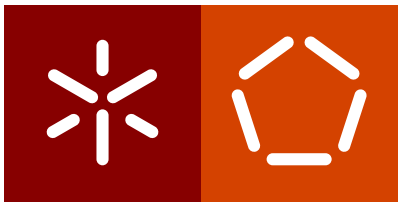
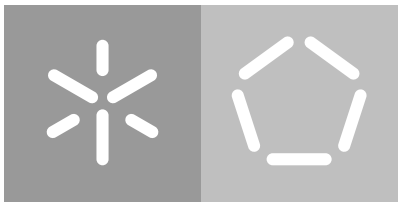


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Tiago André Alves Bouças

Conversão de Esboços de Páginas Web para HTML usando Aprendizagem Automática

Dezembro 2019



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Tiago André Alves Bouças

Conversão de Esboços de Páginas Web para HTML usando Aprendizagem Automática

Dissertação de Mestrado
em Engenharia Informática

Dissertação orientada por
António Joaquim André Esteves

Dezembro 2019

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

AGRADECIMENTOS

Na realização da presente dissertação, muitas foram as pessoas que influenciaram direta e indiretamente e às quais estou profundamente grato.

Ao orientador o Professor António Joaquim André Esteves, aqui lhe exprimo a minha gratidão pelo apoio prestado, pela disponibilidade e incentivo.

À minha família agradeço pelo apoio financeiro, a força e o carinho que demonstraram em toda a minha vida académica.

Aos meus pais, prima e amigos agradeço pelo contributo dado na construção de esboços de páginas Web, e que deste modo ajudaram a desenvolver o conjunto de dados.

À minha namorada pela sua paciência, companhia e apoio prestado durante a realização da dissertação.

Não poderia deixar de agradecer também a todos os meus amigos que me acompanharam e apoiaram em toda a minha vida académica.

A todos o meu profundo e sincero **Obrigado!**

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

ABSTRACT

In the last decades, there has been an exponential development in the area of computing, which includes Artificial Intelligence (AI). The development of AI translates into the emergence of programs that replicate the ability to make decisions, perceive and solve problems in a similar way to humans. Today, artificial intelligence is already part of various areas of society, such as security, health, or virtual assistants.

This dissertation aimed to develop a Web application that converts graphical interface sketches, elaborated with the `Balsamiq Mockups` application, into HTML, CSS and Bootstrap code. Converting a Web page sketch into code is a task that developers typically perform. Due to the time consuming of this task, it becomes impossible to devote more time to the application logic. On the other hand, it is a repetitive and tedious task.

Two deep neural network models were built, divided into two distinct approaches. The first approach, presenting poor results, uses a convolutional network and two recurring networks, according to an encoder-decoder architecture, similar to image captioning. It also uses a DSL language and a compiler that transforms DSL into code. The second approach is completely different and it is more focused on the spatial component of the addressed task. It uses YOLO model and a layout algorithm that converts the output of YOLO into code.

In the same test set, the first approach achieves 71.30% accuracy, while in the second approach it yields 88.28% accuracy.

The Web application, which allows the user to upload images and automatically generate HTML, CSS and Bootstrap code, is supported by the YOLO based model as it gives better results.

Keywords: Artificial Intelligence, neural network, deep, convolutional network, recurring network, YOLO, Web application

RESUMO

Nas últimas décadas assistiu-se a um desenvolvimento exponencial na área da computação, na qual está incluída a inteligência artificial. O desenvolvimento da inteligência artificial traduz-se no aparecimento de programas que replicam a capacidade de decisão, perceção e resolução de problemas do ser humano. Atualmente, a inteligência artificial já faz parte de várias áreas da sociedade, como a segurança, a saúde ou os assistentes virtuais.

A presente dissertação tinha como objetivo desenvolver uma aplicação Web que permitisse converter esboços de interfaces gráficas, elaborados com a aplicação `Balsamiq Mockups`, em código HTML, CSS e Bootstrap. A conversão de esboços de páginas Web em código é uma tarefa normalmente realizada pelos programadores. Devido ao tempo necessário para realizar esta tarefa, reduz-se o tempo disponível para dedicar à lógica da aplicação. Por outro lado, a aplicação a desenvolver eliminaria boa parte de uma tarefa repetitiva e tediosa.

Construíram-se dois modelos de aprendizagem profunda, resultado de duas abordagens distintas. A primeira abordagem, com piores resultados, utiliza uma rede neuronal convolucional e duas redes recorrentes segundo uma arquitetura codificador-descodificador, semelhante ao que se costuma adotar na legendagem de imagens. Utiliza ainda uma linguagem DSL e um compilador que transforma a DSL em código HTML, CSS e Bootstrap. A segunda abordagem, completamente diferente e mais focada na componente espacial do problema a resolver, consiste na utilização do YOLO e um algoritmo de *layout* que converte a saída do YOLO em código HTML, CSS e Bootstrap.

No mesmo conjunto de teste e de acordo com as métricas desenvolvidas para avaliar os modelos, a primeira abordagem resulta em 71.30% de correção, enquanto que a segunda abordagem permitiu alcançar resultados muito superiores (88.28%).

A aplicação Web permite ao utilizador carregar imagens e gerar automaticamente o código HTML, CSS e Bootstrap. A aplicação é suportada pelo modelo que resultou da segunda abordagem e que apresenta melhores resultados.

Palavras-Chave: Inteligência artificial, rede neuronal, profunda, convolucional, recorrentes, YOLO, aplicação web

CONTEÚDO

Acrónimos xiii

1 INTRODUÇÃO 1
- 1.1 Enquadramento 2
- 1.2 Objetivos 3
- 1.3 Estrutura da tese 3

2 ESTADO DA ARTE 5
- 2.1 Inteligência Artificial 5
- 2.2 Aprendizagem automática 6
- 2.3 Aprendizagem profunda 7
- 2.4 Redes Neuronais Artificiais 7
  - 2.4.1 Arquiteturas das Redes Neuronais 9
  - 2.4.2 Algoritmo de retropropagação 11
  - 2.4.3 Funções de Ativação 12
  - 2.4.4 Função de Custo 14
  - 2.4.5 Problema de *vanishing gradients* 16
  - 2.4.6 Otimizadores 16
    - Gradiente estocástico descendente (SGD) 17
    - Adagrad 17
    - Adadelta 18
    - RMSProp 18
    - Adam 19
  - 2.4.7 Métricas 19
  - 2.4.8 Hiperparâmetros 21
    - Hiperparâmetros de Estrutura 21
    - Hiperparâmetros de Treino 21
  - 2.4.9 Redes Neuronais Convolucionais 22
  - 2.4.10 Redes Neuronais Recorrentes 25
    - Long Short-Term Memory 26
    - Gated Recurrent Unit 27
    - Técnicas para melhorar o desempenho das RNNs 28
  - 2.4.11 Generative Adversarial Networks 28
- 2.5 Engenharia de Atributos 29
- 2.6 Overfitting e Underfitting 30

2.7 Técnicas de Avaliação Cruzada 32
2.8 Otimização de modelos 33
2.9 Passos para criar um modelo 35
2.10 Trabalho Relacionado 39
3 TECNOLOGIAS 43
3.1 Conjunto de dados 43
3.1.1 Balsamiq Mockups 43
3.1.2 LabelImg 44
3.2 Servidor e Modelos de aprendizagem profunda 45
3.2.1 Python 45
3.2.2 TensorFlow 45
3.2.3 Keras 46
3.2.4 REST 46
3.2.5 Flask 47
3.3 Aplicação Web 49
3.3.1 HTML 50
3.3.2 CSS 50
3.3.3 JavaScript 50
3.3.4 Bootstrap 51
3.3.5 ReactJS 52
4 ANÁLISE DO PROBLEMA 56
4.1 Levantamento de requisitos 56
4.1.1 Análise de domínio 57
4.1.2 Entrevistas e Questionários 58
4.1.3 Brainstorming 58
4.2 Requisitos da aplicação Web desenvolvida 59
4.2.1 Requisitos funcionais 59
4.2.2 Requisitos não-funcionais 59
4.2.3 Restrições técnicas 60
4.3 Protótipos da Interface Web 60
4.3.1 Protótipos de baixa fidelidade 60
4.3.2 Protótipos de alta fidelidade 62
5 DESENVOLVIMENTO DOS MODELOS DE APRENDIZAGEM PROFUNDA 64
5.1 Conjunto de Dados 64
5.2 Abordagem 1 com CNN e RNNs 70
5.2.1 Pré-processamento do conjunto de dados 70
5.2.2 Linguagem específica de um domínio 71
5.2.3 Arquitetura do modelo desenvolvido 72

5.2.4 Compilador 74
5.2.5 Métrica de avaliação 75
5.3 Abordagem 2 com YOLO e Algoritmo de *Layout* 78
5.3.1 YOLO V1 78
5.3.2 YOLO V2 83
5.3.3 YOLO V3 84
5.3.4 Algoritmo de Layout 85
5.3.5 Arquitetura do Modelo 87
5.3.6 Métrica de Avaliação 88
5.4 Desenvolvimento da Aplicação Web 89
6 EXPERIÊNCIAS E RESULTADOS 93
6.1 Experiências com a abordagem 1 93
6.2 Experiências com a abordagem 2 96
6.3 Resultados 100
7 CONCLUSÕES E TRABALHO FUTURO 102
7.1 Conclusões 102
7.2 Trabalho futuro 104

A ESBOÇO E CÓDIGO HTML GERADO PELO MODELO 110

LISTA DE FIGURAS

1	Representação de um neurónio natural.	8
2	Funcionamento de um neurónio numa rede neuronal artificial.	8
3	Rede neuronal *feed-forward* de camada única.	9
4	Rede neuronal *feed-forward* de múltiplas camadas.	10
5	Rede neuronal recorrente.	10
6	Função de custo que ilustra o problema de *vanishing gradients*.	16
7	Arquitetura de uma Convolutional Neural Network (CNN).	22
8	Operação de convolução.	23
9	Criar novas imagens com a técnica de aumento de dados.	24
10	Célula LSTM.	26
11	Célula GRU.	28
12	Resultado da aplicação de uma GAN com o estilo de Van Gogh.	29
13	Um exemplo de *overfitting*.	31
14	Convolução normal.	34
15	Convolução *depth-wise*.	35
16	Arquitetura do projeto.	40
17	Funcionamento da ferramenta LabelImg.	44
18	Página inicial do sítio Web Sketch2Code da Microsoft.	57
19	Protótipo de baixa fidelidade da página principal.	61
20	Protótipo de baixa fidelidade do editor de código.	62
21	Protótipo de alta fidelidade da página principal.	63
22	Representação de uma imagem.	65
23	Representação de um vídeo.	65
24	Representação de um *slideshow*.	65
25	Representação de uma tabela.	65
26	Representação de um bloco de texto.	66
27	Representação de um título.	66
28	Representação de um botão.	66
29	Representação de um botão de seleção.	66
30	Representação de um seletor de data.	66
31	Representação de uma barra de pesquisa.	66
32	Exemplo de uma barra de navegação superior.	66
33	Representação de uma barra de navegação lateral.	67

34	Representação de uma ligação (*link*).	67
35	Representação de uma etiqueta (*label*).	67
36	Representação de uma caixa para introdução de texto.	67
37	Exemplo de um esboço presente no conjunto de dados.	68
38	Arquitetura do modelo durante a fase de treino.	72
39	Arquitetura do modelo durante a fase de predição.	73
40	Cálculo do `Bleu Score`.	76
41	Imagem dividida numa grelha 4×4.	79
42	Ilustração da métrica *interseção sobre a união (IoU)*.	80
43	Compensações no cálculo de regiões delimitadoras.	84
44	Interseção de dois retângulos.	86
45	Arquitetura do modelo da abordagem 2 em treino.	87
46	Arquitetura do modelo da abordagem 2 na fase de predição.	88
47	Ficheiros resultantes do comando `create-react-app`.	90
48	Página principal desenvolvida em ReactJS.	91
49	Editor de código desenvolvido em ReactJS.	91
50	Primeiro exemplo de predição na abordagem 1.	95
51	Segundo exemplo de predição na abordagem 1.	95
52	Terceiro exemplo de predição na abordagem 1.	96
53	Valor do erro nas primeiras 15 iterações.	97
54	Valor do erro nas últimas iterações.	97
55	Exemplo 1 de predição com a abordagem 2.	98
56	Exemplo 2 de predição com a abordagem 2.	99
57	Exemplo 3 de predição com a abordagem 2.	99
58	Exemplo 4 de predição com a abordagem 2.	100
59	Esboço fornecido ao modelo.	110

LISTA DE TABELAS

1	Funções de ativação e custo mais adequadas a cada problema.	36
2	Sumário das experiências realizadas na abordagem 1.	94
3	Sumário das experiências com 20 iterações na abordagem 1.	94
4	Pontuação de semelhança do modelo no subconjunto de teste.	98

LISTA DE LISTAGENS

3.1 Hello world em Flask. 47
3.2 Hello World em Flask-RESTful. 49
3.3 Exemplo de utilização do Bootstrap. 52
3.4 Hello World em ReactJS. 53
3.5 Exemplo de um componente ReactJS com estado interno. 54

5.1 Código DSL correspondente ao esboço da figura 37. 68
5.2 Código XML relativo ao esboço da figura 37. 69
5.3 Exemplo de código DSL criado. 71

A.1 Código gerado pelo modelo da segunda abordagem. 111

ACRÓNIMOS

A

ADALINE Adaptive Linear Neuron. 11, 13

API Application Programming Interface. 3, 40–42, 45–48, 59, 60, 90, 92

AST abstract syntax tree. 75

C

CNN Convolutional Neural Network. ix, 3, 5, 22–24, 39, 41, 68, 70, 72–74, 82, 93, 94, 102–104

CNTK Microsoft Cognitive Toolkit. 46

CPU Central Processing Unit. 46

CSS Cascading Style Sheets. 1, 3, 39, 43, 49–53, 70, 71, 78, 85, 87, 88, 93, 103, 110

CUDA Compute Unified Device Architecture. 93, 100

D

DOM Document Object Model. 53

DSL Domain-specific language. 4, 43, 64, 67, 68, 70–75, 93, 100, 102–104

G

GAN Generative Adversarial Networks. 3, 5, 28, 29

GPU Graphics Processing Unit. 41, 46, 85, 100, 101

GRU Gated Recurrent Units. 26–28, 68, 73, 93, 100, 101

GUI Graphical User Interface. 1

H

HTML HyperText Markup Language. 1, 3, 39, 40, 43, 47–53, 57, 59, 61, 64, 70–72, 75, 78, 85–88, 90, 91, 93–95, 101–104, 110

HTTP HyperText Transfer Protocol. 46, 48, 49

I

IOT Internet of things. 2

IOU Intersection over Union. 79, 80, 84, 89

J

JSON JavaScript Object Notation. 47, 49

L

LESS Leaner Style Sheets. 51

LSTM Long Short-Term Memory. 26, 27, 43, 93, 100, 101, 104

M

MADALINE Many Adaptive Linear Neuron. 11

MAE Mean Absolute Error. 15, 20

MSE Mean Squared Error. 15

N

NOSQL Not Only Structured Query Language. 48

O

OCR Optical Character Recognition. 39

ORM Object-Relational Mapping. 47

R

REMAUI Reverse Engineer Mobile Application User Interfaces. 39

REST REpresentational State Transfer. 3, 40, 42, 45–49, 59, 60

RMSE Root mean squared error. 20

RNN Recurrent Neural Network. vi, 3, 5, 26, 28, 39, 40, 68, 70, 72–74, 93, 94, 100, 102–104

S

SASS Syntactically Awesome Stylesheet. 51, 52

SPA Single Page Application. 49, 52, 53, 89

SQL Structured Query Language. 71

SVM Support Vector Machine. 42

U

URI Uniform Resource Identifier. 47

URL Uniform Resource Locator. 48, 49, 88, 92

V

VH Viewport Height. 85

VMAX Viewport Maximum. 85

VMIN Viewport Minimum. 85

VOC Visual Object Classes. 44

VW Viewport Width. 85

X

XML Extensible Markup Language. 44, 47, 64, 67, 69, 100, 103

Y

YOLO You Only Look Once. 1, 4, 43, 44, 64, 69, 78–80, 82–85, 87, 88, 96, 98, 101–104

1

INTRODUÇÃO

Neste capítulo introduz-se o tema da conversão automática de esboços de páginas web em código, com o intuito de fornecer uma visão geral do problema e os objetivos propostos inicialmente. Discute-se a importância da criação de uma aplicação com esta finalidade, quais os objetivos principais da dissertação e, termina-se com a organização da dissertação através de uma breve descrição de cada capítulo.

Para concretizar o objetivo proposto inicialmente, desenvolveu-se uma aplicação Web que permite converter esboços de interfaces gráficas, elaborados com a aplicação Balsamiq Mockups, em código HyperText Markup Language (HTML), Cascading Style Sheets (CSS) e `bootstrap`. A conversão de esboços de Graphical User Interfaces (GUIs) para código é uma tarefa normalmente realizada pelos programadores. Devido ao consumo de tempo nesta tarefa, torna-se impossível dedicar mais tempo à lógica da aplicação. Por outro lado, torna-se também uma tarefa repetitiva e tediosa.

