neuronais funcionam como um mapeador onde uma variável entrada é mapeada para uma certa variável de saída. Existem variados tipos de redes neuronais, tais como as redes neuronais convolucionais, redes neuronais recorrentes, entre outras.

2.3.1 Redes neuronais convolucionais

Uma CNN é um tipo de modelo de *deep learning* com a finalidade de processar dados que possuem algum tipo de padrão, como por exemplo imagens (Yamashita et al., 2018).

Numa CNN, a primeira camada é referente a uma camada de convolução, que é a camada onde são extraídas as diferentes caraterísticas presentes na imagem colocada de entrada. Este tipo de camada é constituída por duas matrizes, uma referente à matriz da imagem e outra referente ao filtro. Estas duas matrizes são multiplicadas entre si, resultando um mapa de caraterísticas.

As matrizes de filtros ou também denominadas de *kernel* podem ter variados objetivos, como por exemplo, a deteção de arestas, sendo esta útil para marcar limites e divisões numa imagem.

Associado a CNN, estão diferentes tipos de convoluções, tais como, convoluções 2D e 3D. De acordo com Jie Huang, a convolução 2D, é realizada num mapa de caraterísticas 2D para extração de caraterísticas da camada anterior. Isto ocorre com o movimento do *kernel* sobre cada unidade da camada anterior (Jie Huang et al., 2015). Na Figura 2.7 está presente um exemplo de uma convolução 2D.

Em relação a convoluções 3D, para além da dimensão espacial, que também está presente em convoluções 2D, são constituídas por mais uma dimensão. Usualmente, este tipo de convoluções são utilizadas quando estão em causa dados de imagens 3D, e também vídeos. O autor, também refere um exemplo de convoluções 3D utilizado em vídeo. Neste caso, para além da dimensão espacial é utilizada a dimensão temporal (Jie Huang et al., 2015). Uma convolução 3D, assim como uma convolução 2D, baseia-se no movimento do *kernel* sobre cada unidade da camada anterior, mas neste caso em específico, sobrepondo múltiplas estruturas próximas, como se pode visualizar na Figura 2.8.

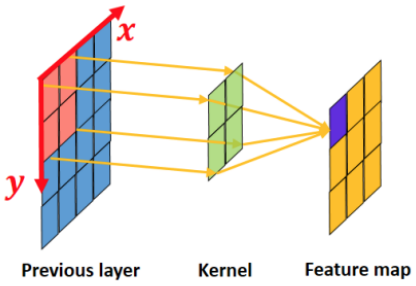

Figura 2.7: Convolução 2D (Jie Huang et al., 2015)

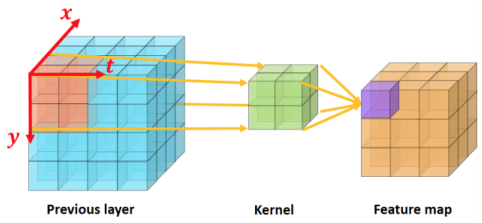

Figura 2.8: Convolução 3D (Jie Huang et al., 2015)

2.3.2 Optimizadores

Os optimizadores são algoritmos que são utilizados para fazer a atualização de parâmetros das redes neuronais, tais como os seus pesos e a taxa de aprendizagem, de forma a reduzir a perda.

Kingma e Ba propõem um método eficiente para optimização, que apenas necessita de gradientes de primeira ordem com poucos requisitos em termos de memória (Kingma and Ba, 2014). Este método foi desenvolvido com o objetivo de combinar as diversas vantagens de dois métodos populares, o *AdaGrad* e o *RMSProp*.

No mesmo artigo, são referidos contextos em que os métodos *AdaGrad* e o *RMSProp* têm um bom desempenho. Referindo que no caso do primeiro, este possui um bom desempenho aquando da sua utilização com *sparse gradients*. No caso do segundo método, refere que este possui um bom desempenho quando utilizando *online* e definições não estacionárias.

São ainda referidas algumas experimentações, estas utilizando, regressão logística, redes neuronais multi-camada e também CNNs. Na experimentação relacionada com CNNs, a arquitetura da mesma possui três estágios alternados de filtros convolucionais, assim como está presente a utilização de *max pooling* de 3x3 com *stride* de valor 2. Seguidamente possui uma camada conectada de 1000 ReLU. A imagem de entrada utilizada nesta experimentação sofre um pré-processamento, inicialmente um branqueamento da mesma, seguindo-se de uma aplicação de um filtro de *dropout noise* na camada de entrada da CNN. Os autores referem ainda que utilizaram *minibatches* de tamanho 128. Os resultados referentes a esta experimentação podem ser visualizados na Figura 2.9.

Por fim, os autores concluem que o método introduzido neste artigo é eficiente, sendo direcionado para problemas relacionados com *Machine Learning* (ML), que possuam grandes conjuntos de dados (Kingma and Ba, 2014).

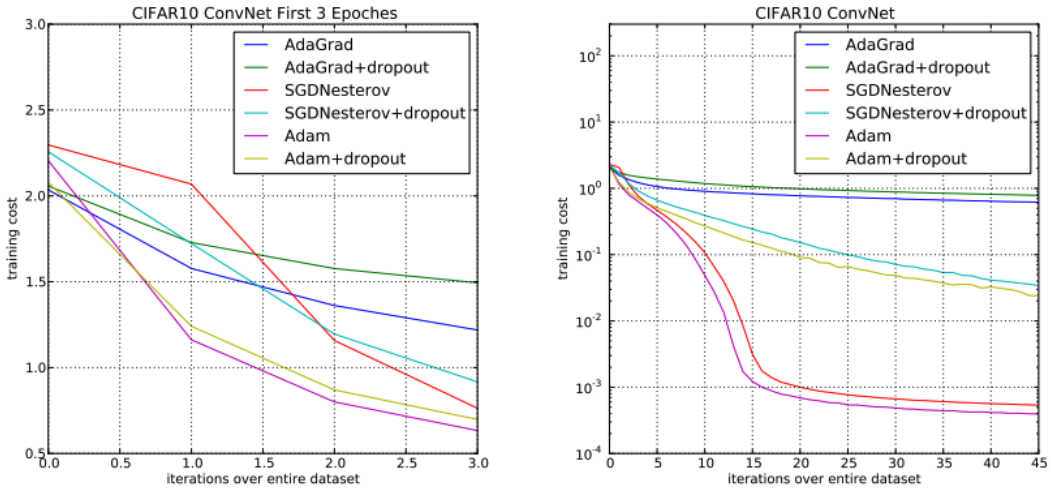

Figura 2.9: À esquerda o custo de treino da CNN nas primeiras 3 épocas e à direita o custo durante 45 épocas (Kingma and Ba, 2014).

2.3.3 Funções de perda

A função de perda é utilizada para se calcular o erro entre o valor que o modelo prevê e o valor que realmente lhe está associado. Resumidamente, da função de perda resulta a avaliação de quão bem o algoritmo se molda ao conjunto de dados. Caso o valor resultante seja elevado significa que as previsões provenientes do modelo estão a ser erradas, caso o valor seja baixo é o contrário.

2.4 Bibliotecas

Existem bibliotecas capazes de fornecer coleções de fluxos de trabalho para desenvolver e treinar modelos, para serem executados na nuvem, nos navegadores de *internet* e até em dispositivos móveis. Iremos agora rever algumas.

2.4.1 *OpenCV*

OpenCV é uma biblioteca de código aberto que tem como objetivo facilitar o desenvolvimento de aplicações na área de Visão Computacional. Esta biblioteca conta ainda com variados módulos, desde processamento de imagem e vídeo, passando por estrutura de dados e até construção de interface gráficas (OpenCV, s.d.). Como é uma biblioteca amplamente utilizada possui variadíssimas linguagens de programação que a suportam, tais como *Python*, *Java*, *MATLAB*, entre outras.

2.4.2 *Tensorflow*

Tensorflow (TF) é uma biblioteca de código aberto para ML, funcionando em variados sistemas operativos, tais como, *Windows*, *Linux*, *macOS*, entre outros (TensorFlow, s.d.). É utilizada para pesquisa e desenvolvimento de modelos, com suporte para treino em CPU como em GPU, este último utilizando os núcleos CUDA das GPU das *NVIDIA*. Está presente no mercado há 5 anos, encontra-se numa versão estável, e é utilizada tanto para pesquisa como também para serviços de produção na *Google*. O TF para além da sua versão core, ainda possui a versão Tensorflow.js (TF.js) que serve para desenvolver modelos em *Javascript* (JS) e utilizar ML diretamente em navegadores de *internet* ou em *Node.js*. A versão tflite é outra versão de TF que é direcionada para a execução de modelos que, à priori, foram treinados utilizando TF, e convertidos para serem compatíveis com tflite, em dispositivos relacionados com a Internet das Coisas (IoT), e dispositivos móveis, tais como, *smartphones*. Em todas estas versões, excecionando o TF.js, a linguagem de programação de eleição é *Python*, apesar de que esta biblioteca tem suporte oficial para outras linguagens de programação tais como JS, *Java*, *Go*, entre outras.

2.4.3 *PyTorch*

PyTorch é uma biblioteca de código aberto para ML, que funciona em variados sistemas operativos, tais como, *Windows*, *Linux* e *macOS*. É utilizada para pesquisa e desenvolvimento de modelos, com suporte para treino utilizando tanto o CPU como o GPU, este último utilizando os núcleos CUDA das GPU das *NVIDIA* (PyTorch, s.d.). Esta biblioteca é utilizada em sistemas desenvolvidos pela *Tesla*, como por exemplo o conhecido *Tesla Autopilot* (Karpathy, 2019).

2.4.4 *Apple Core ML*

Core ML é uma biblioteca desenvolvida e mantida pela *Apple* direcionada para dispositivos *Apple*, como o *iPhone*, *iPad*, *macOS*, entre outros. Está otimizada para desempenho *on-device* nos mais variados dispositivos *Apple*, minimizando tanto a memória utilizada como o consumo de energia. Suporta a conversão de modelos desenvolvidos por outras bibliotecas, como por exemplo, TF e *PyTorch*, tendo ainda suporte para utilização via *CPU*, *GPU* e *Neural Engine*. Este último trata-se de um *hardware* dedicado para redes neuronais (Apple, s.d.).

Durante o presente capítulo, foi efetuada a revisão da literatura. Foram abordados conceitos como linguagem e língua, clarificando qual a sua diferença, bem como identificando o que distingue a língua da língua gestual.

Várias abordagens da aplicação de SLR foram apresentadas com o objetivo de resolver problemas, desde a utilização de dispositivos como as *Data Gloves*, até técnicas de visão computacional utilizando CNN.

Foram ainda apresentadas diferentes bibliotecas com as quais é possível manipular imagem, e também bibliotecas utilizadas por variadíssimas empresas, no seu dia-a-dia, no contexto de visão computacional, com a particularidade de todas as referidas, à exceção do *OpenCV*, possuírem suporte a processamento no próprio dispositivo móvel.

Capítulo 3

Sistema SignPic

Neste capítulo é retratada a especificação do Sistema SignPic, iniciando com uma breve contextualização acerca deste sistema.

De seguida, são apresentados os variados requisitos do sistema, os requisitos funcionais e os não funcionais.

A arquitetura do Sistema SignPic é revelada, dando a conhecer que esta se divide em duas partes. A primeira envolve todo o sistema desenvolvido para treinar e utilizar o modelo gerado para classificação de imagem, e a segunda parte inclui a aplicação móvel para sistema *iOS*, capaz de utilizar o modelo gerado para classificação da imagem no próprio dispositivo, sem qualquer ligação externa.