A construção desta aplicação Web facilitará e agilizará o trabalho de programadores, visto que gera código automaticamente a partir de um esboço da interface da aplicação, elaborado com `Balsamiq Mockups`. O utilizador após receber o código, necessita apenas de adicionar código javaScript, substituir o texto pré-definido e personalizar o aspeto da página gerada.

Na presente dissertação contribui-se com o desenvolvimento de um conjunto de dados com esboços de páginas Web e as respetivas legendas. Devido à falta de conjuntos de dados com esboços Web suficientemente complexos, optou-se pela construção do próprio conjunto de dados.

Contribuiu-se também com a utilização do You Only Look Once (YOLO) na deteção e localização de elementos HTML e desenvolvimento de um algoritmo de *layout* que permite converter o resultado do YOLO em código. É uma abordagem completamente diferente do que se encontra nos trabalhos relacionados.

No capítulo dos resultados demonstra-se o funcionamento da abordagem. Para utilizar o modelo com um conjunto de dados diferente, é necessário legendar as imagens com a ferramenta LabelImg e fornecer os *templates* ao algoritmo de *layout*.

1.1 ENQUADRAMENTO

O planeta transformou-se num lugar digital, sendo atualmente impensável viver sem Internet. Nos últimos anos surgiram aplicações Web que nos ligam aos quatro cantos do mundo. Facebook e Gmail são exemplos de aplicações Web massivamente utilizadas.

Os modelos de aprendizagem automática[1] (ML) têm que ser integrados com algum tipo de aplicação para ficarem acessíveis a todo o tipo de utilizadores, independentemente do seu conhecimento ou experiência. Foi neste contexto, que surgiu a necessidade de criar uma aplicação Web para incorporar os modelos ML desenvolvidos.

A cooperação entre humanos e inteligência artificial é hoje uma realidade. Durante muitos anos e numa fase inicial, as máquinas apenas substituíam o ser humano em tarefas que envolvessem força. Com a utilização da força mecanizada perderam-se muitos postos de trabalho, mas ganharam-se outros mais ligados à capacidade cognitiva.

Atualmente, a inteligência artificial atingiu um desenvolvimento impressionante, que se deve sobretudo ao recente desenvolvimento computacional. Tarefas de enorme complexidade são hoje realizadas de forma mais rápida e eficiente por máquinas. Profissões que envolvam tarefas repetitivas, correm o risco de desaparecer e profissões relacionadas com a medicina podem ser totalmente remodeladas. A aprendizagem automática, com base num conjunto de métodos permite aos computadores aprender através da experiência, algo que já demonstrou ser extremamente eficaz. Os sistemas de deteção de imagem baseados em técnicas de inteligência artificial podem hoje reconhecer padrões com base em imagens, que lhe foram fornecidas para treino. Tudo indica que a inteligência artificial no futuro permita que os automóveis conduzam autonomamente. Os carros terão a capacidade de "aprender" a agir por conta própria. Integrando a informação proveniente dos sensores presentes no carro com sistemas de deteção e reconhecimento de objetos, o automóvel conseguirá interpretar cada situação e prever comportamentos na estrada.

A presente dissertação, enquadra-se no Mestrado Integrado em Engenharia Informática da Universidade do Minho e o objetivo inicial consistia em desenvolver uma aplicação Web que incorpora modelos de aprendizagem automática.

Por exemplo, neste último ano registou-se uma explosão de estudos na área da Internet of things (IoT). Internet das coisas é o conceito associado a objetos ligados entre si e à Internet. Estima-se que em 2020 se atinjam alguns biliões de dispositivos instalados e ligados à Internet, sendo que esse valor deve aumentar para triliões em 2025. Este aumento do número de dispositivos ligados à Internet originará um aumento do número de aplicações Web, e por isso, será extremamente útil aplicações que permitam converter esboços em código automaticamente sem grande esforço.

1 *Machine Learning* na terminologia Inglesa.

1.2 OBJETIVOS

A presente dissertação tem como principal objetivo desenvolver uma aplicação Web que permita converter esboços de páginas Web em código HTML e CSS, com a utilização de aprendizagem profunda.

Elencam-se a seguir os objetivos da dissertação:

- Aprender as boas práticas da construção de aplicações Web e saber lidar com modelos de aprendizagem profunda (criação, treino e avaliação/inferência);
- Perceber como funcionam as aplicações com `FlaskRestFul` (REpresentational State Transfer (REST) Application Programming Interface (API)) e a camada de apresentação em `ReactJs`;
- Desenvolver uma interface de utilizador esteticamente apelativa e fácil de usar;
- Construir uma aplicação Web que permita ao utilizador carregar um esboço de uma página Web e obter automaticamente o código HTML e CSS;
- Encontrar a rede neuronal profunda mais adequada para o problema e perceber como essa rede se integra com a aplicação Web;
- Criar ou adaptar, treinar e avaliar redes neuronais;
- Integrar as redes neuronais com uma aplicação Web;
- Desenvolver métricas que permitam avaliar e comparar os modelos desenvolvidos;
- Realizar testes e avaliar o desempenho da aplicação final.

1.3 ESTRUTURA DA TESE

A presente dissertação é constituída por 7 capítulos que dividem adequadamente o trabalho desenvolvido.

O capítulo 2 documenta a consolidação dos conceitos necessários à realização da dissertação. Apresenta um resumo da revisão bibliográfica, na qual se abordam detalhadamente os conceitos mais importantes da aprendizagem profunda. Nomeadamente as arquiteturas baseadas em CNNs, RNNs e Generative Adversarial Networkss (GANs). Dentro das redes neuronais, abordam-se as funções de custo, os otimizadores, os hiperparâmetros e as técnicas utilizadas no combate ao *overfitting*.

No capítulo 3 é efetuada uma análise das tecnologias utilizadas no desenvolvimento da aplicação Web, do conjunto de dados e dos modelos de aprendizagem profunda.

O capítulo 4 contém a análise do problema a tratar, incluindo o levantamento de requisitos funcionais e não-funcionais, e o desenvolvimento dos protótipos de baixa e de alta fidelidade da interface do utilizador.

No capítulo 5 são descritas todas as etapas do desenvolvimento do conjunto de dados, dos modelos de aprendizagem profunda (utilizando duas abordagens) e da aplicação Web. No caso da primeira abordagem, detalha-se o código Domain-specific language (DSL) criado e o compilador específico para este código. Na segunda abordagem, apresentam-se as evoluções do YOLO até à versão utilizada, o YOLO V3. Em ambas as abordagens são apresentadas as arquiteturas dos modelos e as métricas de avaliação desenvolvidas.

No capítulo 6 descrevem-se as experiências realizadas e os resultados obtidos. Para cada experiência, apresentam-se como resultados o tempo de treino e a precisão obtida.

Finalmente, o capítulo 7 contém as conclusões obtidas a partir dos resultados das experiências, bem como sugestões de trabalho a desenvolver no futuro.

2

ESTADO DA ARTE

Neste capítulo são apresentados alguns conceitos introdutórios sobre aprendizagem automática em inteligência artificial, mais especificamente sobre aprendizagem profunda. A revisão da literatura enquadra-se num contexto em que surgiram imensos métodos de aprendizagem profunda, os quais tiram partido de redes neuronais complexas aplicadas a extensos conjuntos de dados previamente classificados.

As secções 2.1 e 2.2 contêm uma introdução à inteligência artificial e à aprendizagem automática, respetivamente, enquanto que na secção 2.3 é apresentada a aprendizagem profunda. Na secção 2.4 são descritas as redes neuronais artificiais e os conceitos mais relevantes no desenvolvimento de modelos, como as funções de ativação e de custo, os otimizadores e o algoritmo de retropropagação. Nesta secção apresenta-se também as redes neuronais utilizadas em aprendizagem profunda, as CNNs, as RNNs e as GANs. Entre as secções 2.5 e 2.9 apresentam-se alguns conceitos importantes no desenvolvimento de modelos de aprendizagem automática como a engenharia de atributos, *overfitting* e *underfitting*, técnicas de avaliação cruzada, otimização de modelos e os passos a seguir na criação de um modelo. O capítulo termina na secção 2.10 com a apresentação dos trabalhos relacionados com a presente dissertação.

2.1 INTELIGÊNCIA ARTIFICIAL

A inteligência artificial é uma área da ciência da computação, que se dedica a descobrir métodos que permitam replicar a capacidade racional do ser humano. Subsequentemente, pretende-se pensar e racionalizar, o mais amplamente possível, com vista a resolver diversos problemas. A inteligência artificial refere-se a uma criação sintética semelhante à inteligência humana com capacidade de aprender, planear, perceber ou até processar linguagem natural (Russell and Norvig, 2009). O estudo da área iniciou-se após a Segunda Guerra Mundial com a publicação de Turing (1950) "Computing Machinery and Intelligence", o que demonstra o interesse na área desde os seus primórdios. Nas últimas décadas, realizaram-se imensos avanços, grande parte devido ao aumento exponencial da capacidade de processamento das máquinas, à Internet que permitiu gerar grandes quanti-

dades de dados (*big data*) e a evolução dos algoritmos e métricas utilizados na fase de treino. Atualmente, a inteligência artificial já faz parte do quotidiano. É recorrentemente utilizada no reconhecimento de imagem ou voz, na deteção de fraudes, na tradução automática ou na deteção de *spam* no correio eletrónico.

2.2 APRENDIZAGEM AUTOMÁTICA

A aprendizagem automática é uma subárea da inteligência artificial, que consiste em desenvolver *software* computacional com capacidade de instrução autónoma. Em 1959, Arthur Samuel definiu a aprendizagem automática como o "campo de estudo que dá aos computadores o poder e a capacidade de aprender sem serem programados explicitamente" (Simon, 2013).

Os algoritmos são sequências de passos utilizados para solucionar adversidades. Na programação clássica, o programador insere as regras e os dados de entrada para obter as respostas baseadas nas regras fornecidas. Na aprendizagem automática inserem-se apenas os dados de entrada e de saída, e os modelos dispõem de capacidade para aprender regras que permitem mapear as entradas nas saídas. Portanto, os modelos em vez de seguir instruções definidas pelo programador, criam as suas próprias instruções com base nos dados fornecidos.

Os algoritmos que utilizam este conceito, costumam ser classificados de acordo com o tipo de aprendizagem seguida: supervisionada, não supervisionada, semi supervisionada ou aprendizagem por reforço.

A aprendizagem **supervisionada** é a forma mais comum de aprendizagem automática. A aprendizagem é realizada com dados onde são conhecidos os resultados esperados. Visto isso, considera-se que existe um "supervisor", responsável por avaliar a resposta dada pela rede (Chapelle et al., 2010). Tanto a classificação como a regressão são exemplos deste tipo de aprendizagem. A **classificação** permite prever valores discretos. Consiste em atribuir rótulos aos dados de entrada, como um simples "verdadeiro" ou "falso". A **regressão** consiste na previsão de valores contínuos, como "Quanto custa?" ou "Quantos são?".

Na **aprendizagem não supervisionada** o conjunto de dados não tem qualquer informação sobre a saída, consequentemente, não existe um "supervisor". O modelo tem de encontrar padrões, relações ou categorias nos dados que lhe são fornecidos. Uma das formas mais populares de implementação desta aprendizagem consiste em criar agrupamentos (*clustering*). Cada agrupamento funciona como uma partição do conjunto de dados de treino, agrupados por semelhança entre si (Chapelle et al., 2010).

Russell and Norvig (2009) consideram a **aprendizagem semi supervisionada**[1] como um tipo de aprendizagem automática. Neste tipo de aprendizagem agregam-se dados com e sem informação sobre a saída esperada. Desta forma os dados fornecidos têm uma parte para treino supervisionado e outra para treino não supervisionado. É comum encontrar estes conjuntos de dados em contextos reais, pois não necessitam de especialistas para classificar ou avaliar os dados, tornando-se assim numa alternativa menos dispendiosa.

Em **aprendizagem por reforço** não se utilizam conjuntos de dados rotulados. A aprendizagem é realizada com base em ações que resultam num *feedback*. Os agentes agem sobre um ambiente dinâmico com a preocupação de maximizar a recompensa, sem conhecimento de qual a ação que permite obter maiores recompensas. Utilizam o método de tentativa e erro na pesquisa de soluções. O melhor resultado global é alcançado com base não só na recompensa atingida no momento, mas também na decisão que é mais benéfica para a decisão seguinte (Arulkumaran et al., 2017).

2.3 APRENDIZAGEM PROFUNDA

A aprendizagem profunda é um ramo da aprendizagem automática, que se baseia num conjunto de algoritmos que tentam modelar abstrações de dados. Utilizam grafos profundos compostos por várias camadas de processamento, que permitem obter representações de dados em múltiplos níveis de abstração (LeCun et al., 2015). A representação por camadas e, a consequente aprendizagem do modelo, ocorre maioritariamente com redes neuronais.

O termo redes neuronais tem origem na neurobiologia e inspira-se no cérebro humano. O treino destas redes é realizado através da aprendizagem de caraterísticas de forma hierárquica. As caraterísticas presentes nos níveis mais altos são constituídas pela combinação das mesmas nos níveis mais baixos, formando assim uma hierarquia. Pretende com estas caraterísticas simular o comportamento do cérebro humano, por exemplo na execução de tarefas de reconhecimento de imagem ou de voz e no processamento de língua natural.

Contrariamente à aprendizagem profunda outros tipos de abordagens tendem a focar-se apenas numa ou duas camadas, as quais se designam por aprendizagem superficial.

2.4 REDES NEURONAIS ARTIFICIAIS

Nos últimos anos emergiu a computação baseada em modelos que seguem a estrutura e o funcionamento das redes neuronais biológicas. Pode-se intitular este tipo de computação como paralela e distribuída. Estes avanços da tecnologia permitiram às máquinas realizar tarefas que requerem algum tipo de "inteligência".

1 *Self-supervised learning* na terminologia Inglesa.

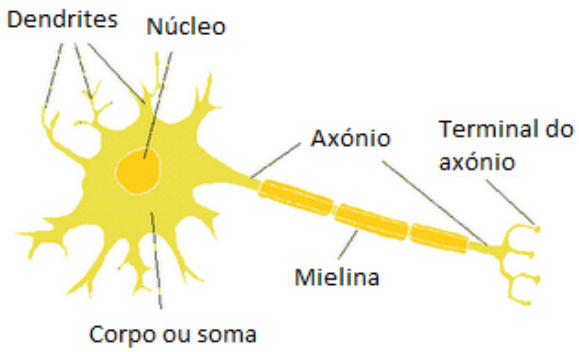


Figura 1.: Representação de um neurónio natural.

As redes neuronais artificiais são modelos simplificados do sistema nervoso central do ser humano. Trata-se de estruturas computacionais interligadas, designadas por nós ou neurónios, dotadas de capacidade de aprendizagem. O conhecimento é extraído a partir de um ambiente e armazenado nas ligações (sinapses) entre neurónios.

O cérebro humano apresenta uma estrutura fortemente ligada e com capacidade de aprender através da experiência. A rapidez do cérebro humano deve-se à sua estrutura altamente paralela, composta por cerca de 10 biliões de neurónios e 60 triliões de sinapses (Bajpai et al., 2011).

Os neurónios são constituídos pelo núcleo, corpo celular, dendrites, axónio e sinapses. Estas células são complexas e respondem a sinais eletroquímicos. As dendrites, que existem em grande quantidade, recebem os sinais provenientes de outros neurónios. O envio de sinais para outros neurónios é feito através do axónio. O transporte dos sinais entre o axónio de um neurónio e a dendrite de outro neurónio realiza-se através das sinapses. As sinapses funcionam como válvulas, que abrem e fecham, para permitir ou não a passagem da informação. Não existe contacto direto entre sinapses, devido à existência de fendas sinápticas onde atuam os neurotransmissores (substâncias libertadas pelo axónio) que fazem o transporte (Bajpai et al., 2011).

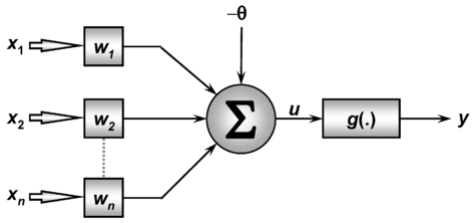


Figura 2.: Funcionamento de um neurónio numa rede neuronal artificial [2].

Da mesma forma que os neurónios recebem estímulos do ambiente e dos órgãos sensoriais, como os ouvidos ou os olhos, o neurónio artificial recebe os sinais de entrada sob a

forma de um vetor. Aos valores desse vetor, representados na figura 2 pela variável X, são aplicados pesos, representados pela variável W. Cada ligação tem associado um peso W que é multiplicado pelo valor de entrada X. Através de uma soma, um integrador transforma num único valor os vários valores recebidos dos vários nodos. Esta operação é efetuada no corpo celular de um neurónio natural. A saída do integrador passa depois através de uma função de ativação, onde é introduzida uma componente de não-linearidade (Noureldin et al., 2011).

O conhecimento de uma rede neuronal é armazenado nas ligações entre nodos, sob a forma de pesos. A aprendizagem ocorre durante a fase de treino, onde os pesos das ligações são ajustados com vista a atingir o melhor resultado possível.

A capacidade de aprendizagem é fundamental para tornar as redes neuronais diferentes da programação clássica, uma vez que a partir de conhecimento passado é possível generalizar e responder adequadamente a situações novas. Apresentam também alguma adaptabilidade pois, a topologia pode ser modificada conforme as alterações existentes no ambiente, mantendo-se em funcionamento.

2.4.1 *Arquiteturas das Redes Neuronais*

A arquitetura é uma caraterística importante que define como os neurónios estão organizados e interligados numa rede neuronal artificial. As redes neuronais artificiais têm uma ou várias camadas, sendo que a camada de entrada não é contabilizada. A topologia refere-se às diferentes composições estruturais que são possíveis com diferentes quantidades de neurónios por camada. As arquiteturas que se costuma diferenciar são apresentadas a seguir.

- **Rede neuronal *feed-forward* de camada única**: Redes simples que contém apenas a camada de entrada e a camada de saída. A camada de entrada não é verdadeiramente contabilizada como camada, uma vez que não realiza cálculos. Por este motivo, estas redes são denominadas de camada única. O fluxo de dados segue sempre na direção da camada de saída, daí a designação de redes *feed-forward* (Nielsen, 2018).

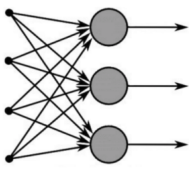


Figura 3.: Rede neuronal *feed-forward* de camada única.

2 https://pt.wikipedia.org/wiki/Perceptron_multicamadas

- **Rede neuronal *feed-forward* de múltiplas camadas**: Este tipo de rede carateriza-se por ter no mínimo uma ou mais camadas intermédias. Tal como nas redes *feed-forward* de camada única, o fluxo de dados fluí sempre na direção da camada de saída (Nielsen, 2018).

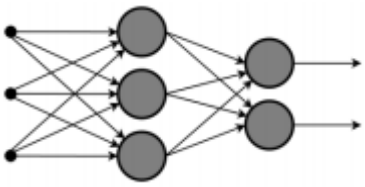


Figura 4.: Rede neuronal *feed-forward* de múltiplas camadas.

A rede da figura 4 possuí uma camada de entrada, uma camada intermédia e uma camada de saída. As camadas intermédias tratam da extração de informação e são caraterizadas pelos pesos que aplicam à informação proveniente dos neurónios da camada anterior. Os neurónios da camada de saída estão associados às classes de um problema de classificação. Num problema de classificação com apenas duas classes, a rede possui naturalmente dois neurónios de saída.

- **Rede neuronal recorrente**: Uma rede neuronal diz-se recorrente quando inclui neurónios em que a sua saída além de ser utilizada na camada seguinte também realimenta a própria camada, o que origina a criação de circuitos fechados, que podemos chamar ciclos. A presença de ciclos impede que as saídas dependam exclusivamente da informação proveniente da camada anterior. Existe assim uma combinação entre informação da camada anterior e informação da própria camada (estado). É devido ao estado que a rede neuronal recorrente tem a capacidade de guardar e lidar com informação passada. Na figura 5 é possível ver a realimentação dos neurónios a formarem ciclos, caraterística que distingue as redes recorrentes das redes *feed-forward* (Nielsen, 2018).

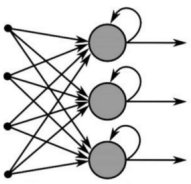


Figura 5.: Rede neuronal recorrente.

Por exemplo, as aplicações em que os dados de entrada são sequências temporais necessitam, obrigatoriamente, de redes que utilizam, além da entrada atual, informação passada para gerar a próxima saída.

2.4.2 *Algoritmo de retropropagação*

Em 1958, Rosenblatt (1958) criou o `percetrão`, o tipo de rede neuronal *feed-forward* mais simples, um classificador linear binário que permite mapear as entradas num valor binário. Posteriormente, em 1960 Widrow (1960) especificou um neurónio artificial baseado no modelo de McCulloch e Pitts, denominado Adaptive Linear Neuron (ADALINE). A combinação de múltiplos ADALINEs designa-se Many Adaptive Linear Neuron (MADALINE).

A diferença entre a ADALINE e o `percetrão` está na fase de aprendizagem. Na ADALINE, os pesos são ajustados de acordo com a soma dos pesos de entrada. Enquanto que no `percetrão`, os valores de entrada são passados individualmente e diretamente para a função de ativação.

Durante a década de 1960 surgiram várias publicações que demonstraram as várias limitações destas primeiras redes neuronais, o que provocou um desinteresse geral nas redes neuronais nos anos seguintes. O interesse ressurgiu novamente nos anos 80, com destaque para as redes recorrentes descobertas por Hopfield em 1982.

Em 1986 surgiu a descrição do algoritmo de retropropagação. O algoritmo baseia-se na retropropagação do erro desde a saída até à entrada, o que permite ajustar os pesos das várias camadas. Este algoritmo foi aplicado na resolução de vários problemas, como previsão de séries, reconhecimento de voz ou tradução automática. O sucesso desta descoberta estimulou exponencialmente a pesquisa na área.

Atualmente, procuram-se redes e algoritmos de treino mais eficientes, de modo a que a sua aplicação se estenda cada vez mais à economia, visão por computador e à robótica.

As redes neuronais artificias treinadas com o algoritmo de **retropropagação** (*backpropagation*) apresentam melhores resultados do que os obtidos com o *percetrão* ou a ADALINE. Este algoritmo pretende simular a aprendizagem com erros e resulta do trabalho desenvolvido por Rumelhart et al. (1986). Seria impossível treinar as redes neuronais profundas que se utilizam atualmente, sem a eficiência deste algoritmo. Recorrendo a um dos métodos de gradiente descendente, o algoritmo de retropropagação é fundamental ao treino de redes neuronais artificiais supervisionadas. Dada uma função de erro, calcula-se o gradiente dessa função em relação a cada um dos pesos aplicados na camada de saída. O processo repete-se para cada uma das camadas da rede até à entrada.

Durante o treino de uma rede, cada iteração decorre em duas etapas. Na primeira etapa ocorre a propagação das entradas para gerar as saídas. Na segunda, decorre a retropropagação do erro da saída para a entrada.

Inicialmente os pesos da rede podem ser inicializados com valores pequenos aleatórios. Posteriormente, na fase de propagação, a entrada é processada camada a camada, onde se aplicam pesos e funções de ativação. A saída de uma camada funciona como entrada da camada seguinte. Para treinar a rede utiliza-se uma função de custo que mede a distância

entre a saída gerada e o valor esperado. O treino tem como propósito encontrar os pesos que minimizem a função de custo.

A fase de retropropagação começa na camada de saída e evolui no sentido contrário ao da propagação. O erro calculado na saída da rede é propagado em direção à entrada, camada a camada. Em cada camada atravessada, os pesos são ajustados de acordo com uma estratégia definida pelo otimizador selecionado.

O treino com o algoritmo de retropropagação é supervisionado e, portanto, quando é apresentada uma entrada também é disponibilizada a respetiva saída. Desta forma, calculado o erro global na saída é possível derivar o contributo para esse erro por parte de cada um dos pesos da rede. Na posse destes contributos, o otimizador corrige os pesos. Estes passos são executados repetidamente até o erro global ser inferior ao limiar pré-estabelecido ou até se atingir o número máximo de iterações.

A equação 1 mostra uma possível fórmula de atualização dos pesos w da rede neuronal, onde E representa o erro e η é a taxa de aprendizagem. Esta é a fórmula utilizada pelo otimizador do gradiente descendente. Para selecionar o valor da taxa de aprendizagem convém estabelecer um compromisso entre tempo de cálculo e oscilação da trajetória seguida pela otimização. Valores de η muito pequenos tornam o treino muito lento. Por outro lado, valores de η muito grandes provocam grandes oscilações na trajetória, o que pode dificultar a procura do ponto ótimo para os pesos da rede.

$$w = w - \eta \frac{\partial E}{\partial w} \tag{1}$$

O algoritmo de retropropagação é hoje utilizado no treino de redes profundas, tanto nas redes neuronais convolucionais (comummente utilizadas na classificação de imagens) como nas redes neuronais recorrentes (muitas vezes usadas no processamento de língua natural).

2.4.3 *Funções de Ativação*

As funções de ativação são importantes para que pequenas alterações nos pesos provoquem pequenas alterações na saída final. Uma rede neuronal sem função de ativação, seria linear e funcionaria apenas como um modelo de regressão linear. É a não-linearidade, aplicada aos dados de entrada, que permite aprender a executar tarefas mais complexas.

A função de ativação, representada por g na figura 2, recebe como entrada o valor do integrador e altera a saída do neurónio de acordo com a função matemática selecionada (Nwankpa et al., 2018). De entre as funções de ativação, a linear é uma das mais comuns.

A função **linear** produz uma saída que não está restringida a um intervalo (Nwankpa et al., 2018). Este tipo de função não se adequa a dados de grande complexidade ou quantidade.

$$f(x) = x \tag{2}$$

As funções lineares eram utilizadas nos primeiros algoritmos de redes neuronais artificiais, como por exemplo a ADALINE ou o percetrão.

As funções não-lineares, são aquelas que se utilizam nas redes neuronais artificiais de múltiplas camadas atuais (Nwankpa et al., 2018). Facilitam a generalização ou adaptação a grande quantidade e diversidade de dados.

A função **logística**, ou ***sigmoide*** é das funções mais utilizadas nas redes neuronais artificiais. É um função não-linear suave, parecida com um S que apresenta um bom compromisso entre linearidade e não-linearidade (Nwankpa et al., 2018). A sua saída situa-se no intervalo [0, 1].

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

É especialmente utilizada quando é necessário prever a probabilidade de a entrada pertencer a uma de duas classes, ou seja, quando se está perante uma classificação binária. A probabilidade a gerar pelo classificador e o valor de saída desta função variam no mesmo intervalo ([0, 1]), o que facilita a sua utilização nestes casos (Nwankpa et al., 2018).

A função **Softmax** é uma generalização da função sigmoide, o que a torna adequada para modelos de classificação em múltiplas classes. A soma das probabilidades de todas as classes deve ser igual a 1. Quando a probabilidade de uma classe aumenta, obriga a que a probabilidade de outra(s) classe(s) diminua. A classe de saída será aquela que possuir a maior probabilidade (Nwankpa et al., 2018).

$$f(x) = \frac{e^{i}}{\sum_{j=0}^{k} e^{j}} \tag{4}$$

A **Tangente hiperbólica** é uma função semelhante à sigmoide, apresentando também a forma de um S. O intervalo de saída varia entre [-1, 1]. A sua utilização adequa-se mais à classificação binária (Nwankpa et al., 2018).

$$f(x) = tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{5}$$

Na prática é muitas vezes preferida, em detrimento da sigmoide, embora ainda sofra do problema de *vanishing gradients*.

A **ReLU**[3] é a função de ativação mais utilizada presentemente, principalmente em redes neuronais de aprendizagem profunda, como as convolucionais. A metade negativa desta função é representada por uma reta de inclinação nula e a outra metade por uma reta com inclinação de 45 graus. Para entradas negativas a função gera uma saída nula (Nwankpa et al., 2018).

$$f(x) = max(0, x) \tag{6}$$

Com uma derivada maior que a sigmoide e que a tangente hiperbólica, quando a entrada é positiva, a ReLu reduz o problema de *vanishing gradients*. No entanto, tem limitações. Quando os valores são inferiores a 0, os pesos não serão ajustados. Nesses casos os neurónios param de responder a variações. Este problema é resolvida na variante Leaky ReLu.

A **Leaky ReLU** corrige o problema da função ReLu. Para valores de entrada negativos, em vez de uma saída com inclinação nula e valor zero, a função é representada por uma reta com inclinação α reduzida. Desta forma, evita-se que os neurónios parem de responder como acontecia na função ReLu (Nwankpa et al., 2018).

$$f(x) = \begin{cases} \alpha x & x \leq 0 \\ x & x > 0 \end{cases} \tag{7}$$

2.4.4 *Função de Custo*

As funções de custo são fundamentais ao treino de redes neuronais, pois servem de indicador para os otimizadores. Estas funções avaliam como o modelo representa o conjunto de dados. Permitem calcular a diferença entre a saída atual e a saída prevista. Diferentes funções apresentam diferentes valores para a mesma previsão, o que leva a diferentes aprendizagens nas redes neuronais (Srivastava et al., 2019).

Problemas de classificação (valores discretos) e regressão (valores contínuos) utilizam funções de custo diferentes. As funções apresentadas a seguir permitem calcular o erro em problemas envolvendo variáveis contínuas, ou seja, em problemas de regressão.

3 Rectified Linear Unit.

Em problemas de regressão, o **erro quadrático médio (MSE)**[4] é a função de cálculo do erro mais utilizada. A função contabiliza a média do quadrado do erro, pelo que dá mais importância a desvios elevados, quando comparada com o erro absoluto médio.

$$MSE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|^2 \tag{8}$$

A função **Mean Absolute Error (MAE)** calcula a média dos erros em valor absoluto (Xu et al., 2017).

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j| \tag{9}$$

É mais fácil treinar modelos com a função Mean Squared Error (MSE) porque o seu gradiente não é constante para todos os valores do erro. No entanto, quando existem *outliers* nos dados de entrada, a função Mean Absolute Error (MAE) é mais robusta porque não atribui um peso exagerado (o quadrado) ao erro associado aos valores que estão muito afastados do valor esperado.

As seguintes funções são comummente utilizadas para calcular o custo em problemas de classificação, ou seja, quando se utilizam variáveis discretas.

Cross entropy é a medida de erro mais comum em problemas de classificação, principalmente em classificação binária. Mede a distância entre a distribuição que o modelo acredita ser a melhor e a distribuição original (Nielsen, 2018). Na classificação binária é dada pela seguinte equação:

$$-(y.log(\hat{y}) + (1-y).log(1-\hat{y})) \tag{10}$$

onde y é a classe associada à entrada atual (um valor binário) e $\hat{y}$ é a probabilidade estimada pelo modelo, para a entrada atual.

Com **Multi-class cross entropy**, variante do custo que permite utilizar *cross entropy* em problemas de classificação em múltiplas classes (Li et al., 2019).

$$-\sum_{c=1}^{N} y_{o,c}\, log(\hat{y}_{o,c}) \tag{11}$$

4 Mean Squared Error (MSE).

Na equação anterior, a variável $y_{o,c}$ é um valor binário que indica se a observação o pertence à classe c ou não. $\hat{y}_{o,c}$ é a probabilidade estimada pelo modelo de a observação o pertencer à classe c e N é o número de classes.

2.4.5 *Problema de vanishing gradients*

Os gradientes que tendem para zero é um problema que afeta o treino das redes neuronais, especialmente nas redes profundas e quando se utilizam métodos de otimização baseados em gradientes. Um exemplo destes métodos é a retropropagação[5]. A origem do problema está na atualização dos pesos com base na derivada parcial da função de custo. Se a rede incluir muitas camadas, um gradiente pequeno na saída facilmente se esbate ao retropropagar-se através das muitas camadas e não consegue atualizar os pesos das camadas mais próximas da entrada da rede (Nielsen, 2018).

No pior caso nenhum peso consegue ser atualizado e a aprendizagem é bloqueada. Este problema pode ser minimizado com a utilização da função de ativação ReLu, em vez da sigmoide ou da tanh.

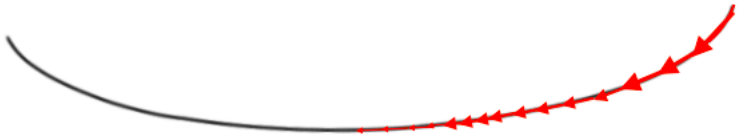


Figura 6.: Função de custo que ilustra o problema de *vanishing gradients* [6].

2.4.6 *Otimizadores*

O treino das redes neuronais é um processo iterativo onde se pretende minimizar o valor da função de custo. Durante este processo, procura-se o valor ótimo para os pesos da rede neuronal, de modo a que se minimize a função de custo. As técnicas de pesquisa baseadas em gradientes, como é o caso do algoritmo de retropropagação, são as mais utilizadas para otimizar redes neuronais.

O algoritmo de otimização calcula os gradientes, isto é, a derivada parcial da função de custo em relação aos diversos pesos da rede neuronal. Cada peso é depois alterado na direção oposta ao valor do respetivo gradiente. Este processo repete-se até se alcançar o mínimo da função de custo, ou até se atingir outro tipo de critério de paragem (Li, 2017).

5 *Backpropagation* na terminologia Inglesa
6 https://miro.medium.com/max/1220/1*OXjoABpGS1Oypaqgiuwnug.png

Os algoritmos de otimização apresentados a seguir são aplicados em cada iteração e para todos os pesos da rede neuronal em simultâneo. Os pesos da rede formam um vetor a partir do qual será calculado o gradiente.

2.4.6.1 *Gradiente estocástico descendente (SGD)*

O método clássico do gradiente descendente é muito dispendioso, pois processa todo o conjunto de dados em cada iteração. Na variante SGD, em cada iteração seleciona-se apenas um subconjunto de amostras para calcular os gradientes e atualizar o valor dos pesos. SGD é dos mais utilizados no treino de redes neuronais (Li, 2017).

$$\theta = \theta - \eta(\nabla L(\theta)) \tag{12}$$

Na equação 12, a variável θ é um parâmetro (ou peso) do modelo, η é a taxa de aprendizagem e $\nabla L(\theta)$ é o gradiente da função de custo.