Para além do anteriormente mencionado é ainda apresentada a implementação do Sistema SignPic. Esta é operada em três fases. A primeira fase, refere-se à escolha, tanto do conjunto de dados como da biblioteca a utilizar para o desenvolvimento. Existem bastantes bibliotecas capazes de desenvolver o trabalho pretendido, como por exemplo, o TF, *PyTorch*, entre outros.

Durante a segunda fase de implementação, é apresentada uma proposta de implementação de um modelo capaz de detetar vários gestos, neste caso, referente a gestos presentes no alfabeto da ASL.

Na última fase, é utilizado o modelo proposto na fase dois. Esse modelo é agora executado num dispositivo móvel onde é efetuado todo o processamento, desde a captura e o pré-processamento da imagem, até à sua classificação ser efetuada no próprio dispositivo, tirando assim a necessidade de utilização da *internet*.

Iremos agora apresentar o funcionamento da nossa aplicação *SignPic*, sendo que todo o código fonte da aplicação encontra-se disponível no *GitHub* (Nogueira, 2020).

3.1 Requisitos

3.1.1 Requisitos funcionais

ID: RF1.1
Título: Deteção e interpretação de gestos efetuados pelas mãos
Descrição: A aplicação móvel deverá conseguir traduzir o gesto efetuado pela mão do utilizador.

3.1.2 Requisitos não funcionais

ID: RNF1.1
Título: Segurança
Descrição: A aplicação móvel não deverá guardar qualquer informação acerca do utilizador.

ID: RNF1.2
Título: Desempenho
Descrição: A aplicação móvel deverá ser capaz de prever o gesto realizado pelo utilizador num curto espaço de tempo.

ID: RNF1.3
Título: Usabilidade
Descrição: A aplicação móvel deverá possuir uma interface de fácil uso.

ID: RNF1.4
Título: Baixo custo
Descrição: A aplicação móvel deverá ser compatível com telemóveis com pouco poder de processamento.

3.1.3 Requisitos de Sistema (Software e Hardware)

ID: RS1.1
Título: Dispositivo com bom poder computacional
Descrição: Foi utilizado um CPU com bom poder computacional para treinar o modelo de classificação.

ID: RS1.2

Título: Dispositivo móvel para executar o modelo de classificação
Descrição: Foi utilizado um Apple iPhone 6S com a versão 13 do iOS para executar o modelo de classificação.

ID: RS1.3
Título: Linguagem de programação para desenvolvimento e treino do modelo
Descrição: Foi utilizado Python conjuntamente com a biblioteca PyTorch para o desenvolvimento e treino do modelo de classificação.

ID: RS1.4
Título: Linguagem de programação para desenvolvimento da aplicação móvel
Descrição: Foi utilizado Swift para o desenvolvimento da aplicação móvel para dispositivos iOS.

3.2 Arquitetura do sistema

O desenvolvimento deste tipo de aplicação envolve diferentes fases, sendo necessário recorrer a várias ferramentas e bibliotecas para auxiliar e para desenvolver código fonte, como é exemplo a biblioteca da linguagem de programação *Python*, *PyTorch*. Esta foi a biblioteca que serviu de base ao desenvolvimento do modelo classificador, que é a primeira das duas partes do sistema desenvolvido.

3.2.1 Classificador de imagem

A biblioteca *PyTorch*, abarca um *Zoo* de modelos *torchvision*, como o *Resnet18*, *Alexnet*, *VGG11_bn*, *Squeezenet*, *Densenet* e *MobileNet v2*, entre outros.

Para o desenvolvimento desta aplicação foram utilizados os modelos enunciados acima para comparação posterior, no Capítulo 4. Todos os modelos utilizados são modelos já pré-treinados, ou seja, os pesos do modelo são transferidos para um diretório de cache do sistema. Sendo os modelos pré treinados, existe a opção de executar dois tipos de *transfer learning*. Pode-se optar por efetuar um *finetuning*, que consiste na atualização de todos os parâmetros do modelo pré-treinado, de modo a ficarem de acordo com o conjunto de dados utilizado, ou efetuar *feature extraction*, que, suportando-se na utilização do modelo pré-treinado, apenas atualiza os pesos relativos à última camada (Inkawhich, s.d.).

No trabalho a desenvolver pretende-se efetuar *finetuning*, porque os dados utilizados para pré-treinar o modelo não são os presentes no conjunto de dados escolhido para o desenvolvimento do sistema. Para além disso, os testes efetuados no capítulo 4, corroboram a escolha de se efetuar *finetuning* em vez de *feature extraction*.

Relativamente à arquitetura da CNN, optou-se pela utilização de uma desenvolvida pela *Google*, a *MobileNet v2*. Trata-se de um modelo direcionado para processamento no próprio dispositivo. Obtiveram-se resultados semelhantes aos demais modelos nos testes realizados e como o modelo gerado possui um tamanho de cerca de 10Mb, que comparado com as outras opções é um valor relativamente baixo, concluiu-se que é passível de ser utilizado num dispositivo móvel.

Para além do enunciado anteriormente, o facto de este modelo ser um dos presentes no *Zoo* da biblioteca *PyTorch*, veio facilitar a sua utilização. A arquitetura da *MobileNet v2* pode ser visualizada na Figura 3.1.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figura 3.1: Arquitetura do modelo utilizado (Sandler et al., 2018)

3.2.2 Aplicação móvel

A arquitetura da aplicação de *iOS*, não assenta em nenhuma arquitetura vulgarmente utilizada, esta é bastante simples, não necessitando de qualquer tipo de persistência de dados. É apenas constituída por uma *ViewController*, onde são declaradas e inicializadas as variáveis necessárias, como por exemplo, o módulo que é utilizado para prever o gesto presente nas imagens e também uma única *Storyboard* onde é implementada a parte referente a interface do utilizador (UI).

O diagrama referente ao fluxo da aplicação móvel *iOS*, pode ser visualizado na Figura 3.2.

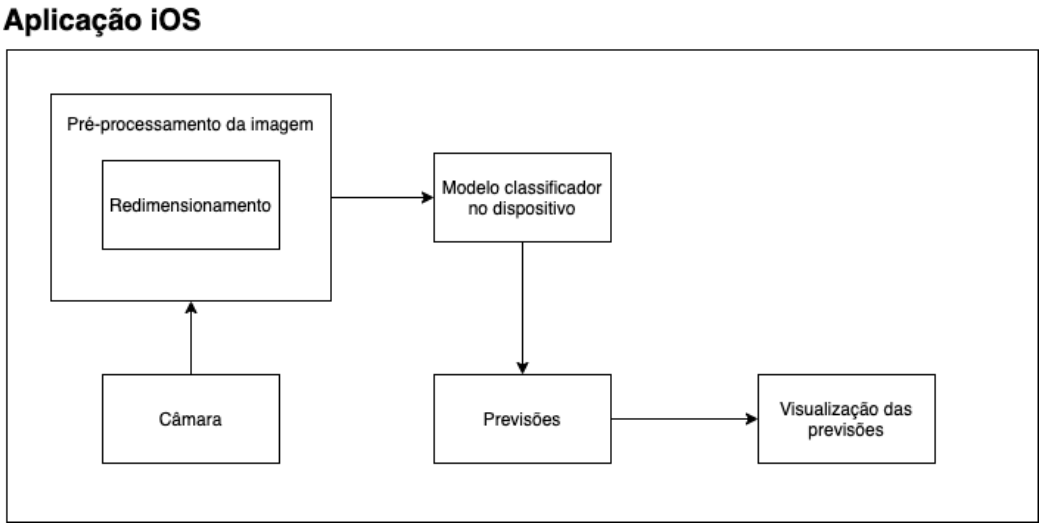

Figura 3.2: Diagrama de fluxo da aplicação móvel *iOS*

3.3 Conjunto de dados utilizados

Para o desenvolvimento deste projeto, era necessário proceder à escolha de um conjunto de dados, referente à LGP. Efetuada a pesquisa, os dados recolhidos mostraram-se insuficientes para o trabalho a desenvolver, pelo que foi necessário a criação de um conjunto de dados.

Num primeiro momento optou-se por criar um pequeno conjunto de dados em formato de vídeo, referente a três letras da LGP, nomeadamente a letra A, a letra B e a letra C. Este conjunto de dados possuía cerca de 250 vídeos por cada entrada. Os vídeos possuíam pouca diversidade referentes à mesma entrada, relativamente a locais, luminosidade, pessoas, era muito custoso em termos de tempo e não foi exequível o desenvolvimento de um conjunto de dados aceitável. A escolha recaiu num conjunto de dados presentes na plataforma *Kaggle*, mas este referente à ASL. Portanto, o conjunto de dados escolhido para o desenvolvimento foi o conjunto de dados denominado *alphabets sign language* presente na Figura 3.3, colocado na plataforma *Kaggle* por Saze.

Este conjunto de dados para além de ter todas as letras da ASL, possui ainda três entradas, o espaço, o apagar e a entrada referente ao vazio. Contém, em média, cerca de 2500 imagens por cada entrada.

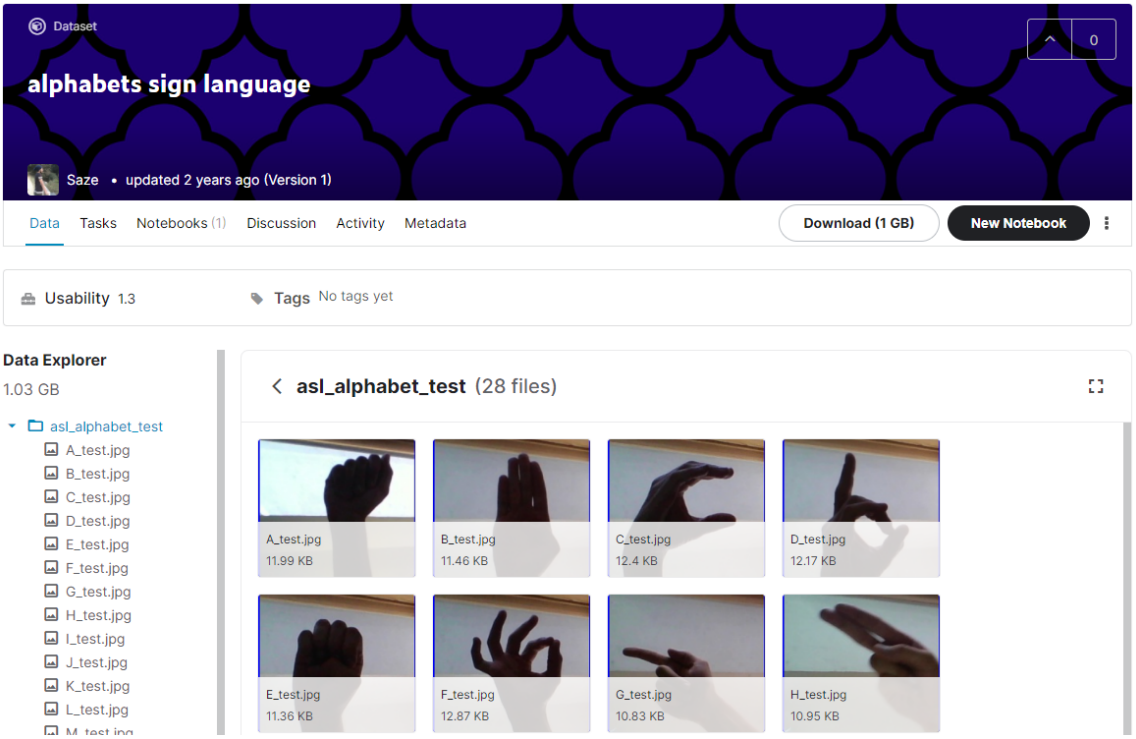

Figura 3.3: Conjunto de dados escolhido para o desenvolvimento (Saze, 2018)

3.4 Tratamento dos dados

O conjunto de dados é todo homogéneo, todas as imagens referentes às diversas entradas possuem a dimensão de 200 pixeis por 200 pixeis. Sendo que a camada de entrada dos modelos necessita de imagens com dimensões de 224 pixeis por 224 pixeis, utilizou-se métodos disponibilizados por uma das biblioteca do *PyTorch*, nomeadamente a *torchvision*, com a qual é possível utilizar *transforms*, em cadeia, para efetuar diferentes tipos de transformações comuns em imagens, neste caso, a aplicação de um redimensionamento da imagem, uma rotação e a transformação da imagem para um *tensor* ou *numpy.array*.