Entre as desvantagens do método SGD inclui-se a necessidade de escolher o valor da taxa de aprendizagem manualmente, não existindo uma forma de obter o seu valor automaticamente. O algoritmo tem ainda dificuldade em fugir de pontos em que a derivada da função de custo é nula, sem que esses pontos sejam mínimos absolutos.

2.4.6.2 *Adagrad*

A maioria dos métodos de gradiente descendente utiliza a mesma taxa de aprendizagem durante todo o treino. O método de otimização Adagrad aplica, a cada parâmetro do modelo, uma taxa de aprendizagem diferenciada. A variação da taxa de aprendizagem resulta de se efetuar uma acumulação de gradientes ao longo das iterações, materializada no termo G_t da equação 13. Deste modo, elimina-se a necessidade de selecionar a taxa de aprendizagem manualmente (Li, 2017).

A taxa de aprendizagem controla o tamanho do ajuste dos pesos aplicado em cada iteração (*epoch*) do treino da rede neuronal. Em cada iteração, cada parâmetro do modelo é atualizado de acordo com a equação 13.

$$\theta = \theta - \frac{\eta}{\sqrt{G_t + \epsilon}}.\nabla L(\theta) \tag{13}$$

onde G_t representa a soma do quadrado dos gradientes anteriormente calculados, ϵ é um termo da ordem de $1e^{-8}$ para evitar a divisão por zero, e as restantes variáveis são as mesmas que se mencionaram no método SGD.

Por vezes, valores muito elevados no denominador provocam uma taxa de aprendizagem muito baixa, o que leva a que o treino seja interrompido.

2.4.6.3 *Adadelta*

O algoritmo Adadelta foi desenvolvido em simultâneo com o algoritmo RMSProp. Com o Adadelta pretende-se corrigir o problema do Adagrad, ou seja, evitar a acumulação de um valor sucessivamente maior no termo G_t incluído no denominador da equação 13.

O problema da acumulação de um valor sucessivamente maior, no denominador do termo que multiplica os gradientes, é resolvido efetuando a acumulação do quadrado dos gradientes em janelas móveis. Ao invés de acumular todos os gradientes desde o início do treino, armazena-se apenas parte desses gradientes, o que impede a acumulação de valores ilimitados (Ruder, 2016).

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \tag{14}$$

Para combater a ineficiência do algoritmo, guarda-se a média dos gradientes para ela ser reutilizada na iteração seguinte. A média $E[g^2]_t$ é calculada com a equação 14, γ permite atribuir diferentes pesos à média dos gradientes anteriores $E[g^2]_{t-1}$ e os pesos são ajustados de acordo com a equação 15.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} . \nabla L(\theta) \tag{15}$$

2.4.6.4 *RMSProp*

O algoritmo RMSProp foi criado por Geoffrey Hinton, mas infelizmente nunca foi publicado um artigo sobre ele. O principal objetivo era colmatar as falhas do algoritmo Adagrad. Para combater a acumulação de gradientes do algoritmo Adagrad, mantém-se uma média móvel com o quadrado dos gradientes anteriores para cada peso. Quase sempre, os métodos de gradiente que utilizam momento convergem mais rapidamente que o método *standard*, como acontece com este algoritmo (Ruder, 2016). O momento é responsável por limitar a atualização dos pesos e, deste modo, evitar alterações bruscas no valor dos pesos.

Enquanto que no Adadelta o peso atribuído aos gradientes anteriores e ao gradiente atual podem variar de acordo com o valor de γ, neste algoritmo os pesos são fixos (equação 16).

$$E[g^2]_t = 0.9 E[g^2]_{t-1} + 0.1 g_t^2 \tag{16}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2] + \epsilon}} . \nabla L(\theta) \tag{17}$$

2.4.6.5 *Adam*

O algoritmo Adam utiliza momentos de primeira e segunda ordem do gradiente. A ideia base por trás deste algoritmo, consiste na redução da velocidade de procura, de modo a que a pesquisa seja mais cautelosa e não se salte o valor mínimo da função de custo. Permite bons resultados quando aplicado a redes neuronais convolucionais, traduzidos numa convergência mais rápida e numa classificação com maior precisão. O Adam aprende rapidamente e é mais estável do que os otimizadores anteriores, uma vez que não apresenta reduções abruptas de precisão (Hayashi et al., 2018).

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}.\hat{m}_t \tag{18}$$

onde $\hat{m}_t$ é a média e $\hat{v}_t$ a variância dos gradientes.

2.4.7 *Métricas*

As métricas são utilizadas para avaliar o desempenho dos modelos. Apesar de a função de custo utilizada pelo otimizador também ser usada para melhorar os modelos, e embora possa estar correlacionada com as métricas de avaliação, custo e métricas de avaliação são coisas distintas.

As métricas variam com o tipo de problema que estamos a resolver. Em problemas de classificação as métricas mais populares são a acurácia, a precisão e o *recall* (ou *sensitivity*). Considera-se que ocorre um falso positivo ou um falso negativo, quando a resposta obtida não é a resposta esperada. Nos casos verdadeiros positivos e verdadeiros negativos, a resposta obtida é igual à resposta esperada.

A **Acurácia** é o número de previsões corretas a dividir pelo número total de previsões realizadas. Esta métrica apresenta bons resultados quando as classes da variável alvo são balanceadas (Danjuma, 2015). Quando existe uma classe dominante em relação às restantes, não se deve utilizar a acurácia.

$$\text{Acurácia} = \frac{\text{Número de previsões corretas}}{\text{Número total de previsões}} \tag{19}$$

A **Precisão** é semelhante à acurácia e mede a proporção de previsões certas numa classe. Supondo que se está a prever quantas pessoas possuem uma determinada doença, a precisão baseia-se nas previsões afirmativas e compara com o número de pessoas que real-

mente possuem a doença (Danjuma, 2015). Esta métrica é mais indicada quando se pretende minimizar os falsos positivos.

$$\text{Precisão} = \frac{\text{Verdadeiros positivos}}{\text{Verdadeiros positivos + Falsos positivos}} \tag{20}$$

Recall* ou *Sensitivity é a medida que indica a proporção de elementos pertencentes à classe, independentemente do algoritmo indicar que pertence ou não à classe (Danjuma, 2015). Por exemplo, no caso de prever se alguém tem diabetes, esta medida dá a proporção de pessoas que possuem a doença, embora o algoritmo possa dizer o contrário. Quando se pretende minimizar os falsos negativos, *recall* é a métrica mais indicada.

$$\text{Recall} = \frac{\text{Verdadeiros positivos}}{\text{Verdadeiros positivos + Falsos negativos}} \tag{21}$$

Em problemas de **regressão** as variáveis são contínuas, e como tal utilizam-se métricas diferentes. Entre as métricas mais populares estão o Root mean squared error (RMSE) e o MAE.

O **RMSE** é frequentemente utilizado em cálculos que envolvem valores estimados e valores reais, como em modelos de previsão. O valor da métrica é dado pela raiz quadrada do erro quadrático médio (Xu et al., 2017).

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2} \tag{22}$$

MAE é a média dos erros absolutos, entre valores estimados e valores reais. Esta métrica tem a particularidade de pesar igualmente todas as diferenças entre os valores reais e estimados, o que não acontece no RMSE.

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j| \tag{23}$$

A métrica RMSE é mais afetada negativamente pelos *outliers* do que o MAE, visto que considera o quadrado do erro (Xu et al., 2017).

2.4.8 *Hiperparâmetros*

Os hiperparâmetros são variáveis (ou parâmetros) que caraterizam a estrutura e a forma como um modelo é treinado (Aszemi and Dominic, 2019). Entre as variáveis que determinam a estrutura da rede, pode estar o número de neurónios por camada e o número de camadas. As variáveis de treino referem-se por exemplo, à taxa de aprendizagem ou ao número de *epochs*. Os hiperparâmetros são definidos antes do treino.

2.4.8.1 *Hiperparâmetros de Estrutura*

- **Número de camadas e neurónios por camada**: As camadas intermédias são aquelas que se encontram entre a camada de entrada e a camada de saída. Deve-se iniciar a construção da rede com poucas camadas. O número de camadas deve aumentar até surgir *overfitting*. O mesmo deve acontecer com o número de neurónios, tendo em atenção que poucos neurónios por camada pode provocar *underfitting*.

- ***Dropout***: Técnica de regularização que permite evitar o *overfitting*. A percentagem de neurónios a desativar por iteração varia normalmente no intervalo 20-50%. O *dropout* é mais efetivo em redes maiores.

- **Função de ativação**: As funções de ativação são utilizadas para introduzir não linearidade nos modelos. A função sigmoide (mais adequada a classificação binária) e softmax (classificação em múltiplas classes) são das funções mais utilizadas.

2.4.8.2 *Hiperparâmetros de Treino*

- **Taxa de aprendizagem**: A taxa de aprendizagem é considerada o hiperparâmetro mais importante. Controla o ajuste dos pesos na rede de acordo com a função de custo. Um valor muito baixo significa que a rede vai demorar muito tempo a convergir, o que não é mau no sentido em que não se perderá nenhum mínimo local, no entanto, pode tornar o problema muito dispendioso e demorado. Além disso, o treino pode ficar preso num mínimo local. Com uma taxa de aprendizagem muito elevada, a rede pode não convergir devido às grandes oscilações de trajetória.

- **Número de *epochs***: É o número de iterações que ocorre sobre todo o conjunto de dados. Com base no valor de custo obtido em cada *batch*, a rede calcula o gradiente dos pesos e procede à atualização dos mesmos. Uma rede que seja definida com *batch_size* de 64 e 10 *epochs* tem 640 atualizações dos gradientes. O número de *epochs* deve aumentar até surgir *overfitting*.

- **Batch size**: Os modelos de aprendizagem profunda não processam todo o conjunto de dados de uma só vez. Dividem o conjunto de dados em grupos mais pequenos

(*batches*). Um bom valor por omissão para o tamanho das *batches* é 32. No entanto, também se deve ponderar a utilização do valor 64, 128, 256 ou mesmo 512, uma vez que, dependendo da quantidade de dados existente, pode ser necessário utilizar valores superiores para o treino ter duração exequível.

2.4.9 *Redes Neuronais Convolucionais*

Em aprendizagem profunda, o tipo de redes neuronais mais comum são as convolucionais (CNNs ou *convnets*) e uma das áreas de aplicação emblemáticas é a visão por computador. Nas redes convolucionais as representações aprendidas podem ser consultadas, uma vez que representam conceitos visuais. Portanto, ao contrário de outros tipos de arquiteturas, as CNNs não se assemelham a caixas negras, sendo possível analisar o seu funcionamento de forma visual (Liu et al., 2015).

Uma CNN divide-se em duas partes. A primeira é composta por várias camadas convolucionais e a segunda inclui algumas camadas densamente conectadas (figura 7). As camadas densamente conectadas, por se localizarem no topo da rede e por terem um campo de atuação alargado, têm a capacidade de aprender caraterísticas mais específicas da imagem de entrada, não se limitando a aprender caraterísticas superficiais e que são comuns a muitas imagens.

Segundo Liu et al. (2015), um aspeto chave para compreender o funcionamento das CNNs é perceber que elas processam o reconhecimento de caraterísticas espaciais de forma hierárquica. As primeiras camadas reconhecem padrões simples, como arestas, vértices ou outros padrões abstratos menos intuitivos. À medida que se avança em profundidade na rede as caraterísticas tornam-se menos abstratas, maiores e mais complexas, ou seja, passam a ser elementos que resultam da acumulação de caraterísticas identificadas nas camadas anteriores. Por exemplo, se aplicarmos uma imagem com um gato na entrada de uma CNN, inicialmente serão detetadas curvas, linhas ou arestas, depois orelhas, patas ou nariz, e posteriormente o corpo de gato por inteiro.

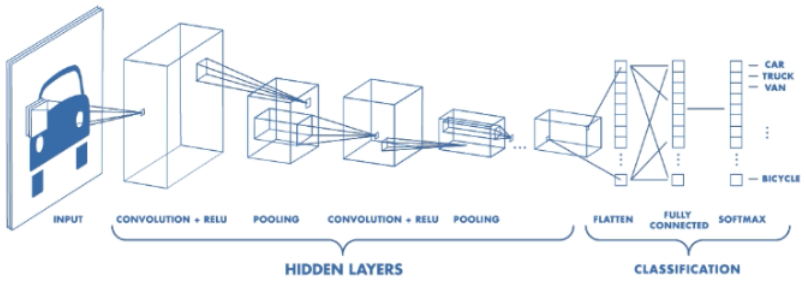


Figura 7.: Arquitetura de uma CNN [7].

7 https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050

Este tipo de rede é diferente das restantes, pois as camadas de extração de caraterísticas estão organizadas em três dimensões: largura, altura e profundidade. Além disso, os neurónios de uma camada não estão ligados a todos os neurónios da camada anterior. Pelo contrário, cada neurónio só recebe entradas provenientes de um conjunto limitado de neurónios, localizados numa janela de tamanho fixo, e aos quais aplica um conjunto de pesos que é partilhado por todos os neurónios da mesma camada.

É comum aplicarem-se operações de amostragem (*pooling*) neste tipo de rede neuronal. A amostragem consiste em reduzir o mapa de caraterísticas, normalmente aplica-se uma redução para metade, pelo que é considerada uma redução agressiva. A não utilização de amostragem implica a utilização de milhões de parâmetros, o que torna as operações muito mais dispendiosas. Existem algumas alternativas de amostragem, como é o caso da média e do máximo, mas o máximo é o mais utilizado porque tende a funcionar melhor.

A segunda parte das CNNs envolve a utilização de camadas densamente ligadas, onde cada neurónio de uma camada recebe entradas provenientes de um conjunto extenso de neurónios da camada anterior. Estas camadas são necessárias para classificar as características extraídas (Chollet, 2017).

Como já foi referido, a extração de caraterísticas é realizada através de convoluções. Em termos matemáticos fala-se da combinação de duas funções que produz uma terceira função. A convolução também pode ser vista como a aplicação de um filtro ou *kernel*. Os filtros são progressivamente aplicados a toda a imagem, capturando as partes marcantes (figura 8). A profundidade da saída de uma convolução é dada pela quantidade de filtros aplicados. Quanto maior for a profundidade, maior será a quantidade e a qualidade das caraterísticas encontradas.

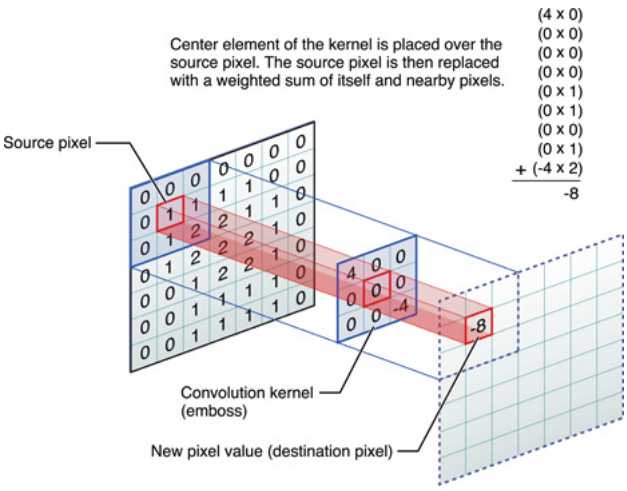


Figura 8.: Operação de convolução [8].

8 https://cdn-images-1.medium.com/max/800/1*EuSjHyyDRPAQUdKCKLTgIQ.png

As redes neuronais convolucionais também podem ser utilizadas para detetar padrões em séries temporais, embora sejam mais adequadas e conhecidas pela sua utilização na área da visão por computador.

Em todos os problemas de aprendizagem automática, o *overfitting* é um grande inimigo, especialmente quando o tamanho do conjunto de dados é insuficiente. Para combater este problema pode utilizar-se a técnica de **aumento de dados** (Chollet, 2017). Esta abordagem, consiste em aumentar a quantidade de dados através da geração de novos dados com base nas amostras existentes. No entanto, durante a fase de treino o modelo não deve ver a mesma imagem duas vezes, mas sim variações da mesma imagem para introduzir nova informação. Por exemplo, as variações podem ser construídas através de rotações, ampliações, reduções ou deslocações em largura ou altura, sendo que essas imagens devem ter um aspeto real, como é ilustrado na figura 9.

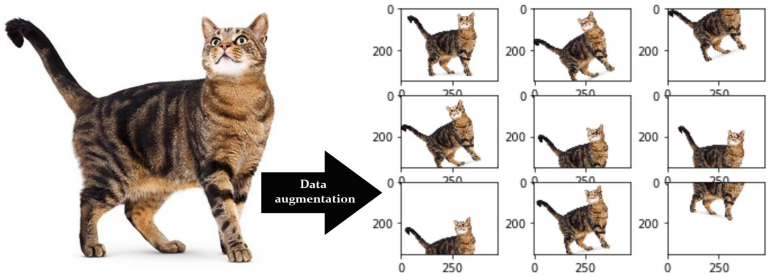


Figura 9.: Criar novas imagens com a técnica de aumento de dados [9].

Outra forma de obter melhores resultados, consiste em utilizar redes pré-treinadas (***transfer learning***). Estas redes foram treinadas anteriormente com grandes conjuntos de dados. Existem duas formas de utilizar modelos pré-treinados, através da extração de caraterísticas ou através de *fine-tuning*.