3.5 Implementação do modelo classificador

Durante a fase de implementação, são definidas as transformações efetuadas no conjunto de dados que é utilizado para efetuar o treino, como se verifica na Figura 3.4. O modelo escolhido necessita de dimensões de entrada de 224 pixeis por 224 pixeis, pelo que foi utilizado um *transform* para efetuar o redimensionamento. De seguida, na imagem foi efetuada uma rotação aleatória de 30º e esta também foi rodada horizontalmente. Esta rotação horizontal apenas acontece em 50% das vezes, que é o valor por omissão desse *transform*.

O *transform.ToTensor()* é utilizado para transformar uma *PIL Image* num *torch.FloatTensor*.

Por fim, é utilizado o *transform.Normalize(...)* para normalizar a *tensor image* com a média, sendo esta o primeiro argumento da função, e o desvio padrão é o segundo argumento que a função recebe.

```
train_transforms = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.RandomRotation(30),
    transforms.RandomHorizontalFlip(p = 0.5),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

Figura 3.4: Transformações efetuadas no conjunto de dados de treino

Na Figura 3.5 estão presentes as transformações efetuadas às imagens referentes ao conjunto de dados de validação. Assim como no conjunto de dados de treino, aqui também ocorre o redimensionamento das imagens para dimensões de 224 pixeis por 224 pixeis. Logo após, é aplicada a transformação da *PIL Image* num *torch.FloatTensor*, e por fim acontece a normalização, tal e qual, como sucede no conjunto de dados de treino.

```
test_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

Figura 3.5: Transformações efetuadas no conjunto de dados de validação

Após a definição das transformações, tanto para o conjunto de dados de treino como do conjunto de dados de validação, é feita a importação dos respetivos conjuntos de dados. Para tal, é utilizada a função de leitura de pastas que contêm imagens presentes na

biblioteca *torchvision*. Para efetuar a leitura é necessário passar como argumentos o caminho até ao diretório de pastas onde cada um dos conjuntos de dados está presente, como se visualiza nas Figuras 3.6 e 3.7, e as respetivas cadeias de transformações referidas anteriormente, que são passíveis de visualização nas Figuras 3.4 e 3.5.

```
train_data = datasets.ImageFolder(train_path, transform = train_transforms)
train_loader = torch.utils.data.DataLoader(train_data, batch_size = batch_size, shuffle = True)
```

Figura 3.6: Leitura dos dados presentes na pasta referente ao treino

```
test_data = datasets.ImageFolder(test_path, transform = test_transforms)
test_loader = torch.utils.data.DataLoader(test_data, batch_size = batch_size)
```

Figura 3.7: Leitura dos dados presentes na pasta referente à validação

No caso de utilização do modelo pré-treinado *MobileNet v2*, visualizável na Figura 3.8, para que seja possível utilizar o modelo escolhido é necessário fazer alterações ao nível da última camada. Esta alteração ocorre pelo facto de o mesmo ser composto por uma camada de *Dropout*, que irá colocar a zero alguns elementos de entrada do *tensor*, por omissão a probabilidade de isso acontecer presente neste modelo é de p = 0.2. Para além da camada de *Dropout*, existe também uma camada linear, que consiste numa alteração linear, $y = xA^t + b$, que espera 1280 variáveis de entrada e 1000 variáveis de saída, o que neste caso, não é o pretendido, devido a estarmos à espera de 29 variáveis de saída, referentes ao alfabeto presente na ASL e alguns caracteres especiais. Todas as alterações referidas são visualizáveis na Figura 3.9.

Em relação a optimizadores, ou funções de optimização, as opções são variadas, sendo o seu objetivo a atualização de parâmetros da rede neuronal, tais como os pesos e a taxa de aprendizagem. O optimizador *Adam* e o optimizador SGD, este último uma variante do *Gradient Descent* com a particularidade de tentar atualizar os parâmetros mais frequentemente, são os utilizados para os diferentes testes que estão disponíveis para visualização no Capítulo 4. Pode verificar-se a implementação na Figura 3.10 onde foi utilizado o optimizador Adam, com uma taxa de aprendizagem inicial, de valor 0.001. A implementação de uma variação dessa taxa a cada época foi efetuada. Na Figura 3.11 é possível visuali-

```
model = models.mobilenet_v2(pretrained = True)
```

Figura 3.8: Definição do modelo utilizado para o desenvolvimento

```
model.classifier = nn.Sequential(
    nn.Dropout(p = 0.2),
    nn.Linear(in_features = 1280, out_features = 29)
)
```

Figura 3.9: Definição da última camada do modelo utilizado

zar a sua implementação, sendo o seu segundo argumento o período de queda da taxa de aprendizagem, que neste caso é de "1", porque se pretende que isto ocorra a cada época. O terceiro argumento de valor 0.8, refere-se ao valor pelo qual se pretende multiplicar a taxa de aprendizagem.

```
optimizer = optim.Adam(model.parameters(), lr = 0.001)
```

Figura 3.10: Definição do optimizador

A função de perda escolhida para ser utilizada é a *CrossEntropyLoss*, como se pode verificar na Figura 3.12. Esta efetua o cálculo $log(Softmax(x))$ que faz com que os valores fiquem compreendidos entre $[-\infty, 0)$, e também para que os valores resultantes sejam *log-probabilities*, para que seguidamente se efetue a *Negative Log Likelihood Loss*.

Por cada iteração de época, a função onde ocorre o treino do modelo é invocada. A implementação dessa função está presente na Figura 3.13, denominada de *train*. Inicial-

```
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, 1, 0.8)
```

Figura 3.11: *Scheduler*

```
loss_fn = nn.CrossEntropyLoss()
```

Figura 3.12: Definição da função de perda

mente colocamos o modelo em modo de treino (*model.train*). Após, é necessário iterar pelo conjunto de dados de treino para obter a imagem e a respetiva classe. A necessidade da invocação da função *zero_grad* prende-se com o facto de a biblioteca *PyTorch* acumular os gradientes durante as passagens pela função *backward*, invocada mais abaixo. Assim, com a utilização da função *zero_grad*, os gradientes são colocados a zero. Segue-se a passagem da imagem para o modelo, que retorna o resultado previsto.

Assim como a função de treino, a função de teste que está presente na Figura 3.14, denominada de *test*, é invocada a cada iteração de época, após a invocação da função de treino. Esta função, inicialmente coloca o modelo em modo de validação (*model.eval*). Após isso, a utilização da função *no_grad*, faz com que não sejam usados gradientes, desde que estejam dentro da indentação da mesma. De seguida, ocorrem iterações no conjunto de dados de validação, em que em cada iteração a imagem é passada para dentro do modelo. Por cada iteração também é calculada a perda e adicionada ao valor anterior, que está presente na variável *test_loss*, que inicialmente é igualada a 0. Para obter a classe, que é mais provável que seja referente à imagem inserida no modelo, utiliza-se $\_, predicted = torch.max(output.data, 1)$, que de todos os valores previstos, retorna o valor mais elevado.

Na Figura 3.15, está demonstrado a iteração pelas diversas épocas, que inicialmente invoca a função de treino e posteriormente a função de teste. Para além destas duas invocações, é neste bloco de código onde ocorrem as escritas de diversas estatísticas para

```python
def train():
    model.train()
    train_loss, correct = 0, 0
    for index, value in enumerate(train_loader):
        inputs, labels = value[0], value[1]
        optimizer.zero_grad()
        output = model(inputs)
        loss = loss_fn(output, labels)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
        _, predicted = torch.max(output.data, 1)
        correct += (predicted == labels).sum().item()
```

Figura 3.13: Função referente ao treino

```python
def test():
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for index, value in enumerate(test_loader):
            inputs, labels = value[0], value[1]
            output = model(inputs)
            test_loss += loss_fn(output, labels).item()
            _, predicted = torch.max(output.data, 1)
            correct += (predicted == labels).sum().item()
```

Figura 3.14: Função referente ao teste

posterior visualização e avaliação e é também onde o modelo é guardado.

```
for epoch in range(1, epochs + 1):
    train_loss, train_accuracy = train()
    writer.add_scalar('Loss/train', train_loss, epoch)
    writer.add_scalar('Accuracy/train', train_accuracy, epoch)

    test_accuracy, test_loss = test()
    writer.add_scalar('Loss/test', test_loss, epoch)
    writer.add_scalar('Accuracy/test', test_accuracy, epoch)
```

Figura 3.15: Ciclo de iterações referente às épocas

O modelo é guardado de diferentes formas: i) utilizando a invocação da função para guardar, disponibilizada pela biblioteca; ii) utilizando a mesma função, mas desta vez, com diversos parâmetros, como a época atual, um dicionário que mapeia cada camada para o seu respetivo *tensor*, assim como um dicionário contendo o estado do optimizador e também a perda, para que seja possível, caso se queira, continuar a treinar o modelo a partir de um *checkpoint*; e iii) efetuando a optimização do modelo para dispositivos móveis e posteriormente guardado utilizando a API presente nas utilidades do *PyTorch*, a *mobile_optimizer*. Esta API, neste momento, ainda está em fase *Beta*, mas durante a utilização da mesma não verificamos nenhum problema.

Apesar de a biblioteca *PyTorch* suportar, tanto a utilização do CPU como do GPU para efetuar o treino, aquando da utilização do modelo gerado numa aplicação móvel *iOS*, é necessário que seja treinado em CPU, devido à incompatibilidade em relação ao suporte de inferência em GPU.

Como realizar treino em CPU é muito mais demorado comparando com treino em GPU, para obtenção de resultados dos demais testes foi utilizado GPU, sendo usado apenas CPU para o treino, com finalidade de geração do modelo compatível com a aplicação móvel para *iOS*.

Para a visualização das diversas estatísticas guardadas durante os treinos e os testes, é utilizado um kit de ferramentas de visualização da biblioteca TF. Apesar de o *Tensorboard* fazer parte de uma biblioteca diferente da que utilizamos para a implementação, o *PyTorch* possui compatibilidade com a mesma. Sendo apenas necessário a instalação do

Tensorboard utilizando o gerenciador de pacotes de padrão do *Python* (pip), *pip install tensorboard*, e a posterior execução do comando $tensorboard --logdir = runs$ no diretório onde as estatísticas são guardadas. Na Figura 3.16, podemos visualizar o painel de controlo do *Tensorboard* com dados estatísticos referentes a variados treinos e testes efetuados durante a implementação, para a posterior comparação entre as diferentes *runs*.