Com ***transfer learning*** os modelos treinam mais rapidamente e alcançam maior precisão. Em vez de iniciar o treino do zero, utiliza-se os padrões aprendidos na resolução de outro problema. Esta técnica expressa-se através da utilização de modelos pré-treinados em conjuntos de dados diferentes. É muito comum ser utilizada com CNNs (VGG, ResNet, inception, etc). Uma CNN típica possuí duas partes, as camadas convolucionais (extração de caraterísticas) e o classificador. Antes de compilar e treinar o modelo é necessário congelar as camadas de extração de caraterísticas, isto é, fixar as camadas de extração de modo a impedir que os seus pesos sejam atualizados durante o treino. Desta forma, o modelo mantém a base da extração de caraterísticas resultante do conjunto de dados anterior. O classificador não é fixado e, portanto, vai adaptar-se ao novo problema a resolver (Chollet, 2017).

9 https://i.stack.imgur.com/UKwFg.jpg

Outra técnica que permite utilizar redes pré-treinadas é ***fine-tuning***, uma técnica complementar à extração de caraterísticas. Consiste em descongelar camadas de extração de caraterísticas, o que não acontece na técnica anterior. Enquanto que na extração de caraterísticas apenas se ajustam os pesos do classificador, com *fine-tuning* adapta-se parte ou totalidade das camadas de extração, o que contribuí para a aprendizagem de caraterísticas específicas do novo conjunto de dados.

2.4.10 *Redes Neuronais Recorrentes*

Posteriormente à decada de 80, os estudos com redes neuronais aumentaram drasticamente. Em 1982, John Hopfield, criou um tipo de rede diferente daquelas que existiam até à data (Sulehria and Zhang, 2007). Neste modelo a rede apresentava ligações recorrentes, isto é, o sinal não se propagava apenas para a frente, também realimentava a própria camada. John Hopfield utilizou uma função de energia para construir as redes recorrentes, embora bastante simples, pois a rede só possuía uma camada.

A memória utilizada era associativa, uma vez que utilizava um vetor com tamanho igual ao número de neurónios. O estado dos neurónios podia assumir um de dois valores: '+1' quando o neurónio estava ativo e '-1' quando não estava ativo. Em alguma bibliografia encontra-se o estado dos neurónios representado em binário (0 ou 1). A atualização do estado resultava da soma das entradas de um neurónio, que através de uma função signum colocava o neurónio ativado ou desativado. Cada par de neurónios, tem uma ligação que é descrita por um peso. A ligação pode ser descrita como um grafo não direcionado completo sem ciclos, ou seja, a ligação de um neurónio com ele próprio tem peso nulo. Portanto, nenhum neurónio tem ligação com ele próprio, mas aos restantes neurónios da mesma camada (Chollet, 2017).

Estas foram as primeiras redes recorrentes e desencadearam diversos estudos que levaram à descoberta de novas redes. Contudo, nem elas per si, nem a descoberta de novos algoritmos, foram suficientes para as tornar populares. Foram os avanços dos sistemas computacionais que permitiram a sua utilização em larga escala.

As redes neuronais recorrentes possuem memória, o que não acontece nas redes neuronais *feed-forward*. Como não existem apenas ligações às camadas seguintes, é possível armazenar informação para influenciar o processamento das próximas entradas. Desta forma torna-se possível resolver problemas que as redes neuronais *feed-forward* eram incapazes de resolver, devido à falta de estado, o que leva ao processamento das entradas de forma independente. Ao guardar informação, as redes recorrentes conseguem manter a sequência dos acontecimentos, como acontece nas previsões meteorológicas, nas séries temporais ou no processamento de texto. Em cada ciclo das redes neuronais recorrentes, não há necessidade de utilizar o resultado de todas as iterações anteriores, uma vez que em cada

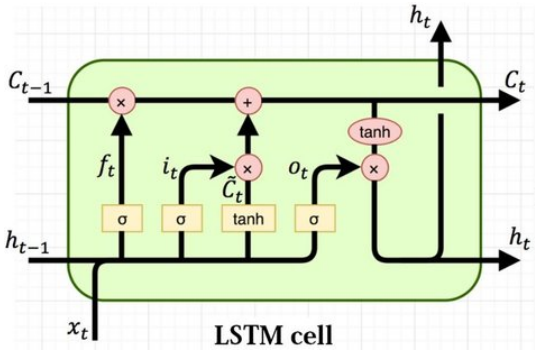
iteração pode acumular informação desde a iteração inicial até à iteração anterior. Assim, é suficiente utilizar a informação da iteração anterior como estado. Obviamente, alguma dessa informação será alterada e consequentemente, perdida ao longo das várias iterações.

O problema das dependências de longo prazo, ou seja, a informação que se perde ao longo do tempo, afeta consideravelmente as RNNs simples. Nas subseções seguintes serão apresentados alguns tipos de redes neuronais recorrentes que permitem resolver este problema, como é o caso das redes neuronais Long Short-Term Memory (LSTM) ou Gated Recurrent Units (GRU), que foram concebidas propositadamente para combater este problema, e, por conseguinte, são normalmente utilizadas na prática em detrimento das RNNs simples.

Uma das maiores aplicações destas redes é no processamento de texto. O interesse não está na sequência temporal, mas sim na manutenção do sentido da frase. No modelo biológico do ser humano, quando se está a ler uma frase, a informação é processada incrementalmente, mantendo um modelo interno do que se está a processar e que é atualizado à medida que chegam novas informações. As redes neuronais recorrentes funcionam de forma semelhante, pois guardam informação que pode ser atualizada com as novas entradas.

2.4.10.1 *Long Short-Term Memory*

Hochreiter and Schmidhuber (1997) desenvolveram um tipo de rede neuronal recorrente que permite combater o problema da perda de memória ao longo do tempo. Esta designa-se por rede de memória de longo prazo (LSTM). A LSTM é uma variante da RNN simples, aumentada com um caminho para transportar informação intacta ao longo da sua execução. As redes LSTM guardam informação para aceder mais tarde, sem que parte dessa informação desapareça gradualmente (Kent and Salem, 2019).



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

Figura 10.: Célula LSTM [10].

10 https://www.researchgate.net/figure/Structure-of-the-LSTM-cell-and-equations-that-describe-the-gates-of-an-LSTM-cell_fig5_329362532

Em cada neurónio é combinada a informação da entrada com a informação presente no circuito fechado, tendo a nova informação impacto no cálculo do estado e da nova saída.

De acordo com Kent and Salem (2019), a arquitetura de uma célula LSTM inclui três reguladores: *input gate*, *forget gate* e *output gate* (figura 10). As funções de ativação utilizadas na LSTM são a sigmoide (σ) e a tangente hiperbólica (*tanh*), apresentadas na subseção 2.4.3. Como referido anteriormente, as funções de ativação são importantes na inserção de não-linearidades, o que permite ao modelo aprender padrões complexos.

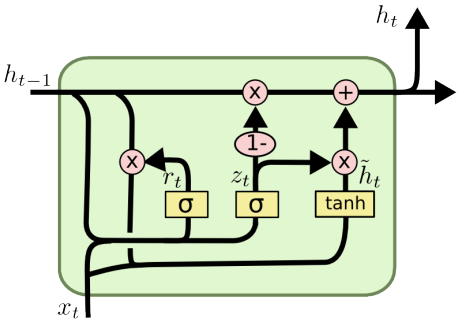
Os sinais i_t (*input gate*), f_t (*forget gate*) e o_t (*output gate*) servem para regular o fluxo de informação. Através do sinal i_t, a porta de entrada decide quais os valores a adicionar de modo a obter o estado atual (C_t). Através do sinal f_t, a porta de *forget* gera um valor entre zero e 1. O valor zero indica que toda a informação de estado anterior é esquecida, enquanto que o valor 1 expressa totalmente o oposto, isto é, a informação do estado anterior mantém-se intacta. A porta de saída, através do sinal o_t, decide qual será o próximo *hidden state* (h_t).

O vetor que contém os novos candidatos a entrar na memória da célula atual (C_t), de acordo com a equação presente na figura 10, designa-se por $\widetilde{C}_t$. Este vetor tem em consideração o *hidden state* anterior (h_{t-1}) e a entrada atual (x_t). A memória da célula atual (C_t), resulta da soma da memória da célula anterior, influenciada pela informação a esquecer ($f_g \times C_{t-1}$), com a nova informação a ser inserida em memória ($i_t \times \widetilde{C}_t$). Para finalizar, o novo *hidden state* resulta do produto entre a tangente hiperbólica da memória atualizada (C_t) e o sinal da porta de saída o_t.

2.4.10.2 *Gated Recurrent Unit*

Tal como a célula LSTM, a GRU resolve o problema da perda de informação ao longo do tempo. Esta célula é mais simples que a LSTM e tem menor poder de representação, o que torna a sua execução mais rápida (Chollet, 2017). A dificuldade em estabelecer um equilíbrio entre tempo de execução e poder de representação, está presente em todos os algoritmos de aprendizagem automática. O que torna a GRU tão especial é a existência de apenas duas portas, uma de atualização (*update*) e outra de *reset*. A GRU funciona como dois vetores, que decidem qual a informação que chega à saída, sem que a informação relevante seja removida (Dey and Salem, 2017).

A célula GRU é mais simples do que a LSTM, pois não possui estado de célula e contém apenas duas portas, porta de atualização (z_t) e porta de *reset* (r_t). A porta de atualização decide a quantidade de informação que transita do passado, enquanto que a porta de *reset* decide a quantidade de informação passada que deve ser descartada (Dey and Salem, 2017). O número de operações matemáticas em cada célula é menor que na LSTM, devido à menor quantidade de portas, o que proporciona maior eficiência computacional e com desempenho funcional muito semelhante.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figura 11.: Célula GRU [11].

2.4.10.3 *Técnicas para melhorar o desempenho das RNNs*

Um problema muito comum na aprendizagem profunda é a presença de *overfitting*. A aplicação de ***dropout*** permite reduzir consideravelmente o problema, embora a sua aplicação nas redes recorrentes não seja trivial. A aplicação de *dropout* antes de uma camada recorrente dificulta a aprendizagem (Chollet, 2017). Gal (2016) descobriu a melhor forma de utilizar *dropout* em redes recorrentes. Em cada passo deve ser aplicado o mesmo padrão, isto é, deve descartar-se a mesma quantidade de unidades. A utilização da mesma quantidade de *dropout* permite que a propagação seja feita de forma adequada, o que não acontece com valores aleatórios. Gal (2016) contribuiu ainda para a criação do mecanismo de *dropout* para redes recorrentes em `keras`, designado por *dropout* recorrente. Consiste em adicionar *dropout* às ligações recorrentes.

Tal como em outros tipos de redes neuronais, geralmente é boa ideia aumentar a capacidade da rede até que o *overfitting* seja o principal obstáculo, assumindo que já se realizaram os passos básicos para o combate do mesmo (utilização de *dropout*). O aumento da capacidade da rede, pode ser feito através da adição de neurónios por camada ou do aumento do número de camadas na rede.

2.4.11 *Generative Adversarial Networks*

As GANs são uma classe de rede neuronal não supervisionada, implementadas com duas redes neuronais. A primeira é uma rede geradora que utiliza pontos do espaço latente e os descodifica numa imagem sintética, a chamada fase de treino. A segunda rede é uma rede discriminadora, que recebe uma imagem da rede anterior e verifica se é uma imagem real ou gerada pela rede. Com esta verificação, garante-se a obtenção de imagens com estilos realistas. A rede discriminadora é treinada com um conjunto de imagens consideradas reais, enquanto que, a rede geradora é treinada para "enganar"a rede discriminadora. As

11 http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-var-GRU.png

GANs geram imagens realistas porque forçam as imagens geradas a serem estatisticamente indistinguíveis das imagens reais (Goodfellow et al., 2014).

As GANs diferem de outros tipos de redes porque não é definido um mínimo de otimização para terminar o treino. A imagem é toda alterada em cada iteração e procuram obter um equilíbrio entre duas forças. Durante a fase de treino o discriminador deve estar "congelado", isto é, os pesos não devem ser alterados. A atualização de pesos levaria a que os resultados previstos classificassem as imagens sempre como reais, o que não é de todo o que se quer obter. Por vezes, o gerador fica preso em imagens geradas que se parecem a ruído. Para combater este problema utiliza-se *dropout* no gerador e no discriminador (Chollet, 2017).

Estas redes são profissionais a falsificar estilos. Supondo que a rede geradora é treinada com pinturas de Van Gogh, aprende o estilo e cria novas pinturas baseadas nesse estilo, como se verifica na figura 12.

Figura 12.: Resultado da aplicação de uma GAN com o estilo de Van Gogh à imagem da esquerda (GanGogh) [12].

2.5 ENGENHARIA DE ATRIBUTOS

Engenharia de atributos é o processo em que se utiliza o conhecimento do domínio dos dados para criar atributos que permitem aos modelos de aprendizagem funcionar melhor. Quando realizado corretamente, pode aumentar consideravelmente o poder de previsão e desta forma fazer a distinção entre bons e maus modelos. Um atributo é geralmente representado por uma coluna no conjunto de dados. Considerando um conjunto de dados bidimensional, as linhas representam entradas, enquanto que as colunas representam os atributos. Esta engenharia visa criar atributos adicionais, ou seja, atributos que não existem inicialmente nos dados. Requer a extração de informação relevante e, muitas vezes, a junção de vários atributos para construir nova informação. Este processo, além de cansativo, consome bastante tempo, especialmente quando é necessário utilizar mais do que um atributo (Chollet, 2017).

12 www.boredpanda.com

Um exemplo do que se pode fazer em engenharia de atributos, é a separação de datas em colunas contendo apenas a hora, o dia, o mês ou o ano. Outro exemplo com datas, seria a extração da idade a partir de um conjunto de dados que contém apenas as datas de nascimento. É muito mais relevante, em termos de aprendizagem, saber a idade de um individuo do que a sua data de nascimento. A criação do novo conhecimento pode ser realizada através de operações como a média, o máximo ou o mínimo. Embora não sejam operações complexas, podem tornar-se mais complicadas quando existem grandes quantidades de dados.

2.6 OVERFITTING E UNDERFITTING

A causa do mau desempenho obtido em modelos de aprendizagem automática está, na maior parte dos casos, relacionada com *overfitting* ou *underfitting*. São os principais causadores de baixo desempenho na execução dos algoritmos, pelo que é essencial aprender a lidar com eles (Nusrat and Jang, 2018). Convém desde já mencionar que, de entre os dois problemas, o que é realmente desafiante é o *overfitting*. O *underfitting* é relativamente fácil de ultrapassar.

Começamos por definir dois conceitos fundamentais, que são a otimização e a generalização. A **otimização**, refere-se ao processo de ajuste de um modelo com o objetivo de conseguir o melhor desempenho com base nos dados de treino, ou seja, o processo de aprendizagem. A **generalização** refere-se ao resultado da aplicação de um modelo em dados que nunca viu. Um modelo que apresente boa generalização tem capacidade de responder adequadamente quando é confrontado com dados que nunca viu (Caruana et al., 2001).

Ocorre ***underfitting*** quando o modelo não consegue aprender suficientemente bem todos os padrões presentes nos dados de treino, o que origina piores resultados. Este tipo de problema ocorre essencialmente em dados de treino com baixa variância e alta tendência. Por vezes, este problema também é provocado por modelos demasiado pequenos, ou seja, modelos simples. Durante a fase de aprendizagem, o erro deve diminuir nos dados de treino e de teste. Enquanto o erro diminui em ambos os casos, existe a possibilidade de melhorar, uma vez que ainda não foram reconhecidos todos os padrões relevantes existentes nos dados de treino (Ghasemian et al., 2019).

Quando as métricas de avaliação começarem a mostrar resultados piores, significa que o modelo está a começar a aprender padrões específicos dos dados de treino. A adaptação do modelo aos dados de treino origina piores resultados quando aplicado a novos dados de teste, devido à menor capacidade de generalização. Este problema designa-se por ***overfitting***. Uma solução óbvia para este problema seria obter mais dados, o que nem sempre é possível (Nielsen, 2018).

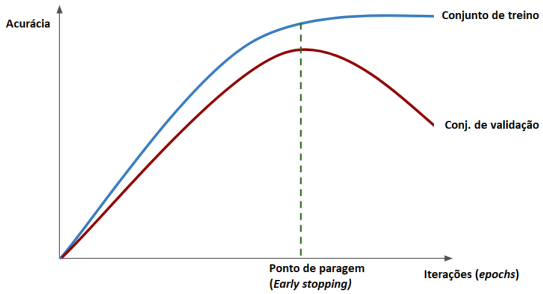


Figura 13.: Um exemplo de *overfitting*.

A figura 13 mostra um exemplo em que um modelo começa a incorrer em *overfitting*. Uma vez que o *overfitting* é originado pela adaptação excessiva do modelo aos dados de treino, facilmente se percebe que o *overfitting* começa quando as curvas da acurácia no treino e validação começam a divergir. Portanto, o modelo apresenta melhores resultados nos dados de treino, mas perde capacidade de generalização e consequentemente os resultados nos dados de validação (ou teste) são piores. A curva azul representa o valor de acurácia para os dados de treino, enquanto que a curva a vermelho representa a acurácia para o conjunto de dados de validação com os quais o modelo nunca teve contacto. Enquanto ambas as curvas crescem não há problema, desde que o treino não termine precocemente (*underfitting*). No entanto, chega-se a um ponto em que a acurácia no conjunto de validação começa a diminuir e a acurácia de treino continua a aumentar. O treino deve terminar nesse ponto, para que o modelo não se adapte especificamente aos dados de treino e generalize cada vez pior (Nusrat and Jang, 2018).

O processo de combate ao *overfitting* designa-se por **regularização**. Apresentam-se de seguida algumas técnicas utilizadas para combater o *overfitting*.

- **Regularização dos pesos**: Uma forma de controlar o *overfitting* consiste em restringir o valor dos pesos da rede. Este controlo faz-se através da adição de um termo extra à função de custo, que aumenta de valor quando os pesos aumentam. O custo adicionado pode ser proporcional ao valor absoluto dos pesos (**regularização L1**) ou ao quadrado do valor dos pesos (**regularização L2**). A regularização L1 afeta apenas valores diferentes de zero, tornando-os mais próximos ou iguais a zero. Geralmente, obtêm-se melhores resultados com a regularização L2. Com L2, os valores aproximam-se de zero mas não chegam a zero, enquanto que com L1 podem chegar. Portanto, L1 funciona como um seletor de caraterísticas, que normalmente provoca piores resultados devido à eliminação de informação (Han et al., 2015).

- **Reduzir o tamanho da rede neuronal**: A forma mais simples para reduzir o *overfitting* é a redução do tamanho do modelo, ou seja, o número de camadas ou o número de neurónios por camada. Normalmente, modelos maiores têm maior capacidade

de memória, o que pode levar a que a aprendizagem seja mais completa nos dados de treino e origina perda de generalização. Quando se tenta reduzir um modelo é importante ter em atenção que a redução não deve ser em demasia, pois pode provocar *underfitting* (o modelo não aprende o suficiente). É importante que se encontre um compromisso entre muita capacidade e capacidade insuficiente. Não existe uma fórmula mágica para encontrar o melhor modelo. Por isso, o processo habitual consiste em começar com modelos pequenos e aumentar progressivamente o número de camadas e o número de neurónios, até atingir o melhor resultado sem *overfitting* (Chollet, 2017).

- **Adicionar *dropout***: A utilização de *dropout* é das técnicas de regularização mais utilizadas e eficientes. Esta técnica foi desenvolvida por Geoffrey Hinton na universidade de Toronto. Quando se aplica *dropout* a uma camada, o que acontece é que parte dos nodos da camada não são considerados na iteração atual. O valor normalmente utilizado varia entre 0.2 (20%) e 0.5 (50%), segundo Chollet (2017). Supondo que uma camada recebe um vetor com tamanho 10, e lhe é aplicado um *dropout* de 0.5, metade desses valores serão colocados a zero (removem-se esses neurónios). A remoção de neurónios diferentes torna a rede muito aleatória, evitando assim a existência de padrões que não são significativos, aos quais Hinton chamou conspirações (Hinton et al., 2014).

2.7 Técnicas de avaliação cruzada

A avaliação cruzada é outro tipo de técnica que também ajuda a evitar o *overfitting*, o principal obstáculo nos modelos de aprendizagem automática. Segundo as boas práticas, o conjunto de dados é dividido em três subconjuntos. Um subconjunto de treino, um subconjunto de teste e um subconjunto de validação. Os modelos não devem ser avaliados com os dados de treino, uma vez que já se adaptaram a esses dados. Por isso, os dados de treino não permitem avaliar a verdadeira capacidade de generalização do modelo. O modelo treina apenas com os dados de treino, é avaliado e afinado com os dados de validação e, no final, o modelo é testado com os dados de teste que nunca viu. Com a utilização destes três subconjuntos consegue-se evitar o ajuste do modelo aos dados de teste.

Nos seguintes métodos de validação cruzada, considera-se que o conjunto de dados de treino é utilizado para treino e validação. Assim sendo, o conjunto de dados divide-se apenas em dois subconjuntos.

Na **Validação *hold-out*** divide-se uma parte do conjunto de dados num subconjunto de treino que será utilizado para treinar o modelo, normalmente 2/3 dos dados (Kohavi, 2001). O restante subconjunto de dados é utilizado para validar o modelo. Para evitar *overfitting*

não se deve utilizar o mesmo conjunto de dados para validação e teste como se viu anteriormente, pelo que é necessário guardar um subconjunto de dados para teste, com o qual o modelo nunca tenha estabelecido contacto. Este é considerado o tipo de validação cruzada mais simples, com um subconjunto para treino (divide-se em treino e validação) e outro para teste. Dependendo do resultado obtido, pode ser necessário ajustar os hiperparâmetros e realizar o processo novamente.

O método **Validação *K-Fold*** representa um aperfeiçoamento da validação *hold-out* e que permite obter melhores resultados. É muito utilizado quando existem poucos dados de treino. O conjunto de dados é divido em n subconjuntos e, para cada uma das n avaliações a realizar, treina-se o modelo com $n-1$ subconjuntos de treino. O subconjunto que sobra, que é diferente em cada avaliação, é utilizado para testar o modelo. Após obter o desempenho das n avaliações, calcula-se a média final. Uma das grandes vantagens da validação *K-Fold* é que não importa como os dados são divididos, uma vez que todos os dados serão utilizados pelo menos uma vez como dados de treino e de teste. Torna-se mais demorado, visto que o modelo é treinado em $n-1$ subconjuntos (Chollet, 2017).

A **Validação *K-Fold* iterativa com mistura dos dados** é o método mais utilizado quando se necessita de um modelo que seja o mais preciso possível com poucos dados de treino. Consiste em aplicar o método anterior I vezes, baralhando os dados antes de serem divididos em n subconjuntos (Chollet, 2017). A avaliação final resulta da média do resultado obtido em todas as I iterações. Se o valor de I for elevado, o método pode tornar-se muito moroso, uma vez que será executado I (número de iterações) x n (número de avaliações e de partições no método *K-fold*) vezes.

2.8 Otimização de modelos

As otimizações fazem a diferença entre uma rede boa e uma rede muito boa. Consegue-se através de padrões de normalização e de *depthwise* construir redes com melhor desempenho.

A normalização é um conjunto de métodos que permite fazer com que diferentes amostras de entrada se tornem mais semelhantes a outras, o que facilita a aprendizagem do modelo. As normalizações utilizadas durante a fase de pré-processamento dos dados normalizam os dados apenas uma vez, antes de alimentarem o modelo. No entanto, a normalização pode ser realizada sempre que é efetuada uma operação no modelo através de ***batch normalization***.

Este método consiste em normalizar os valores de saída de uma camada, antes de entrarem na camada seguinte e ocorre durante a fase de treino. Permite que cada camada da

rede aprenda um pouco mais, independentemente de outras camadas. Também aumenta a estabilidade da rede neuronal, pois aplica à saída das camadas anteriores a subtração da média e divisão pelo desvio-padrão da *batch*.

A técnica ***Depth-wise separable convolution*** permite reduzir o tamanho e a complexidade dos modelos. A motivação reside no facto dos modelos mais pequenos possuírem menor complexidade e menor quantidade de parâmetros, o que implica a realização de menos cálculos. Nos próximos parágrafos apresenta-se uma explicação da técnica.

Consideremos um modelo em que a imagem de entrada tem um tamanho $T_i \times T_i \times N$, onde T_i é a largura e altura da imagem e N é o número de canais (3 no caso de imagens a cores). Supondo que aplicamos k filtros/*kernels* de tamanho $T_j \times T_j \times N$, então uma operação normal de convolução resulta numa saída de tamanho $T_p \times T_p \times k$ (figura 14). Se uma única operação de convolução envolve $T_j \times T_j \times N$ multiplicações, então, o número de multiplicações para toda a imagem é $(T_p^2 \times k) \times (T_j^2 \times N)$.

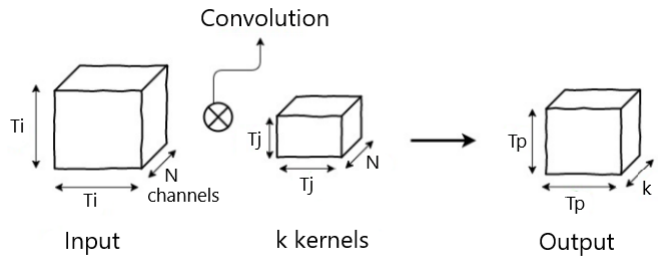


Figura 14.: Convolução normal.

De acordo com Łukasz Kaiser et al. (2017), em *depth-wise separable convolution* o processo divide-se em duas operações diferentes: *depth-wise* e *point-wise*. Na operação de *depth-wise*, a convolução é aplicada apenas a um canal de cada vez. Então os filtros possuem tamanho $T_j \times T_j \times 1$. Para N canais, o tamanho da saída será $T_p \times T_p \times N$ e uma simples convolução requer $T_j \times T_j$ multiplicações. A operação de *depth-wise* requer então $T_p^2 \times N \times T_j^2$ multiplicações.

Na operação *point-wise* aplica-se uma convolução 1×1 aos N canais. Portanto, uma operação tem um custo de $1 \times 1 \times N$ e origina uma saída de $T_p \times T_p \times k$. O custo total desta operação é de $T_p^2 \times N \times k$ multiplicações, onde k é o número de filtros.

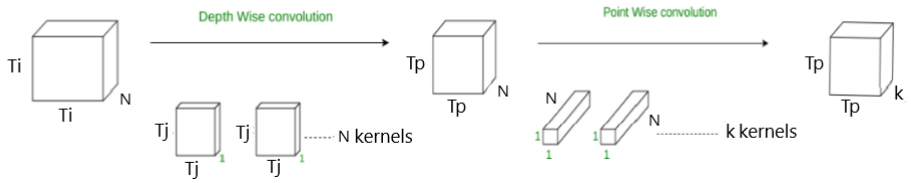


Figura 15.: Convolução *depth-wise*.

Globalmente, as duas operações requerem $(T_p^2 \times N \times T_j^2) + (T_p^2 \times N \times k)$ multiplicações.

$$\text{Conv. normal} = T_p^2 \times k \times T_j^2 \times N \tag{24}$$

$$\text{Conv. depth-wise} = (T_p^2 \times N \times T_j^2) + (T_p^2 \times N \times k) \tag{25}$$

$$\text{Rácio} = \frac{\text{Conv. depth-wise}}{\text{Conv. normal}} = \frac{(T_p^2 \times N \times T_j^2) + (T_p^2 \times N \times k)}{T_p^2 \times k \times T_j^2 \times N} = \frac{1}{k} + \frac{1}{T_j^2} \tag{26}$$

Quando comparamos os dois tipos de convolução, conclui-se que a convolução normal envolve um número muito maior de multiplicações. Supondo que $k = 3$ e $t_j^2 = 25$, o rácio será 0.3733, ou seja, a convolução normal executa praticamente 3 vezes o número multiplicações da convolução *depth-wise*.

2.9 Passos para criar um modelo

Os passos seguintes podem ser utilizados para resolver qualquer problema que envolva redes neuronais artificiais (Chollet, 2017).

1. **Definir o problema**
   Nesta fase deve-se responder às seguintes questões: O que se vai tentar prever? Que dados de entrada estão disponíveis? Os dados estão em quantidade suficiente ou são adequados ao problema?

   Após responder a estas questões é importante perceber o tipo de problema. Pode-se estar perante problemas de classificação binária, classificação de múltiplas classes, regressão, agrupamento[13] ou aprendizagem por reforço.

2. **Obter o conjunto de dados**
   A forma como se obtém o conjunto de dados é pouco relevante. Não interessa se o conjunto de dados já existe ou é criado especificamente para o problema a resolver. O importante é que os dados sejam adequados, preparados e em quantidade suficiente.

13 *Clustering*, na terminologia Inglesa.

Existem alguns pormenores a ter em conta. Por exemplo, um classificador para prever qual o tipo de roupa mais vendida num determinado dia, não deve possuir apenas informação de uma estação do ano, uma vez que a roupa varia conforme o tempo, principalmente entre verão e inverno.

Entre os erros mais frequentes está a utilização de dados em quantidade insuficiente, dados de baixa qualidade (por exemplo, dados inadequados ao problema ou dados com ruído), conjuntos de dados tendenciosos (grande parte dos dados pertence a uma classe e as restantes possuem poucas amostras) e para terminar, dados sem pré-processamento ou validação. Antes de avançar deste passo, deve-se perceber se é possível obter as saídas pretendidas a partir dos dados de entrada. Pode não ser possível encontrar os padrões necessários entre as entradas e as saídas.

3. **Medidas de sucesso**
   Para saber se algo está a correr como previsto, é necessário definir o que é sucesso. Neste passo deve-se decidir qual a função de custo mais adequada ao problema, isto é, o que o modelo vai otimizar. Algumas métricas funcionam melhor em determinado tipo de problema. A tabela 1 apresenta a função de ativação da última camada e a função de custo mais adequadas a cada tipo de problema de aprendizagem automática (Chollet, 2017).

Tipo de problema	**Ativação da última camada**	**Função de custo**
Classificação binária	sigmoide	`binary crossentropy`
Classificação de saída única com múltiplas classes	softmax	`categorical crossentropy`
Classificação com múltiplas saídas e múltiplas classes	sigmoide	`binary crossentropy`
Regressão para valores arbitrários	———————-	`mse`
Regressão para valores entre 0 e 1	sigmoide	`mse` ou `binary crossentropy`

Tabela 1.: Funções de ativação e custo mais adequadas a cada problema.

4. **Protocolo de avaliação**
   Nesta fase é suposto existir uma boa noção do problema e das necessidades que existem para o resolver. Deste modo, opta-se por uma das seguintes técnicas de avaliação cruzada vista anteriormente.
   - *Hold-out validation set*
   - *K-fold cross-validation*
   - *Iterated K-fold validation*

Na maior parte dos casos a primeira técnica é suficiente para obter resultados satisfatórios, no entanto, é aquele caminho que apenas deve ser seguido quando existe uma boa quantidade de dados. A segunda técnica é mais adequada quando existem poucos dados e a precisão do modelo não é preocupante. Por último, a terceira técnica apresentada deve ser utilizada quando se pretende obter modelos precisos e com poucos dados disponíveis.

5. **Preparação dos dados**
   O conjunto de dados deve agora ser preparado antes de ser fornecido ao modelo. Em aprendizagem profunda, os dados devem ser formatados como tensores. Tensores é o termo utilizado para generalizar a noção de escalares, vetores e matrizes na matemática. Os valores presentes nos tensores devem ser pequenos, geralmente entre [-1, 1] ou [0, 1]. Diferentes caraterísticas devem estar no mesmo intervalo de valores. Caso contrário, os dados devem ser normalizados. Pode ser útil realizar extração de caraterísticas, especialmente, quando a quantidade de dados disponível é pequena.

6. **Desenvolvimento do modelo**
   Se os passos anteriores correram como planeado, inicia-se a construção do modelo. O desenvolvimento do modelo em `Keras` divide-se em quatro etapas chave:

   - **Definir o modelo**: Consiste em definir a rede quanto ao número de camadas intermédias, número de neurónios e funções de ativação.
   - **Compilar**: Consiste na configuração da rede neuronal. As configurações mais comuns são a escolha do otimizador, a taxa de aprendizagem, a função de custo e as métricas de avaliação do modelo.
   - **Otimizar**: Corresponde ao treino. Permite a configuração de alguns hiperparâmetros da rede, como o número de *epochs* ou o tamanho de *batch*.
   - **Testar**: O último ponto corresponde ao teste do modelo, preferencialmente com dados que não foram utilizados durante a fase de treino.

`Keras` é uma biblioteca de código aberto, escrita em python, que foi desenhada para permitir criar modelos de redes neuronais de forma eficiente. O Keras pode utilizar como *backend* o TensorFlow, o Microsoft Cognitive Toolkit (CNTK), o Theano ou o MXNet, e foi desenhado sobretudo para facilitar a utilização dos mesmos.

Apresenta-se a seguir um exemplo da construção de um modelo em `Keras`. Inicialmente carrega-se o conjunto de dados para tensores. Divide-se o conjunto de dados em dois subconjuntos: dados de treino e de teste. A rede não pode contactar com os dados de teste para não se adaptar aos mesmos, pelo que é importante que a divisão do conjunto de dados seja feita inicialmente. A extração de representações dos dados é feita nas camadas encadeadas. As representações são recolhidas progressivamente camada a camada. As

camadas funcionam como uma sucessão de filtros, em que a extração de dados se torna mais refinada à medida que passa pelas camadas.