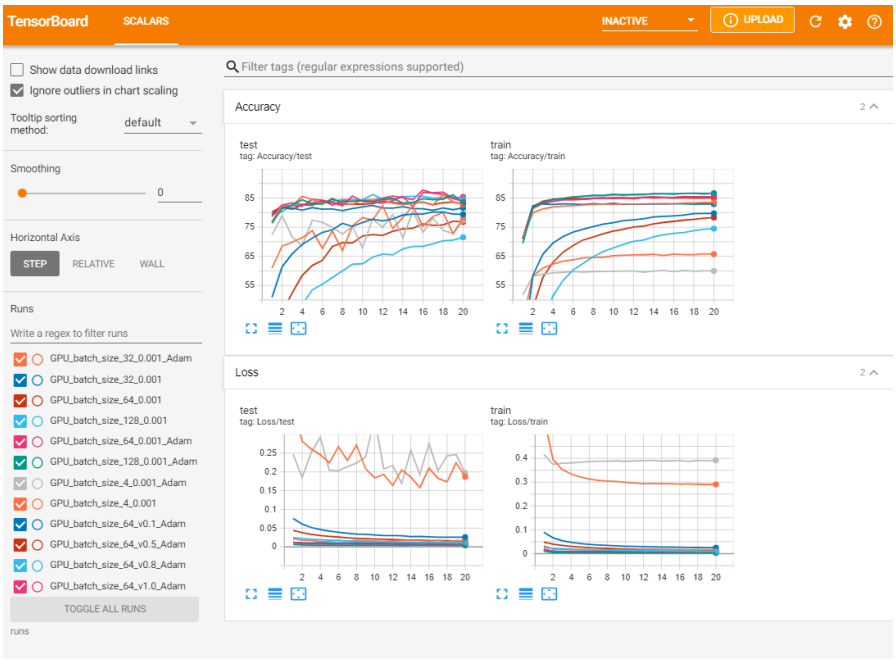

Figura 3.16: Painel de controlo do Tensorboard

3.6 Implementação da aplicação móvel

Nesta fase, após o treino e constituição do modelo classificador, é necessário o desenvolvimento da aplicação para *iPhones* (*iOS*).

Para o desenvolvimento da aplicação, é utilizada como base, a aplicação demo do *PyTorch*, presente num repositório *Git* no perfil da biblioteca na plataforma *Github*, como se pode ver na Figura 3.17.

Em relação ao desenvolvimento da aplicação para a plataforma *iOS*, inicialmente é necessário a criação de um ficheiro denominado *podfile*, que é o ficheiro onde estão referidas as dependências da aplicação, sendo necessário para tal, a execução do comando *pod init* no diretório base do projeto *XCode*.

Com o ficheiro *podfile* criado, é adicionada a dependência necessária para este projeto, que é a *LibTorch* na versão 1.6.0.

Para finalizar a etapa referente às dependências apenas é necessário a execução do comando *pod install*, que origina um ficheiro *.xcworkspace*, que será utilizado para abrir o projeto.

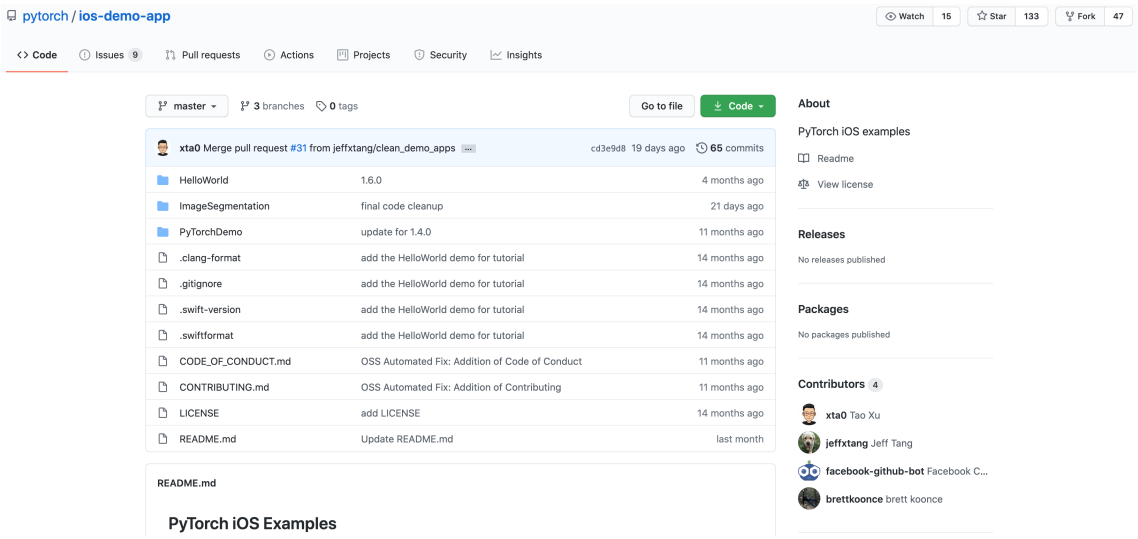

Figura 3.17: Repositório da aplicação demo para iOS (PyTorch, 2020a)

Visto que a biblioteca é desenvolvida utilizando *C++*, é necessária a utilização de um *wrapper*. Primeiro descarrega-se os ficheiros *"TorchModule.h"* e *"TorchModule.mm"* para uma pasta denominada com o nome do projeto. Ao adicionar os ficheiros ao *XCode*, vai ser criado um ficheiro de *Bridging-Header*, no qual é necessário importar o *"TorchModule.h"* (PyTorch, 2020b).

Em relação à UI da aplicação presente na Figura 3.18, é idêntica à presente no repositório exemplo da biblioteca *PyTorch* (PyTorch, 2020a), sendo constituída por: i) uma *UIImageView*, onde a imagem a ser classificada pode ser visualizada; ii) uma *UILabel* a ser utilizada como título; iii) outra *UILabel*, onde aparecem as classes previstas pelo modelo; iv) e por fim um botão referente à implementação do módulo de câmara do dispositivo *iOS* para que seja possível obter a imagem a partir da câmara.

Para que seja possível a utilização da câmara do dispositivo, é necessário a inserção de uma entrada no ficheiro *info.plist*, para que aquando da utilização da aplicação seja pedida ao utilizador permissões de acesso à câmara do seu dispositivo, sem isso, esta funcionalidade não funcionaria. Na Figura 3.19 estão presentes os dois ecrãs da aplicação *iOS*.

De seguida, no ficheiro *ViewController.swift*, como se visualiza na Figura 3.20, são efetuadas as configurações necessárias para prever a imagem proveniente da câmara. Inicialmente, é necessária a inicialização do módulo do *PyTorch*, bastando, para isso, ler o ficheiro referente ao modelo. Para além do enunciado anteriormente, também é necessário o carregamento do ficheiro onde estão presentes a diferentes classes.

Após a captura da imagem proveniente da câmara do dispositivo, e visto que o modelo anteriormente implementado na Figura 3.5 necessita de entrada uma imagem com dimensões de 224 pixeis por 224 pixeis, é efetuado um redimensionamento da imagem, sendo utilizada uma extensão da classe *UIImage* incluída no repositório exemplo. Para

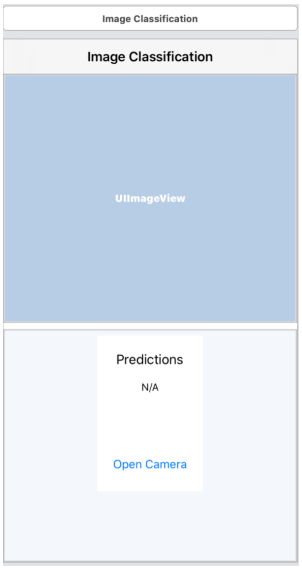

Figura 3.18: *Storyboard* da aplicação *iOS*

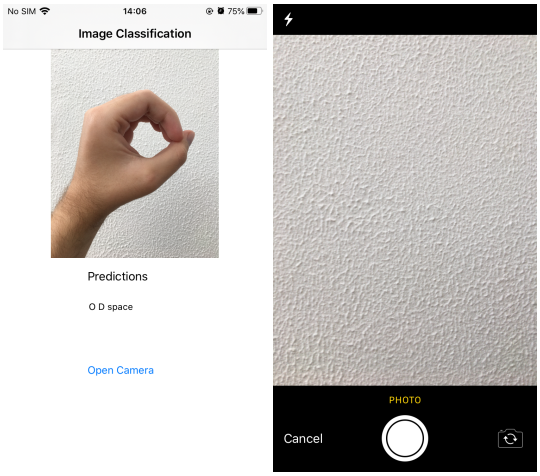

Figura 3.19: Ecrãs da aplicação *iOS*

além do redimensionamento, também é necessário proceder-se à conversão da imagem já redimensionada num *Float32 tensor* que, assim como o redimensionamento da imagem também utiliza a mesma classe que possui a função *normalized*.

Sendo que a imagem já possui as dimensões corretas, basta agora utilizar o módulo da biblioteca *LibTorch*, para prever a classe presente na imagem proveniente da câmara, que está presente na Figura 3.21.

```
private lazy var module: TorchModule = {
    if let filePath = Bundle.main.path(forResource: "model_mob", ofType: "pt"),
        let module = TorchModule(fileAtPath: filePath) {
        return module
    } else {
        fatalError("Can't find the model file!")
    }
}()

private lazy var labels: [String] = {
    if let filePath = Bundle.main.path(forResource: "labels", ofType: "txt"),
        let labels = try? String(contentsOfFile: filePath) {
        return labels.components(separatedBy: .newlines)
    } else {
        fatalError("Can't find the text file!")
    }
}()
```

Figura 3.20: Inicialização do módulo do *PyTorch* e das classes

```
func predict(image: UIImage) {
        let resizedImage = image.resized(to: CGSize(width: 224, height: 224))
        guard var pixelBuffer = resizedImage.normalized() else {
            return
        }
        guard let outputs = module.predict(image: UnsafeMutableRawPointer(&pixelBuffer)) else {
            return
        }
        let sortedResults = zip(labels.indices, outputs).sorted{ $0.1.floatValue > $1.1.floatValue }.prefix(3)
        var text = ""
        for result in sortedResults {
            text += "\(labels[result.0]) "
        }
        resultsLabel.text = text
    }
```

Figura 3.21: Função encarregada de prever a classe

No presente capítulo demonstrou-se a implementação, tanto do modelo classificador como da aplicação móvel *iOS*.

Relativamente à implementação do modelo classificador, foi apresentada uma proposta que refere o conjunto de dados utilizados, o modo como estes dados foram tratados, e uma descrição pormenorizada, passo a passo, das diferentes etapas de desenvolvimento.

No que diz respeito à implementação da aplicação móvel, também aqui foi apresentada uma descrição pormenorizada, passo a passo, das diferentes etapas de desenvolvimento da aplicação móvel.

Capítulo 4

Avaliação do Sistema SignPic

O presente capítulo destina-se a avaliar a nossa abordagem através da execução de testes que serão depois avaliados e discutidos.

Os testes realizados durante a implementação do sistema proposto, foram efetuados utilizando dois equipamentos distintos. O equipamento utilizado para implementar e treinar o modelo classificador foi o equipamento presente na Tabela 4.1.

Na implementação da aplicação móvel, utilizamos um equipamento *Apple*, sendo que as especificações do equipamento em questão encontram-se na Tabela 4.2.

Para comparação foram utilizados diferentes modelos pré-treinados, efetuando *finetuning* e *feature extracting*.

Os modelos utilizados para testes foram a *Resnet18*, *Alexnet*, *VGG11_bn*, *Squeezenet*, *Densenet* e *MobileNet v2*.