A construção do modelo em `Keras` divide-se nos quatro passos essenciais já descritos: definir o modelo, compilar, otimizar e testar. No primeiro passo personaliza-se a rede quanto ao número de camadas intermédias, ao número de neurónios e as funções de ativação. O método `Dense`, permite criar camadas densamente ligadas. Entre os argumentos mais relevantes está o número de unidades da camada, a função de ativação e a matriz de pesos (*kernel*). Um exemplo:

```
network = models.Sequential()
network.add(layers.Dense(256, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

Antes de otimizar um modelo é preciso efetuar a configuração do processo. O método que permite configurar o treino em `Keras` é o `compile`. Entre os argumentos mais relevantes deste método estão o otimizador, a função de custo e as métricas de avaliação do modelo. Um exemplo:

```
network.compile(optimizer='rmsprop', loss='categorical_crossentropy',
    metrics=['accuracy'])
```

A fase de otimização coincide com o treino da rede, o que se consegue em `Keras` com o método `fit`. Entre os argumentos mais relevantes deste método estão os dados de treino, o número de *epochs*, o tamanho de *batch* e as classes de saída. Um exemplo é:

```
network.fit(images, labels, epochs=10, batch_size=64)
```

De acordo com o código apresentado, o modelo será treinado com *batch size* de 64 durante 10 iterações. O processo de criação de uma rede neuronal termina com a avaliação do modelo. Uma vez treinada a rede, o método `evaluate` permite obter a precisão e o custo (ou erro) do modelo no subconjunto de dados de teste. Um exemplo de código é:

```
test_loss, test_acc = network.evaluate(test_images, test_labels)
```

A partir deste momento, o modelo está pronto a realizar previsões, o que se consegue em `Keras` com o método `predict`.

```
y = model.predict(test_x)
```

2.10 TRABALHO RELACIONADO

A legendagem de imagens é muito mais do que reconhecimento de imagem ou classificação. Esta tarefa possui desafios adicionais como o reconhecimento de dependências entre objetos que fazem parte da mesma imagem e a criação de texto sequencial. Srinivasan et al. (2018), Mullachery and Motwani (2016) e Hossain et al. (2018) são alguns exemplos de trabalhos que permitem legendar imagens automaticamente, como por exemplo, fotografias obtidas no dia a dia através da combinação de redes neuronais convolucionais e redes neuronais recorrentes.

Nguyen and Csallner (2015) provaram ser possível converter capturas de ecrã em código, ao realizar engenharia reversa sem a utilização de aprendizagem profunda. A aplicação Reverse Engineer Mobile Application User Interfaces (REMAUI) permite inferir parte do código fonte de capturas de ecrã ou desenhos concetuais, apenas para aplicações Android ou iOS. A aplicação foi construída com Optical Character Recognition (OCR), detetores de contornos de Canny e a biblioteca OpenCV.

Deng et al. (2017) mostrou que com redes neuronais profundas, CNN e RNNs, se conseguem obter melhores resultados quando comparado com técnicas clássicas como OCR, mesmo em dados manuscritos. Este projeto tem bastante utilidade, na medida em que permite gerar o código em `LaTeX` de uma equação a partir de uma imagem. Afirmam também ser possível obter bons resultados com equações escritas à mão, no entanto, não se focaram neste problema devido à falta de dados para treinar o modelo. Seguem uma abordagem baseada em atenção. A atenção permite salientar caraterísticas importantes presentes numa imagem, algo que o ser humano faz muito bem. Com esta técnica, pretendem evitar a perda de informação relevante (Xu et al., 2015).

Também Chen et al. (2018) conseguiram converter imagens de interfaces com o utilizador (UI) em *skeletons* com aprendizagem profunda, isto é, obter o *layout* e a composição de componentes utilizados na imagem da interface fornecida à rede. A abordagem é muito semelhante à legendagem de imagens ou à conversão de fórmulas para `LaTeX`, recorrendo também a uma CNN e duas RNNs. Embora permita saber que componentes utilizar para obter a interface pretendida, não gera código funcional automaticamente.

Atualmente existe grande curiosidade sobre o que é possível alcançar com a geração de código automático. Ainda numa fase inicial, Beltramelli (2017) mostrou que é possível a partir de esboços de páginas HTML gerar código HTML e CSS. Uma vez que os conjuntos de dados fornecem imagens e o respetivo código associado, é possível através da utilização de uma CNN extrair as caraterísticas das imagens e com uma RNN obter a respetiva descrição da imagem. Tanto a imagem como o código que descreve essa imagem, são codificados e serão enviados como entrada para a segunda RNN num vetor concatenado. A RNN descodificadora foi treinada com dados supervisionados, imagem e respetivo código, ao

contrário das anteriores em que a aprendizagem era não supervisionada. À saída da RNN utiliza-se um compilador que permite obter o código HTML pronto a utilizar. Este projeto foca-se apenas no *layout* e ignora as partes textuais.

Ainda na área da indução de código, Balog et al. (2017) mostraram ser possível treinar uma rede neuronal de aprendizagem profunda para prever propriedades de programas a partir das entradas e respetivas saídas. Esta aplicação permite aumentar as técnicas tradicionais de pesquisa de linguagens de programação, como `SMT-based solver`.

Mou et al. (2015) realizou um estudo onde mostra ser possível converter uma intenção, descrita textualmente, para código C. O modelo foi treinado com código bem estruturado e comentado. Através de redes recorrentes é possível perceber o que o utilizador pretende e gerar parte do código. Destaca-se a capacidade do modelo gerar código diferente daquele que lhe foi fornecido durante o treino. Também apresenta estilo e estrutura de código variável. O modelo nem sempre consegue gerar código completamente correto, o que leva a que o utilizador tenha de fazer ajustes.

Os trabalhos desenvolvidos na área da aprendizagem automática não se limitam apenas à visão por computador ou legendagem de imagens, cobrem uma vasta área como demonstram os seguintes projetos de diversas áreas, envolvendo aplicações Web.

Villamayor (2017) desenvolveu uma aplicação Web que permite reduzir o trabalho repetitivo e tedioso que é elaborar propostas de seguros em seguradoras, neste caso apenas para automóveis. Com um *smartphone* fotografa-se a traseira do automóvel e fornece-se essa imagem ao sistema através da interface Web. Essa interface comunica com uma API REST da qual vai obter respostas. Do lado do servidor a aplicação divide-se em três partes fundamentais, reconhecimento de matrículas, reconhecimento da marca e modelo do automóvel (figura 16).

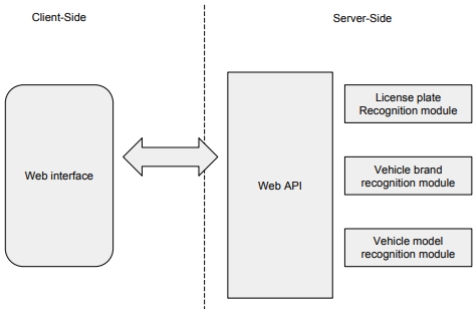


Figura 16.: Arquitetura do projeto.

Esta proposta tem desafios sérios, uma vez que utiliza fotografias tiradas pelo cliente para saber qual é o veículo. Daí, surge a necessidade de lidar com problemas como brilho da imagem, a posição em que é fotografado e a necessidade de desempenho dos algoritmos desenvolvidos. A deteção da matrícula foi realizada através de uma ferramenta open-source

designada OpenALPR. O modelo de aprendizagem automática utilizado foi o mesmo, tanto na deteção da marca como na deteção do modelo do automóvel. A classificação atingiu uma acurácia de cerca de 80%.

Neste projeto, poderia ter sido útil construir redes neuronais convolucionais. Segundo os resultados obtidos em Murillo et al. (2017), que apesar de comparar uma CNN com um classificador Haar num contexto diferente (classificação de instrumentos de cirurgia), conclui que as CNNs são bastante mais rápidas a dar respostas, uma vez que classificadores Haar utilizam uma janela deslizante que é aplicada várias vezes. Relativamente à precisão das suas previsões são semelhantes.

O sistema desenvolvido por Capece et al. (2016) implementa um reconhecedor de moedas com aprendizagem profunda baseado numa arquitetura cliente-servidor. Mostram que é possível obter boas acurácias com CNNs no reconhecimento de moedas. Para isso, necessitaram de um conjunto de dados relativamente grande, uma vez que as CNNs não apresentam bons resultados com poucos dados. Os utilizadores podem fotografar moedas e enviar para o servidor. A imagem fornecida é sujeita ao modelo treinado e classificada com base na informação aprendida durante a fase de treino. Esta abordagem permite aproveitar o Graphics Processing Unit (GPU) para treinar, validar e testar, além de permitir ao cliente receber respostas imediatamente.

Sharma et al. (2018) mostrou que é possível construir uma aplicação Web em Flask para analisar sentimentos em comentários do Twitter. A aplicação Web é bastante simples. O utilizador escreve uma *hashtag*, uma palavra ou frase, e com base nela o sistema devolve um mapa geográfico que indica a média do sentimento associado ao que o utilizador solicitou. Os comentários são obtidos através da API do Twitter, sendo que nem todos permitem saber a localização do utilizador, o que limita os resultados tal como a análise de sentimentos que é realizada apenas em inglês. A aprendizagem automática entra no cálculo do sentimento, que nesta aplicação é realizado através de uma biblioteca do python *textblob*, a qual permite tratar do processamento de língua natural, análise de sentimentos, classificação e tradução.

Na dissertação realizada por Dooley (2017) é descrita a implementação de um modelo de aprendizagem automática e uma aplicação Web para análise estatística de dados dos utilizadores. A aplicação foi desenvolvida em R, é baseada na biblioteca Weka, recebe como entrada um conjunto de dados fornecido pelo utilizador e gera modelos estatísticos que os descrevem. Com esta aplicação os utilizadores conseguem facilmente investigar os seus conjuntos de dados através da visualização de *box plots*, *density plots* e *scatter plots*. Além disso, a aplicação permite testar vários algoritmos de classificação e agrupamento. Embora possa ser aplicada em várias áreas, a aplicação foi utilizada neste projeto para análise de dados desportivos. Para integrar a linguagem R na aplicação Web utilizaram uma tecnologia da Microsoft designada por DeployR. Esta tecnologia permite correr *scripts* em R, o que facilita a execução do código no servidor. Relativamente aos algoritmos de

aprendizagem automática, a aplicação permite classificar com *K-nearest neighbours*, *Support Vector Machines (SVMs)*, *Naive bayes* ou árvores de decisão. Os algoritmos de agrupamento disponíveis são apenas dois, *hierarchical clustering* e *Simple k-means*.

No projeto realizado por Željko Jovanović et al. (2017) apresenta-se a integração de duas *frameworks* Java muito populares (REST e Weka) com uma aplicação desenvolvida em Spring Boot. Através do Weka é possível obter resultados de previsões, uma vez que o Weka permite analisar conjuntos de dados, pré-processar dados, aplicar algoritmos de classificação ou agrupamento. Como exemplo utilizaram uma árvore de decisão à qual lhe forneceram dados relacionados com meteorologia. O acesso a estas funcionalidades é realizado através de uma API REST, que pode ser utilizada por vários tipos de cliente.

3

TECNOLOGIAS

Neste capítulo apresentam-se as tecnologias utilizadas durante a dissertação para desenvolver o conjunto de dados, os modelos de aprendizagem profunda e uma aplicação Web. Para cada tecnologia é apresentada uma introdução, com foco em alguns pormenores técnicos úteis ao trabalho da dissertação.

3.1 CONJUNTO DE DADOS

O conjunto de dados envolvia a criação de imagens, contendo o esboço da interface com utilizador de uma aplicação Web, assim como a respetiva legenda.

Na primeira abordagem com LSTMs, utilizou-se apenas o Balsamiq mockups para criar as imagens e um simples editor de texto para criar os ficheiros com código DSL. A DSL desenvolvida nesta dissertação facilita a escrita de código HTML. É uma linguagem mais simples para um domínio específico e, por conseguinte, não é necessário um programador para a perceber. Devido à maior facilidade em ser aprendida que HTML, o modelo aprende então a gerar código DSL que é posteriormente convertido em código HTML e CSS. Na segunda abordagem com YOLO surgiu a necessidade de categorizar os elementos e criar as regiões delimitadoras de cada imagem devido às necessidades do modelo, portanto, utilizou-se a ferramenta LabelImg. Neste contexto, a legenda é o código DSL para utilizar como entrada em LSTMs no caso da primeira abordagem e a categorização mais as regiões delimitadoras no caso da segunda abordagem.

3.1.1 *Balsamiq Mockups*

A ferramenta Balsamiq Mockups foi lançada em 2008 com o intuito de ajudar as pessoas a desenvolver software e páginas Web com interface mais intuitiva (Balsamiq). A ferramenta permite construir protótipos de baixa fidelidade[1] da interface gráfica das aplicações Web. Os protótipos são construídos no editor da aplicação, utilizando *drag-and-drop*, o que facilita

1 Também conhecidos como esboços ou *mockups*.

a tarefa do utilizador. A elaboração de esboços para a interface de um produto, numa fase inicial do seu desenvolvimento, é importante porque facilitam a discussão e a perceção do que se pretende desenvolver. Desta forma é possível prever com mais exatidão a intenção do cliente e, por conseguinte, evitar possíveis falhas ou esquecimentos na construção do produto. A versão Balsamiq Mockups 3 está disponível para Windows e Mac OS, existindo também uma versão *online*, designada por Balsamiq Cloud. Ambas as versões estão disponíveis para avaliação gratuita durante 30 dias.

3.1.2 *LabelImg*

O LabelImg é uma ferramenta gráfica de anotação de imagens escrita em Python. As anotações são guardadas num ficheiro Extensible Markup Language (XML), segundo o formato seguido pelo ImageNet, designado por Pascal Visual Object Classes (VOC). O ImageNet é um conjunto de dados construído para reconhecimento de imagem, constituído por imagens de plantas, animais, desporto, comida, instrumentos musicais, entre muitas outras categorias com as quais estamos habituados a lidar diariamente. No seu todo é composto por mais de 14 milhões de imagens anotadas, divididas em mais de 20000 categorias. No entanto, pouco mais de 1 milhão dessas imagens possuem caixas delimitadoras e, portanto, apenas essas são utilizadas em modelos de redes neuronais como o YOLO.

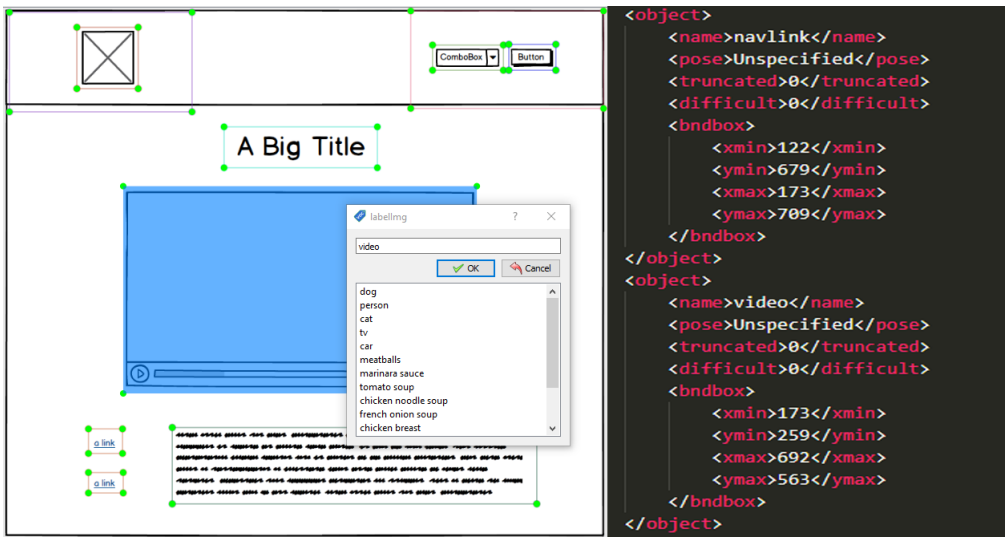


Figura 17.: Regiões delimitadoras, categorização e parte do XML gerado com LabelImg.

No LabelImg as anotações realizam-se na ferramenta gráfica, onde o utilizador apenas seleciona a região delimitadora em volta do objeto e o classifica quanto à categoria. A ferramenta gera automaticamente um ficheiro com os objetos da imagem, onde é descrito

o nome e a região delimitadora através de 2 pontos (x_{min}, y_{min}) e (x_{max}, y_{max}) para cada objeto.

3.2 SERVIDOR E MODELOS DE APRENDIZAGEM PROFUNDA

A construção do *backend* da aplicação Web necessitou de várias tecnologias, com destaque para a linguagem Python. A API REST foi construída em Flask, uma *framework* de Python muito simples e prática. Para os modelos de aprendizagem profunda utilizou-se Keras, uma das várias APIs Python que o TensorFlow disponibiliza. Keras é uma API de alto nível, com uma interface bem concebida, de fácil utilização e que permite uma boa produtividade.

3.2.1 *Python*

Python é uma linguagem fácil de aprender e que permite expressar ideias complexas com clareza. É largamente usada em vários campos da ciência, nomeadamente em inteligência artificial e ciência de dados. É uma linguagem interpretada, compilada em tempo de execução[2], pelo que apresenta alguma perda de desempenho em relação a linguagens compiladas. Contudo apresenta algumas vantagens, como por exemplo efetuar e testar alterações mais facilmente, escrever menos código, escrever código mais legível e por isso mais fácil de corrigir.

O Python está a tornar-se cada vez mais popular devido à sua simplicidade, compatibilidade, possibilidade de usar orientação a objetos e sobretudo por existir uma grande quantidade de bibliotecas. Os tipos de dados são dinâmicos, ao contrário do Java e do C que utilizam tipos estáticos, o que provoca uma alteração significativa na programação. A grande vantagem de utilizar linguagens simples reside no facto do programador ter mais tempo para pensar na solução do que na implementação da mesma. Além disso, tanto permite desenvolver *scripts* simples como programar no paradigma da orientação aos objetos, em projetos de grande envergadura.

Os modelos de aprendizagem profunda e o servidor aplicacional foram desenvolvidos em Python. Atualmente, há duas versões de Python em uso, o Python 2 e o Python 3. As versões não são totalmente compatíveis, pois diferem essencialmente em detalhes de sintaxe. O código desenvolvido nesta dissertação está de acordo com Python 3.

3.2.2 *TensorFlow*

TensorFlow é uma biblioteca de código aberto, para aprendizagem automática, desenvolvida pela Google (Abadi et al., 2016). Embora tenha sido pensada para aprendizagem

2 Do inglês *runtime*.

automática e profunda, permite outro tipo de utilização, como por exemplo, na realização de pesquisas orientadas a dados. Atualmente o TensorFlow é a biblioteca de aprendizagem profunda mais popular, disponibilizando interface para as linguagens Python, C, Go e Java. Para facilitar a sua utilização, foram construídas algumas APIs de alto nível que escondem parte do trabalho complicado do TensorFlow, como é o caso do Keras ou Luminoth. A grande comunidade, a documentação bem escrita e a disponibilização de uma ferramenta de visualização e depuração, o TensorBoard, são algumas das vantagens do TensorFlow. Foi utilizada a versão 1.14 no desenvolvimento dos modelos de aprendizagem profunda.

3.2.3 *Keras*

O Keras é uma API de código aberto, para aprendizagem automática, escrita em Python e que funciona sobre TensorFlow, Microsoft Cognitive Toolkit (CNTK) ou Theano (Keras). Foi desenvolvido para experimentações rápidas, para que seja possível passar da ideia ao resultado no menor tempo possível. Suporta redes neuronais convolucionais e recorrentes, bem como a utilização de ambas em simultâneo. É compatível com a versão 2 e 3 do Python, sendo normalmente usada para realizar projetos de aprendizagem profunda. Utilizou-se a versão 2.2.4 do Keras no desenvolvimento dos modelos.

Keras permite criar protótipos com rapidez e simplicidade. Oferece ainda modularidade e extensibilidade. É uma API construída para humanos, colocando a facilidade de utilização no topo das prioridades através de uma escrita simples, da redução do número de ações requeridas ao utilizador e ainda fornece *feedback* claro sobre os erros que possam ocorrer. O modelo é visto de forma sequencial com módulos totalmente configuráveis, como o número ou o tipo das camadas, as funções de ativação e custo, os otimizadores ou os regularizadores, por exemplo. É facilmente extensível na medida em que novos módulos são facilmente criados e adicionados como novas classes ou funções. Permite treinar os modelos em Central Processing Unit (CPU) e GPU.

3.2.4 *REST*

A arquitetura REST permite criar interoperabilidade entre sistemas computacionais na Internet. Este conceito foi introduzido por Fielding (2000) e não deve ser visto como um protocolo ou uma especificação, mas como um estilo arquitetural. Esta arquitetura define um conjunto de restrições e propriedades baseadas em HyperText Transfer Protocol (HTTP). Quando um sistema solicita informação a um serviço Web, recebe como resposta representações textuais que pode aceder e manipular. A comunicação entre cliente e servidor não guarda o estado, isto é, todos os pedidos são independentes.

Os serviços Web que possuem esta arquitetura obedecem às seguintes restrições:

- **Cliente-Servidor**: Baseia-se no princípio da separação de conceitos, o que permite que ambos evoluam independentemente.
- **Sem estado**: Todos os pedidos são independentes, ou seja, do lado da API REST não é guardada qualquer informação relativa ao cliente. Isto permite que os serviços possuam grande escalabilidade e, desta forma, sirvam grandes quantidades de clientes.
- **Operações**: Existe um conjunto de operações definidas que são o `POST`, `GET`, `PUT` e `DELETE`.
- **Sintaxe universal e bem conhecida**: Os pedidos são realizados todos da mesma forma, de acordo com os parâmetros que é necessário fornecer e sempre com os mesmos Uniform Resource Identifiers (URIs).
- **Representação do estado**: O resultado de um pedido deve utilizar uma representação textual, que pode variar entre JavaScript Object Notation (JSON), HTML ou XML.
- **Sistema em camadas**: Tal como acontece em todas as arquiteturas cliente-servidor, também o REST está preparado para a separação em várias camadas.

3.2.5 *Flask*

Micro-*framework* Web escrita em Python. É simples e altamente personalizável, o que permite criar a própria arquitetura com flexibilidade. Embora seja uma micro-*framework*, i.e., não disponibiliza algumas funcionalidades que é comum encontrar em *frameworks full-stack*, como por exemplo, um Object-Relational Mapping (ORM) próprio. No entanto, não se pode considerar uma desvantagem, uma vez que é possível instalar um ORM à escolha, inclusive o do Django. Devido à simplicidade da *framework* a curva de aprendizagem é mais baixa. Apenas com algumas linhas de código, é possível construir uma simples aplicação Web que funciona como "Hello World", como se mostra na listagem 3.1. Isto só é possível devido à natureza minimalista do Flask que, sem comprometer funcionalidades, permite criar aplicações Web em poucas linhas de código (Flask).