De forma a determinar o número épocas necessárias, definimos um valor inicial de 20. Verificamos que após a décima época os valores obtidos começaram a convergir. Posto isto, o valor definido para número de épocas foi o dobro do valor onde a convergência foi adquirida, ou seja, 20. Como se verifica na Figura 4.1, em relação à taxa de aprendizagem utilizada, após diversos testes onde foram utilizadas taxas de aprendizagem de valor 0.1 representada a linha cinza, de valor 0.01 representada a linha verde e de valor 0.001 representada a linha cinzenta, verificou-se que esta última obteve melhores resultados e de forma mais rápida comparada com as demais. Neste teste foi utilizado o modelo *MobileNet v2* e o optimizador *Adam*, efetuando *finetuning*.

De acordo com as estatísticas presentes nas Figuras 4.2 e 4.3, sendo a linha laranja referente ao modelo *Densenet*, a azul ao modelo *Resnet18*, a verde ao modelo *VGG11_bn*,

Tabela 4.1: Especificações do sistema de teste do modelo

Parâmetro	**Valor**
CPU	i5 6600k
GPU	GTX 1060 6GB
OS	Windows 10

Tabela 4.2: Especificações do sistema de teste da aplicação móvel *iOS*

Parâmetro	**Valor**
CPU	i5 5257U
GPU	Iris 6100
OS	macOS Catalina

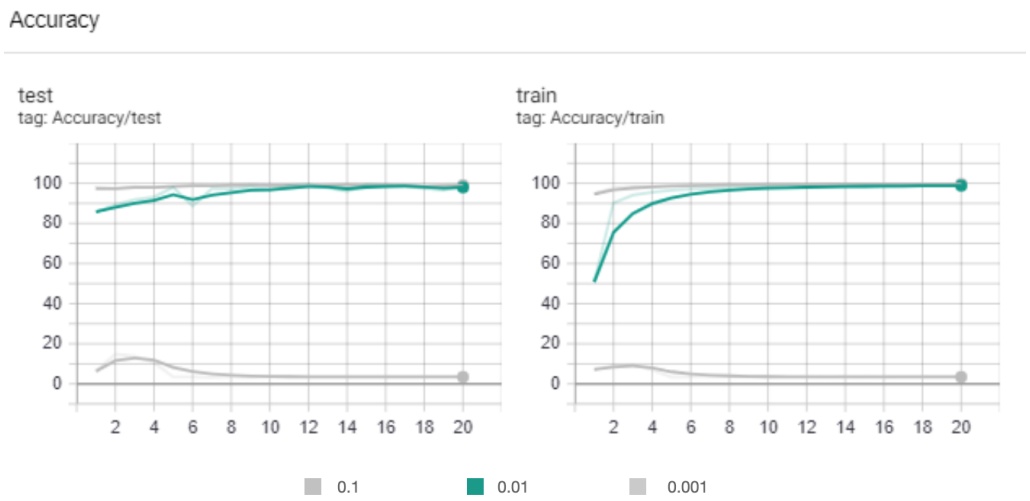

Figura 4.1: Acurácia em treino/teste usando vários valores de taxa de aprendizagem

a vermelha ao modelo *MobileNet v2*, a cinza ao modelo *Squeezenet* e a rosa ao modelo *Alexnet*, onde foi efetuado *finetuning*. Todos os modelos utilizados para comparação obtiveram resultados idênticos, tanto a nível de perda como a nível de acurácia, sendo que neste último os valores compreendem-se entre 98% e 99% aproximadamente, sendo o modelo *Alexnet* a única exceção, tendo obtido valores de acurácia ligeiramente inferiores, cerca de 96%.

As diferenças mais notórias entre os vários modelos prendem-se com o tamanho do modelo gerado e com a duração do treino, como se pode visualizar na Tabela 4.3.

Em relação ao tamanho do modelo, as diferenças ainda são bastante notórias, existindo um modelo com apenas 3Mb e outros até 503Mb. Na duração do treino, os valores são semelhantes, sendo apenas os modelos *Resnet18* e *Alexnet* a obter valores mais curtos. Os demais modelos demoraram cerca 5 horas num treino de 20 épocas.

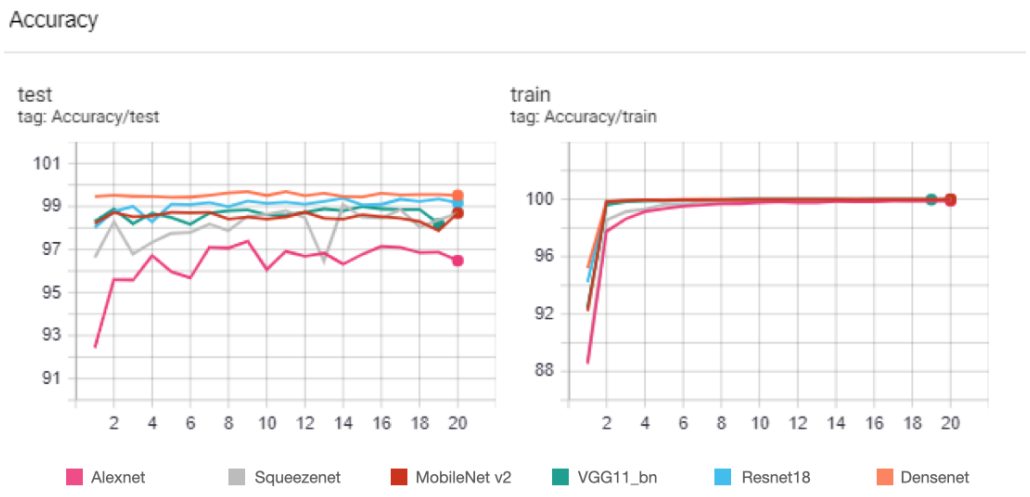

Figura 4.2: Acurácia em treino/teste dos modelos utilizados para comparação

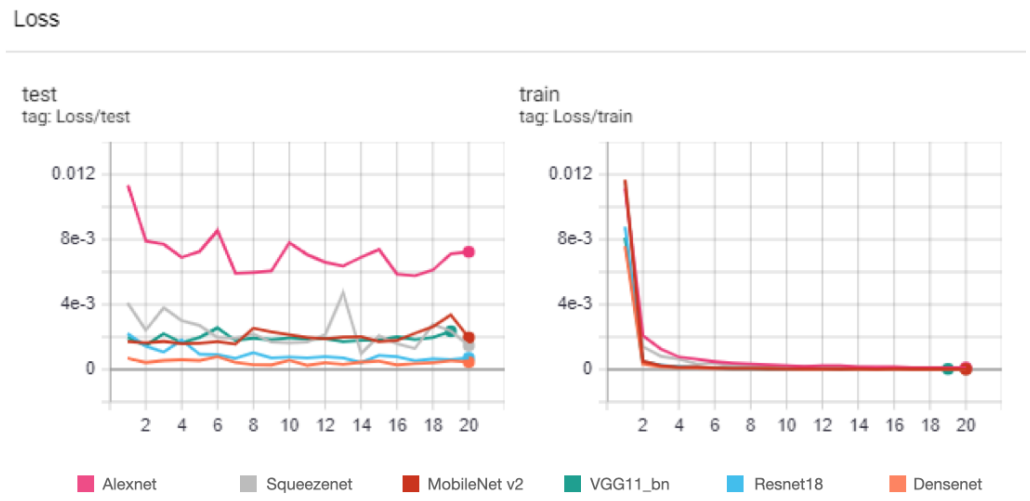

Figura 4.3: Perda em treino/teste dos modelos utilizados para comparação

Tabela 4.3: Comparação de tamanho do modelo e da duração do treino das diferentes redes neuronais efetuando *finetuning*

	Resnet18	**Alexnet**	**VGG11_bn**	**Squeezenet**	**Densenet**	**MobileNet v2**
Tamanho	44 Mb	223 Mb	503 Mb	3 Mb	28 Mb	10 Mb
Duração	2h 39m	1h 30m	5h 36m	5h 56m	5h 53m	5h 23m

Após ter concluído a implementação, foi necessário optimizá-la. Deste modo, foi sendo necessário definir vários parâmetros, o tipo de optimizador, (*Adam* ou o SGD), o número de épocas utilizadas nos testes, 20 épocas, a utilização ou não de uma taxa de aprendizagem fixa, e o tamanho de uma *batch*.

O computador utilizado para treinar o modelo possui cerca de 8GB de RAM, não se mostrando suficiente para utilizar um tamanho de *batch* relativamente alto. Nos testes realizados o valor máximo de *batch* utilizado é de 128. Verificou-se assim uma limitação em termos de hardware.

Efetuando *finetuning*, e suportando-nos no modelo *MobileNet v2*, foram realizados uma série de testes, utilizando dois optimizadores diferentes, o *Adam* e o *SGD*. Os resultados apresentam-se nas Figuras 4.4 e 4.5. Estes testes consistiram em 20 épocas e taxas de aprendizagem fixas e variáveis, ambas com taxa de aprendizagem inicial de 0.001. Nessas Figuras, a linha azul escura refere-se à utilização do optimizador SGD com taxa de aprendizagem variável de 0.1, a linha cinza é referente ao optimizador *Adam* com taxa de aprendizagem fixa, a linha vermelha, azul clara e rosa são referentes à utilização do optimizador *Adam* com taxa de aprendizagem variável de 0.1, 0.5 e 0.8, respetivamente.

Em todos os testes foi possível atingir uma taxa de acurácia de 99%.

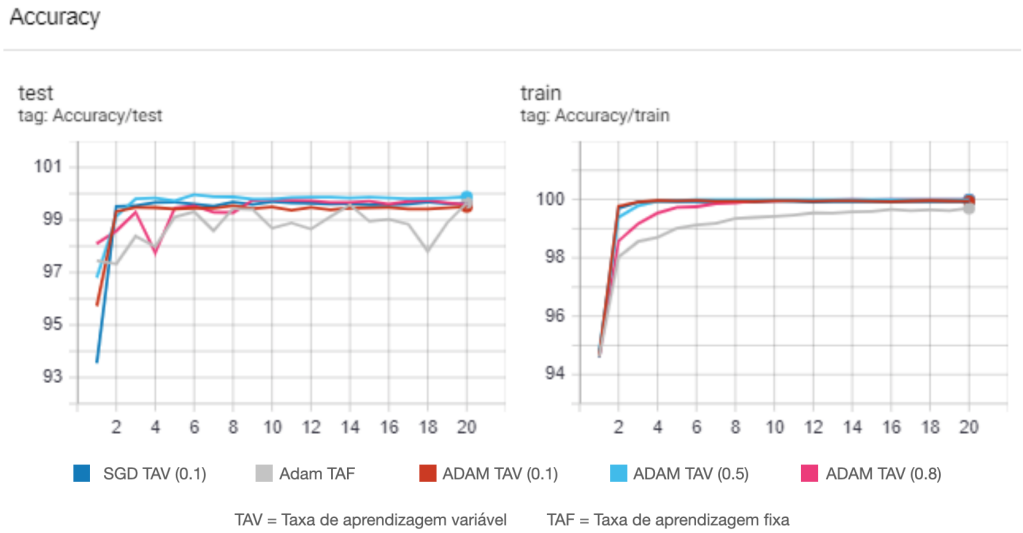

Figura 4.4: Acurácia em treino/teste do modelo *MobileNet v2*

Aquando da utilização de *feature extration* e recorrendo ao optimizador SGD com taxa de aprendizagem fixa, de valor 0.001, e diferentes tamanhos de *batch*, presente na Figura 4.6 e na Figura 4.7, sendo a primeira referente à acurácia obtida e a segunda referente à perda obtida, ambas durante os treinos e os testes. Nessas Figuras, a cor azul clara, vermelha, azul escuro e laranja, são referentes a tamanho de *batch* de 4, 32, 64 e 128, respetivamente.

Analisando os gráficos referentes à acurácia, chegou-se à conclusão que com o aumento do tamanho utilizado para cada *batch*, foi possível atingir um valor mais elevado em termos de acurácia durante o treino. Apesar disso, é necessário ter em consideração que unicamente foram utilizados dados das primeiras 20 épocas. Assim como aconteceu durante o teste, também foi reproduzido durante o treino, sendo que a acurácia foi maior

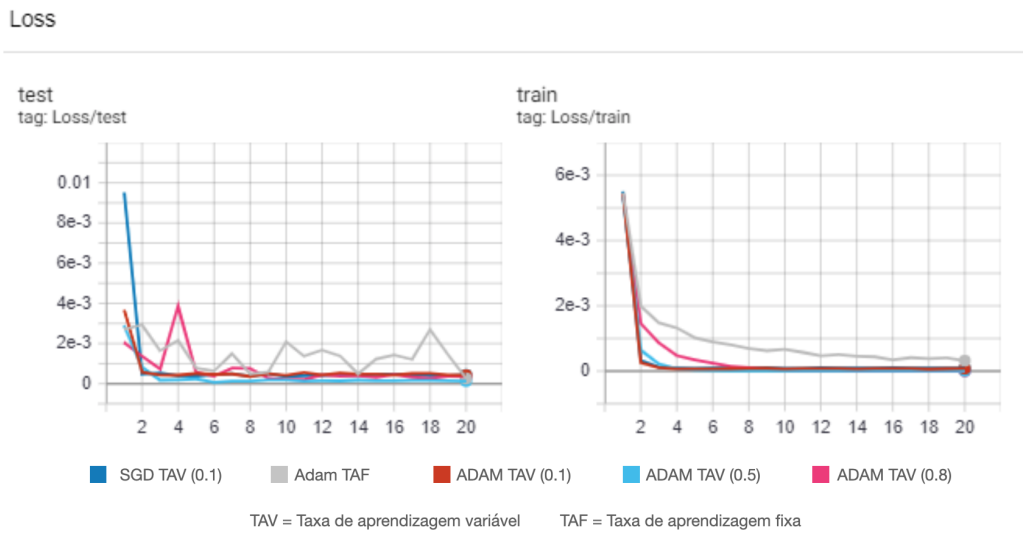

Figura 4.5: Perda em treino/teste do modelo *MobileNet v2*

quanto maior for o valor de *batch* utilizado, com a exceção da utilização de tamanho de *batch* 128, que foi o máximo utilizado durante os testes.

Em relação às perdas, tanto durante o treino como em teste, verificou-se que com o decréscimo do tamanho de *batch* utilizado, as perdas foram menores. No caso de tamanho de *batch*, 4, 32 e 64 a diferença após 20 épocas foi residual, enquanto que a de tamanho 128 foi bastante superior às demais, durante todas as épocas.

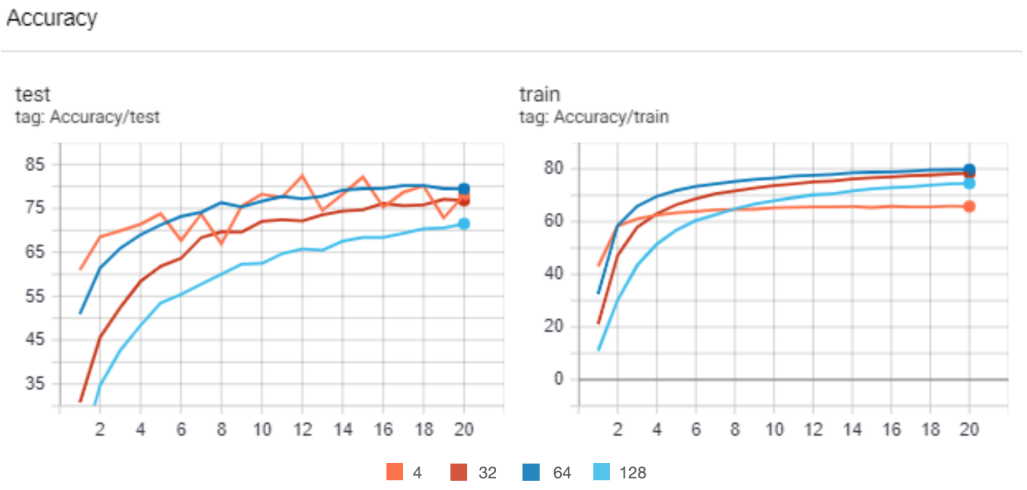

Figura 4.6: Acurácia treino/teste utilizando o optimizador SGD com taxa de aprendizagem fixa em 0.001 com diferentes tamanhos de *batch*

Com a utilização do optimizador SGD, com uma taxa de aprendizagem fixa em 0.001, foi possível atingir uma acurácia de cerca de 81%.

Como referido anteriormente, também foi utilizado o optimizador *Adam* para treinar o modelo, sendo possível visualizar as diferentes estatísticas referentes aos treinos e testes,

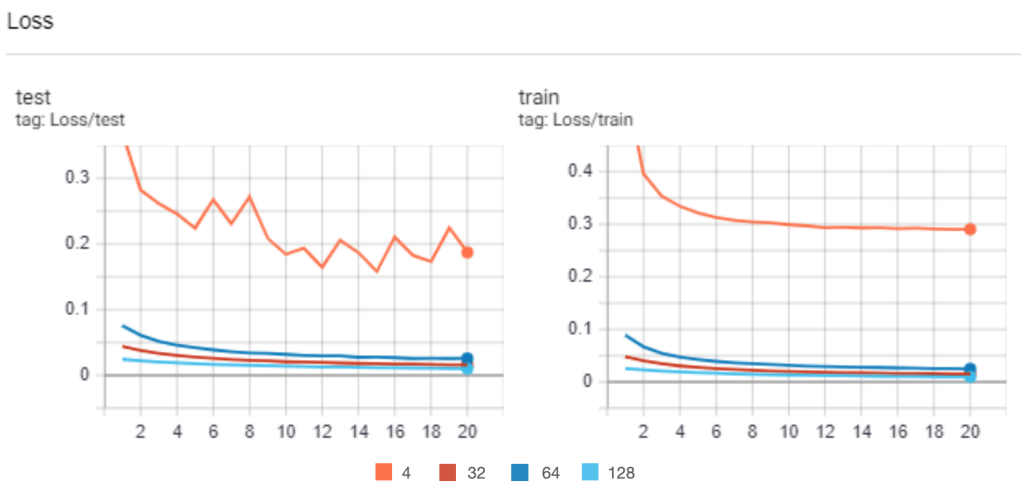

Figura 4.7: Perda treino/teste utilizando o optimizador SGD com taxa de aprendizagem fixa em 0.001 com diferentes tamanhos de *batch*

utilizando uma taxa de aprendizagem fixa de valor 0.001, na Figura 4.8 e na Figura 4.9.

Analisando os gráficos referentes aos testes com este optimizador, em termos de acurácia, tanto durante o treino como durante o teste, visualizou-se que aquando da utilização de um tamanho de *batch* de valor 4, foram obtidos dados bastante díspares entre eles, o que não aconteceu com a utilização do optimizador SGD. Quando se utilizou os demais tamanhos de *batch*, obteve-se valores muito idênticos durante todo o treino e teste.

Em relação aos gráficos referentes à perda, a tendência manteve-se, com exceção do teste utilizando tamanho de *batch* de 4. Constatamos que os valores nos demais tamanhos foram idênticos, aproximando-se do zero.

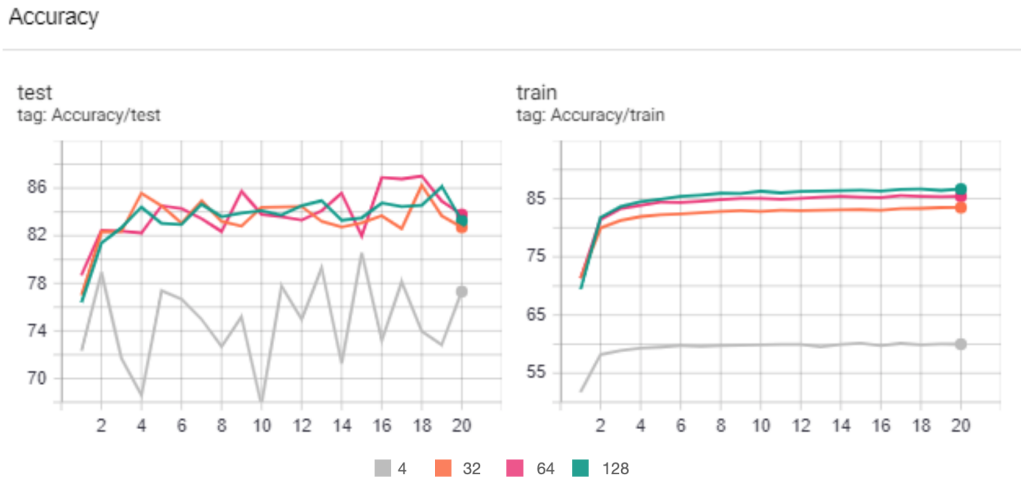

Figura 4.8: Acurácia treino/teste utilizando o optimizador Adam com taxa de aprendizagem fixa em 0.001 com diferentes tamanhos de *batch*

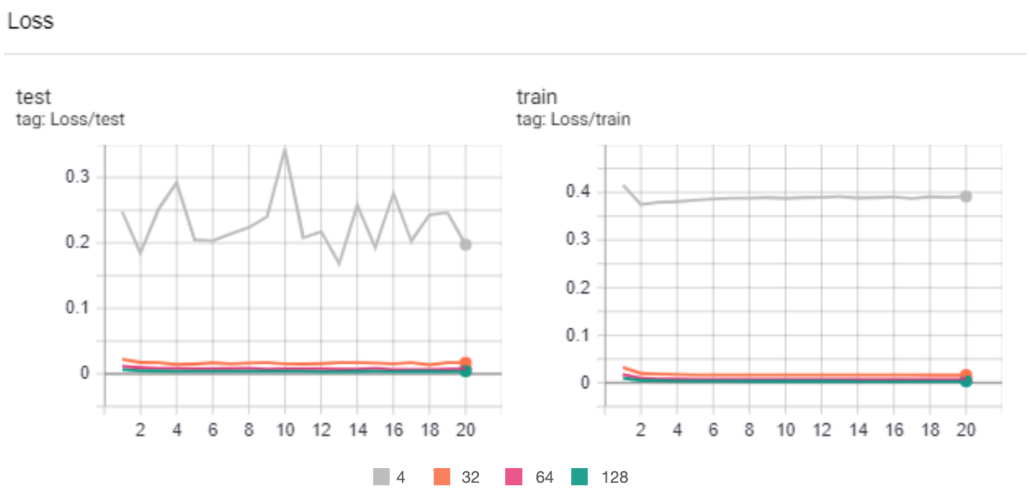

Figura 4.9: Perda treino/teste utilizando o optimizador Adam com taxa de aprendizagem fixa em 0.001 com diferentes tamanhos de *batch*

Aquando da comparação dos dois optimizadores, verificou-se que com a utilização do optimizador *Adam*, foi possível obter uma acurácia mais alta, mais rapidamente. Portanto, testou-se utilizando uma taxa de aprendizagem variável por cada *step*, que equivale, neste caso, a cada época, para ver se resultava em algum tipo de ganho. Sendo assim, a taxa de aprendizagem inicial é de 0.001 e por cada *step*, a mesma é multiplicada por um fator. No caso da cor azul escura, a taxa de aprendizagem foi multiplicada por 0.1, na cor vermelha por 0.5 e por fim na cor azul clara por 0.8. As estatísticas referentes a estes testes podem ser visualizados nas Figuras 4.10 e 4.11.

Figura 4.10: Acurácia treino/teste utilizando o optimizador Adam com taxa de aprendizagem inicial em 0.001 mas variável por cada época

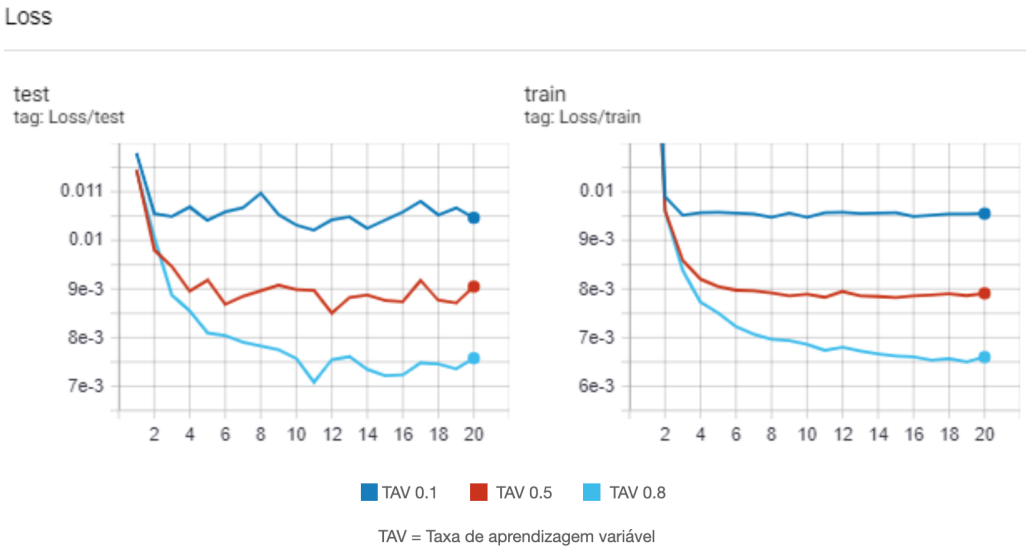

Figura 4.11: Perda treino/teste utilizando o optimizador Adam com taxa de aprendizagem inicial em 0.001 mas variável por cada época

De referir também, que efetuando *feature extraction*, a duração do treino, durante as 20 épocas é menor quando comparado com os dados presentes na Tabela 4.3, como se pode verificar na Tabela 4.4.

Tabela 4.4: Duração do treino das diferentes redes neuronais efetuando *feature extraction*

	Resnet18	**Alexnet**	**VGG11_bn**	**Squeezenet**	**Densenet**	**MobileNetv2**
Duração	1h 31m	1h 13m	2h 44m	1h 44m	2h 51m	1h 54m

Em relação à aplicação *iOS* desenvolvida, foram testados se os resultados efetuados ao nível do modelo classificador poderiam ser corroborados aquando da utilização da aplicação móvel.

De forma a avaliarmos a aplicação móvel proposta, realizamos um primeiro teste onde o mestrando reproduz os gestos do alfabeto da ASL. Repetimos esta experiência usando um vídeo feito por uma pessoa experiente nesta língua onde efetuava todos os mesmos gestos, com a exceção dos símbolos, também incluídos no conjunto de dados. Apresentamos os dados nas Tabelas 4.5 e 4.6.

Tabela 4.5: Avaliação do sistema de deteção de ASL utilizando a aplicação móvel com gestos efetuados pelo mestrando tendo por base a Figura 2.2

ASL	Tentativas	Acertou (%)	Falhou (%)	Notas
"A"	3	0	100	-
"B"	3	100	0	-
"C"	3	100	0	-
"D"	3	0	100	-
"E"	3	0	100	Sempre "S", gesto idêntico
"F"	3	100	0	-
"G"	3	100	0	-
"H"	3	100	0	-
"I"	3	100	0	-
"J"	3	33	66	-
"K"	3	33	66	Dois "V", gesto idêntico
"L"	3	100	0	-
"M"	3	33	66	Dois "N", gesto idêntico
"N"	3	100	0	-
"O"	3	100	0	-
"P"	3	100	0	-
"Q"	3	33	66	Dois "P", gesto idêntico
"R"	3	100	0	-
"S"	3	100	0	-
"T"	3	0	100	Sempre "S", gesto idêntico
"U"	3	100	0	-
"V"	3	100	0	-
"W"	3	100	0	-
"X"	3	0	100	-
"Y"	3	100	0	-
"Z"	3	0	100	-
del	3	100	0	-
space	3	100	0	-
nothing	3	100	0	-

Tabela 4.6: Avaliação do sistema de deteção de ASL utilizando a aplicação móvel com recurso ao vídeo (StartASL, 2020)

ASL	Tentativas	Acertou (%)	Falhou (%)	Notas
"A"	5	0	100	-
"B"	5	100	0	-
"C"	5	0	100	-
"D"	5	0	100	Erro presente no conjunto de dados
"E"	5	0	100	Sempre "T"
"F"	5	0	100	Sempre "W", mas "F"é a classe com a segunda maior probabilidade
"G"	5	0	100	Erro presente no conjunto de dados
"H"	5	0	100	Erro presente no conjunto de dados
"I"	5	80	20	-
"J"	5	0	100	Gesto com movimento
"K"	5	0	100	Sempre "V", gesto idêntico
"L"	5	100	0	-
"M"	5	0	100	Erro presente no conjunto de dados
"N"	5	0	100	Erro presente no conjunto de dados
"O"	5	0	100	Erro presente no conjunto de dados
"P"	5	0	100	Erro presente no conjunto de dados
"Q"	5	20	80	Erro presente no conjunto de dados, diferença ligeira
"R"	5	100	0	-
"S"	5	0	100	Sempre "T", gesto idêntico
"T"	5	60	40	Dois "Z"
"U"	5	100	0	-
"V"	5	100	0	-
"W"	5	100	0	-
"X"	5	60	40	Dois "space"
"Y"	5	80	20	Um "Z"
"Z"	5	0	100	Gesto com movimento

Para o modelo utilizado, com base nos dados presentes nas Tabelas 4.5 e 4.6, verificou-se que nem para todas as imagens de entrada utilizadas foram estimadas devidamente. Esta situação acontece devido: i) às semelhanças entre alguns gestos (e.g., letra M com a letra N, a letra S com a letra T; e ii) às diferenças no ângulo da mão aquando um gesto/símbolo é realizado. Verificamos também que o conjunto de dados utilizado apresenta alguns erros comuns na execução do gestos, como por exemplo as letras D, F e O, que no conjunto de dados é realizado com a palma da mão de lado; quando estes deveriam ser realizados com a palma da mão virada para a frente (Miller, 2018). Por fim, fatores externos, como por exemplo, a luminosidade e o fundo da imagem, podem ter contribuído para que certas previsões não tenham sido as melhores.

Visto que o conjunto de dados apresentava alguns erros na execução de alguns gestos, optou-se por efetuar os gestos de uma forma similar aos presentes no conjunto de dados utilizado, os resultados estão presentes na Tabela 4.7. Estes dados apresentam resultados melhores que os dados presentes nas Tabelas 4.5 e 4.6, o que era um resultado esperado.

Tabela 4.7: Avaliação do sistema de deteção de ASL utilizando a aplicação móvel com gestos efetuados pelo mestrando tendo por base os dados presentes no conjunto de dados utilizado

ASL	**Tentativas**	**Acertou (%)**	**Falhou (%)**
"A"	5	0	100
"B"	5	100	0
"C"	5	100	0
"D"	5	100	0
"E"	5	0	100
"F"	5	100	0
"G"	5	100	0
"H"	5	100	0
"I"	5	80	20
"J"	5	20	80
"K"	5	100	0
"L"	5	100	0
"M"	5	0	100
"N"	5	100	0
"O"	5	100	0
"P"	5	100	0
"Q"	5	100	0
"R"	5	100	0
"S"	5	100	0
"T"	5	100	0
"U"	5	100	0
"V"	5	100	0
"W"	5	100	0
"X"	5	100	0
"Y"	5	100	0
"Z"	5	100	0
del	5	100	0
space	5	100	0
nothing	5	100	0

Na Figura 4.12, pode-se visualizar o utilizador a efetuar o gesto referente à letra *"O"* no alfabeto da ASL, capturado pela câmara do dispositivo e este prevendo com sucesso a letra *"O"* presente efetivamente na imagem.

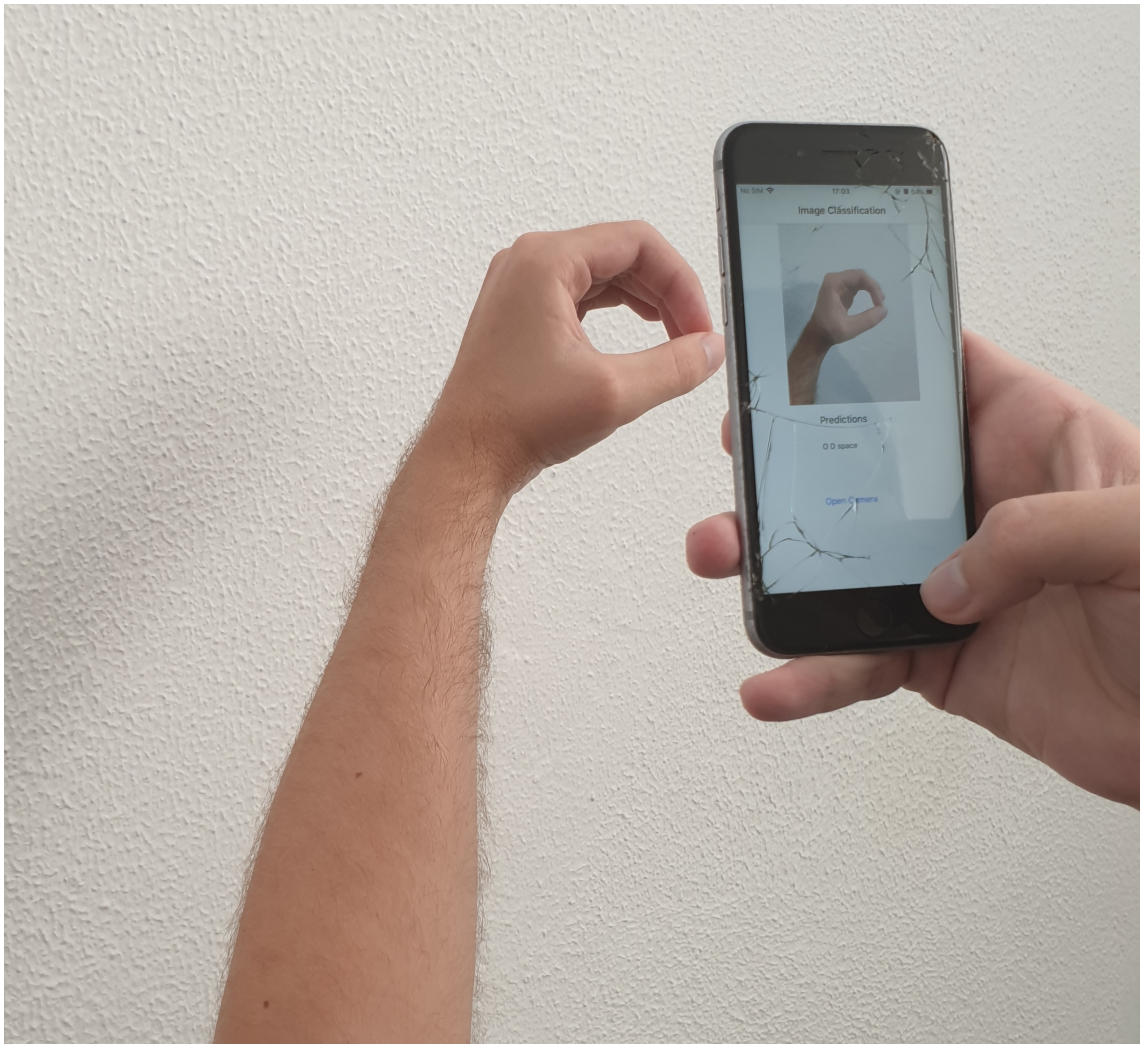

Figura 4.12: Teste da aplicação *iOS* referente à letra O

No presente capítulo, foi possível concluir que utilizando este conjunto de dados, obteve-se uma melhor acurácia, aquando da utilização de *finetuning* com o optimizador *Adam*, com taxa de aprendizagem inicial de 0.001, sendo a mesma variável, época após época, num fator de 0.5, como se verifica na Figura 4.4.

Utilizando estes parâmetros foi possível alcançar valores de acurácia na validação a rondar 99%, uma melhoria quando comparado com a utilização do optimizador SGD e também, comparando com a utilização do optimizador *Adam* com taxa de aprendizagem fixa.

Capítulo 5

Conclusão

No início desta dissertação foi delineado como objetivo, a criação de um sistema capaz de detetar gestos referentes à LGP efetuados por um indivíduo, apenas recorrendo a um *smartphone*. Ao utilizador basta abrir a câmara, a partir da aplicação móvel, e capturar uma imagem referente ao gesto que pretende que seja traduzido.

Para que o objetivo fosse atingido foi necessário proceder à implementação e treino de um modelo. O modelo utilizado foi o *MobileNet v2*, presente no *Zoo* de modelos da biblioteca utilizada para implementação, a *PyTorch*.

Diversas dificuldades foram identificadas, sendo as mais impactantes: i) a escolha do modelo para classificar; ii) a quantidade de testes necessários executar, com variadas funções e parâmetros diferentes, para que fosse possível atingir o melhor resultado, dentro da coletânea de testes realizados; iii) e o tempo de execução dos diversos testes, mesmo utilizando 20 iterações nas épocas.

Durante os testes foi possível atingir acurácia de 99% em validação, aquando da utilização do optimizador *Adam* com taxa de aprendizagem de 0.001 conjuntamente com o *CrossEntropyLoss* como função de perda.

Após ultrapassadas as dificuldades ocorridas durante a fase de implementação do modelo, o desenvolvimento e o funcionamento da aplicação móvel para sistemas operativos *iOS* correspondeu ao expectável.

De concluir que, como não foi efetuada uma deteção da mão e posterior corte da mesma a partir da imagem proveniente da câmara do *smartphone*, caso a imagem não possua estritamente a mão, a deteção do gesto efetuado, dificilmente é reconhecida.

5.1 Trabalho futuro

A aplicação móvel desenvolvida, apesar de ser funcional está longe de ser um sistema capaz de colmatar a barreira de comunicação entre pessoas que comunicam utilizando LG com pessoas que comunicam usando língua oral.

Apesar de se ter atingido uma acurácia elevada em validação, esta não se reproduz em

situações do dia-a-dia. Embora as imagens presentes no conjunto de dados utilizado para treinar o classificador possuam diversidade em termos de luminosidade, posição da mão na mesma, apenas possuem a mão a efetuar o gesto. Como não foi desenvolvido qualquer sistema capaz de fazer o recorte da mão da imagem de entrada do classificador, o sistema nem sempre consegue detetar com sucesso o gesto.

Outros aspetos a considerar para trabalho futuro são a introdução de gestos não estáticos, a introdução de mecanismos capazes de classificar, não só gestos efetuados pelas mãos dos utilizadores mas também expressões faciais, movimentos corporais, que são também utilizados para comunicação em LG.

A adição de mais gestos para enriquecer a panóplia existente é importante, porque os gestos presentes neste sistema apenas se referem maioritariamente a letras do alfabeto ASL.

Referências Bibliográficas

M. P. de Almeida. Língua de sinais x libras: uma abordagem da historiografia linguística. https://docplayer.com.br/4731519-Lingua-de-sinais-x-libras-uma-abordagem-da-historiografia-linguistica.html, 2014. [Online; acedido em 2020-11-10]. 7

M. A. Amaral, A. Coutinho, and M. R. D. Martins. *Para Uma Gramática da Língua Gestual Portuguesa*. Caminho, 1994. ISBN 9789722109819. 7

Apple. Core ml. https://developer.apple.com/documentation/coreml, s.d. [Online; acedido em 2020-11-19]. 18

D. Armstrong and S. Wilcox. *The Gestural Origin of Language*. Oxford University Press, 2007. doi: 10.1093/acprof:oso/9780195163483.001.0001. 6

Associação de Surdos do Porto. A língua gestual. http://www.asurdosporto.org.pt/artigo.asp?idartigo=71, s.d. [Online; acedido em 2020-11-22]. 8

K. Bantupalli and Y. Xie. American sign language recognition using deep learning and computer vision. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 4896–4899, 2018. doi: 10.1109/BigData.2018.8622141. 14

Gong Chao. Human-computer interaction: Process and principles of human-computer interface design. In *Proceedings - 2009 International Conference on Computer and Automation Engineering, ICCAE 2009*, pages 230–233, 2009. ISBN 9780769535692. doi: 10.1109/ICCAE.2009.23. 1

B. M. Chiari. *Língua e Linguagem - Forma, Conteúdo e Uso na Presença dos Déficits de Audição*, pages 632–634. Editora Roca, São Paulo, 2014. ISBN 9788527726412. 5

Teak-Wei Chong and Boon Giin Lee. American sign language recognition using leap motion controller with machine learning approach. *Sensors*, 18:3554, 10 2018. doi: 10.3390/s18103554. viii, 9

A. Guerreiro. *Comunicação e Cultura Inclusivas*. Edições Universitárias Lusófonas., 2012. 5

N. Inkawhich. Finetuning torchvision models. https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html, s.d. [Online; acedido em 2020-11-19]. 22

Jie Huang, Wengang Zhou, Houqiang Li, and Weiping Li. Sign language recognition using 3d convolutional neural networks. In *2015 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2015. viii, 12, 13, 15, 16

A. Karpathy. Pytorch at tesla. https://www.youtube.com/watch?v=oBklltKXtDE, 2019. [Online; acedido em 2020-11-19]. 18

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014. viii, 16, 17

P. T. Krishnan and P. Balasubramanian. Detection of alphabets for machine translation of sign language using deep neural net. In *2019 International Conference on Data Science and Communication (IconDSC)*, pages 1–3, 2019. viii, 11

Scott K. Liddell. *Grammar, Gesture, and Meaning in American Sign Language*. Cambridge University Press, 2003. doi: 10.1017/CBO9780511615054. 6

John Miller. Common fingerspelling mistakes new signers make. https://www.signingsavvy.com/blog/251/Common+Fingerspelling+Mistakes+New+Signers+Make, 2018. [Online; acedido em 2020-12-23]. 47

National Association of the Deaf. States that recognize american sign language. https://www.nad.org/wp-content/uploads/2018/06/List_States_Recognizing_ASL.pdf, 2016. [Online; acedido em 2020-11-22]. 8

André Nogueira. Signpic implementation. https://github.com/andredsnogueira/thesis, 2020. [Online; acedido em 2020-12-05]. 20

OpenCV. Opencv. https://docs.opencv.org/master/d1/dfb/intro.html, s.d. [Online; acedido em 2020-11-19]. 17

In-Kwon Park, Jung-Hyun Kim, and Kwang-Seok Hong. An implementation of an fpga-based embedded gesture recognizer using a data glove. In *Proceedings of the 2nd international conference on Ubiquitous information management and communication*, pages 496–500, 2008. 10

Lionel Pigou, Sander Dieleman, Pieter-Jan Kindermans, and Benjamin Schrauwen. Sign language recognition using convolutional neural networks. volume 8925, pages 572–578, 03 2015. doi: 10.1007/978-3-319-16178-5_40. viii, 13

PyTorch. Pytorch ios examples. https://github.com/pytorch/ios-demo-app/tree/master/HelloWorld/HelloWorld, 2020a. [Online; acedido em 2020-11-26]. viii, 33

PyTorch. Pytorch mobile runtime for ios. https://www.youtube.com/watch?v=JFy3uHyqXn0, 2020b. [Online; acedido em 2020-11-26]. 33

PyTorch. Pytorch. https://pytorch.org/docs/stable/index.html, s.d. [Online; acedido em 2020-11-19]. 18

Dan Rasband. Asl alphabet test. https://www.kaggle.com/danrasband/asl-alphabet-test/home, 2018. [Online; acedido em 2020-09-29]. 10

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning Representations by Back-Propagating Errors*, page 696–699. MIT Press, Cambridge, MA, USA, 1988. ISBN 0262010976. 15

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. pages 4510–4520, 06 2018. doi: 10.1109/CVPR.2018.00474. viii, 23

W. Sandler and D. Lillo-Martin. *Sign language and linguistic universals*. Cambridge University Press, 2006. 8

F. Saussure. *Curso de Linguística Geral*. Publicações D. Quixote, 1978. 6

Saze. alphabets sign language. https://www.kaggle.com/amarinderplasma/alphabets-sign-language, 2018. [Online; acedido em 2020-09-29]. viii, 25

I. Sim-Sim, I. Duarte, and M. Ferroz. *A Língua Materna na Educação Básica. Competências Nucleares e Níveis de Desempenho*. Ministério da Educação: Departamento da Educação Básica, 1997. 6

I. Sim-Sim, A. C. Silva, and C. Nunes. *Linguagem e Comunicação no Jardim-de-Infância*. Ministério da Educação: Direcção-Geral de Inovação e de Desenvolvimento Curricular, 2008. 6

C. B. Skliar. *A Educação para os Surdos entre a Pedagogia Especial e Políticas para as Diferenças*, page 32–47. Líttera Maciel Ltda, Rio de Janeiro, 1997. 8

T. C. C. Souza. *Língua e Escrita: uma questão de história*, page 27–31. Líttera Maciel Ltda, Rio de Janeiro, 1997. 7

StartASL. American sign language alphabet. https://www.startasl.com/american-sign-language-alphabet/, 2020. [Online; acedido em 2020-11-26]. x, 46

W. C. Stokoe. *Língua gestual como primeira língua da Humanidade*, pages 339–348. Editorial Caminho, Lisboa, 2006. 5

M. Taskiran, M. Killioglu, and N. Kahraman. A real-time system for recognition of american sign language by using deep learning. In *2018 41st International Conference on Telecommunications and Signal Processing (TSP)*, pages 1–5, 2018. viii, 1, 11, 12

TensorFlow. Tensorflow. `https://www.tensorflow.org/learn`, s.d. [Online; acedido em 2020-11-19]. 18

L. S. Vygotsky. *Pensamento e linguagem*. Martins Fontes, 1993. 5

Wikipédia. Língua gestual portuguesa. `https://pt.wikipedia.org/wiki/L%C3%ADngua_gestual_portuguesa#/media/Ficheiro:Alfabeto_Manual_LGP.jpg`, s.d. [Online; acedido em 2020-11-17]. viii, 9

World Federation of the Deaf. Our story. `https://wfdeaf.org/who-we-are/our-story`, s.d. [Online; acedido em 2020-11-22]. 8

Rikiya Yamashita, Mizuho Nishio, Richard Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9, 06 2018. doi: 10.1007/s13244-018-0639-9. 15