```
from flask import Flask
app = Flask(__name__)

@app.route("/helloworld")
def hello():
    return u'Hello World!'

if __name__ == "__main__":
    app.run()
```

Listagem 3.1: Hello world em Flask.

Apresentam-se a seguir algumas das principais caraterísticas do Flask.

- **Decoradores:** Os decoradores são um dos recursos mais interessantes do Python. São utilizados para injetar funcionalidades adicionais nas funções. Em Flask podem ser usados para implementar os mecanismos de *login*, *caching* ou encaminhamento.
- **Encaminhamento:** O decorador `@app.route` liga uma função a um Uniform Resource Locator (URL). No caso da listagem 3.1, a função `hello()` vai responder aos pedidos com o caminho relativo `"/helloworld"`.
- **Métodos HTTP:** As aplicações Web utilizam diferentes métodos HTTP nos acessos aos URLs. Por omissão, para efetuar pedidos utiliza-se o método `Get`. Ao adicionar o argumento `methods` ao decorador `route` com diferentes métodos HTTP, `@app.route('/helloworld', methods=['GET', 'POST'])`, a função `hello()` passa a responder a pedidos `POST` e `GET`. É ainda possível acrescentar os métodos `Put` e `Delete`.
- **Objeto de pedido:** Para aceder a dados transmitidos nos pedidos `POST` ou `PUT`, pode-se utilizar o atributo `form`. Se a chave não existir é gerado um erro. No caso do `GET`, os parâmetros podem ser enviados no URL e a sua extração é feita com o método `get`.
- **Objeto de resposta:** O valor de retorno de uma função é automaticamente convertido para um objeto de resposta. Supondo que se devolve apenas uma *string*, essa *string* será colocada no corpo da resposta. Por omissão, a resposta será enviada com o tipo HTML e `status code 200`, exatamente como acontece na função `hello()`.
- **Bases de dados:** Para trabalhar com bases de dados relacionais utiliza-se o `SqlAlchemy`, o qual permite operar na base de dados diretamente a partir de Python. Fornece um conjunto de padrões de persistência, projetados para acessos eficientes e de alto desempenho, adaptados à linguagem Python. O Flask também permite utilizar bases de dados não relacionais (Not Only Structured Query Language (NOSQL)), por exemplo através da extensão flask_pymongo.

Flask-RESTful, uma extensão do Flask, adiciona suporte a construções rápidas de APIs REST. Para os utilizadores familiarizados com Flask torna-se fácil compreender a sintaxe da extensão que contém ligeiras diferenças.

A listagem 3.2 mostra como seria a aplicação *Hello World* em `Flask-RESTful`. São poucas as diferenças introduzidas com a utilização de Flask-RESTful. Utiliza-se uma classe para definir os métodos HTTP. A atribuição das rotas faz-se através da função `add_resource`, em vez da utilização de um decorador.

Apresentam-se a seguir algumas caraterísticas do Flask-RESTful.

- **Encaminhamento Resourceful:** Uma classe define as rotas para um ou vários métodos HTTP de um URL. A função `add_resource` faz o mesmo que os decoradores em *Flask* (sem REST), i.e. a atribuição do URL aos métodos HTTP.
- **Objeto de resposta:** As respostas são automaticamente convertidas em JSON. O mesmo não acontece quando não se utiliza esta extensão, pois aí é necessário usar uma função de conversão como `jsonify`.

```
from flask import Flask
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)

class HelloWorld(Resource):
    def get(self):
        return {'message': 'Hello world!'}

    def delete(self, id):
        pass

api.add_resource(HelloWorld, '/helloworld')

if __name__ == '__main__':
    app.run()
```

Listagem 3.2: Hello World em Flask-RESTful.

3.3 Aplicação Web

Nesta secção abordam-se as tecnologias utilizadas no desenvolvimento do *frontend* da aplicação Web. Os *browsers* entendem HTML, CSS e JavaScript, portanto importa introduzir estas tecnologias. Para facilitar o desenvolvimento da aplicação, foi utilizado ReactJS.

Atualmente, é comum utilizarem-se *frameworks* como ReactJS, Angular ou Vue.js para facilitar o desenvolvimento. Por serem baseadas em componentes, ou seja, é possível reutilizar componentes de terceiros, são úteis na criação de Single Page Applications (SPAs). O ReactJS contém diversas bibliotecas que podem ser utilizadas para armazenamento, central ou partilhado, de variáveis e no tratamento de caminhos (*routing*).

3.3.1 *HTML*

HTML é a linguagem base da Internet. A maioria dos documentos da Web está escrito em HTML, daí a sua incontornável importância. Apresenta-se como a única linguagem interpretada pelos *browsers* sendo por isso, a única que permite representar conteúdo. É uma linguagem de marcação, realizada através de *tags* ou marcas. As marcas definem o formato do conteúdo das páginas Web, as quais podem conter texto, imagens, vídeos ou qualquer outro elemento suportado na linguagem.

Os documentos HTML são compostos por cabeçalho e corpo. Geralmente as *tags* surgem aos pares, uma para iniciar e outra para finalizar a marcação. A quebra de linha `<br>` não justifica a existência de uma *tag* de abertura e outra de fecho, uma vez que não possui conteúdo no seu interior, apenas quebra a linha. No entanto, algumas das *tags* devem ser utilizadas para estruturar o código, como é o caso das *tags* `html`, `head` e `body`. A *tag* `head` indica o que é carregado em primeiro lugar numa página, e só após o seu término se avança para a *tag* `body`. O que distingue uma *tag* das restantes palavras da linguagem é a presença do caractere "<" no início e o caractere ">" no fim da palavra.

3.3.2 *CSS*

As folhas de estilo CSS controlam a aparência dos elementos HTML. A principal diferença entre a formatação no próprio HTML e as folhas de estilo, reside na separação dos elementos HTML e na formatação em documentos separados. É possível definir a aparência dos elementos no próprio HTML através da *tag* `style`. No entanto, esta alternativa é menos produtiva, dificulta as alterações e limita a liberdade de formatação.

Entre as várias propriedades do CSS, destaca-se `position` pela importância que tem no posicionamento dos elementos na página Web. Existem várias formas de posicionar os elementos numa página, uma vez que é possível variar o tipo de posicionamento (`static`, `absolute`, `relative` ou `fixed`) e as coordenadas do próprio elemento. Os elementos podem ser colocados em posições exatas com as propriedades `left`, `right`, `top` e `bottom`. Estas propriedades têm efeitos diferentes conforme a propriedade `position` usada.

3.3.3 *JavaScript*

A linguagem JavaScript foi criada em maio de 1995 por Brendan Eich. A ideia por trás da sua criação consistia na utilização de Java no lado do cliente, mas também tornar o HTML mais interativo. À semelhança do Python, o JavaScript é uma linguagem interpretada, de alto nível, com tipos dinâmicos e orientada a objetos. Tal como o Java, utiliza chavetas na sua sintaxe. Juntamente com HTML e CSS formam as linguagens fundamentais da Internet.

A maior parte dos *browsers* tem um motor dedicado para execução de JavaScript, devido à vasta utilização nos mais diversos sítios da Internet. É fundamental na ativação das partes interativas das aplicações Web. Inicialmente, foi desenvolvido para ser executado apenas no lado do cliente.

Atualmente, o NodeJS permite executar código JavaScript fora do *browser*. NodeJS é essencialmente um ambiente de execução para JavaScript, que permite aos programadores utilizarem JavaScript para escrever ferramentas de linha de comando ou programas que executam no servidor com alta escalabilidade. Foi criado para unificar o desenvolvimento Web, dado utilizar a mesma linguagem do lado do cliente e do lado do servidor.

ReactJS, Angular e Vue.js são as bibliotecas JavaScript mais populares para criar as interfaces Web do lado do cliente. De acordo com sondagens do *Stack Overflow* a programadores profissionais, em 2019 as *frameworks* Web mais utilizadas são ReactJS e Angular, com percentagens muito semelhantes. O Vue.js ainda está distanciado, mas tendo em consideração que é mais recente do que as outras duas *frameworks*, e que se encontra entre as *frameworks* mais amadas e procurada em 2019, a tendência é para que aumente a sua utilização no futuro (StackOverflow).

Nesta dissertação optou-se por ReactJS devido à tendência do mercado, a simplicidade e por se tratar de uma *framework* interessante de se aprender.

3.3.4 *Bootstrap*

Bootstrap é uma *framework* Web, de código aberto, útil ao desenvolvimento de componentes da interface gráfica de sítios e aplicações Web. Utiliza HTML, CSS e JavaScript e tem como principal objetivo tornar o trabalho do programador o mais simples possível. Facilita sobretudo a construção de páginas Web simples, apelativas e responsivas.

Foi criado para ser usado no Twitter, com o nome de Twitter Blueprint. Após alguns meses de desenvolvimento, o projeto deu resultados e manteve-se a aposta. Acabou por ser lançado em 2011, como projeto de código aberto e com o nome Bootstrap (Bootstrap). Entretanto, já foram lançadas 4 versões para acrescentar novas funcionalidades. A versão Bootstrap 2 ficou marcado pela introdução da grelha de *layout* responsivo com doze colunas, sem dúvida uma das funcionalidades mais importantes e utilizada desta *framework*. Com esta grelha, uma página Web é facilmente dividida por linhas e colunas, ajustando-se automaticamente ao tamanho do dispositivo.

A versão Bootstrap 3 foi desenvolvida para ser responsiva em dispositivos móveis. Segue uma abordagem "*mobile first*", onde tudo é desenhado para se iniciar em ecrãs menores e posteriormente ser escalado para ecrãs maiores (Bootstrap, 2013). Até à versão 3, o Bootstrap utilizava Leaner Style Sheets (Less), uma extensão da linguagem CSS. Na versão 4 migraram para Syntactically Awesome Stylesheet (Sass). A extensão (ou pré-processador)

Sass procura diminuir a redundância do código CSS, é muito dinâmica e descreve os estilos de maneira estruturada e sequencial. A mudança para Sass permitiu acelerar a compilação do código CSS. Na versão Bootstrap 4 foram ainda introduzidos novos temas e componentes.

A *framework* é muito fácil de utilizar, basta apenas incluir no cabeçalho do código HTML a ligação para o estilo dos componentes CSS e no corpo, inserir os *scripts* de JavaScript do Bootstrap, como mostra o excerto de código 3.3. Os componentes estão disponíveis na página Web do Bootstrap e são utilizados como qualquer outra *tag* HTML. Um exemplo é o botão primário utilizado na listagem 3.3.

```
<!doctype html>
<html lang="en">
    <head>
    ...
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap
        /4.3.1/css/bootstrap.min.css" integrity="sha384-
        ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
        crossorigin="anonymous">
    </head>
    <body>
    <button type="button" class="btn btn-primary">Primary</button>
    ...
    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="
        sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8
        abtTE1Pi6jizo" crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/
        popper.min.js" integrity="sha384-
        UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
        crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/
        bootstrap.min.js" integrity="sha384-
        JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
        crossorigin="anonymous"></script>
    </body>
</html>
```

Listagem 3.3: Exemplo de utilização do Bootstrap.

3.3.5 *ReactJS*

ReactJS, também conhecido como React.js ou simplesmente React, é uma biblioteca JavaScript de código aberto que permite criar interfaces Web. Esta biblioteca é utilizada para desenvolver a camada de apresentação, tanto em aplicações de página única (SPA) como

em aplicações móveis. SPA é um método relativamente recente de desenvolvimento Web. Com esta técnica pretende-se codificar menos do lado do servidor e mais do lado do cliente, proporcionando-se ao cliente uma aplicação dinâmica que apenas carrega conteúdo conforme for necessário. Desta forma, a aplicação está praticamente toda no lado do cliente, sendo o servidor muitas das vezes utilizado apenas para efetuar a comunicação com a base de dados.

O ReactJS é declarativo, diminuindo assim a dificuldade em criar interfaces gráficas. Para cada estado, efetua-se o planeamento da página Web e o ReactJS, automaticamente e de forma eficiente, atualiza e desenha apenas os componentes necessários, de acordo com as alterações nos dados. A *framework* utiliza virtual Document Object Model (DOM), uma abstração do HTML DOM. O HTML DOM foi pensado para páginas estáticas, sendo por isso pouco eficiente a lidar com aplicações dinâmicas. Quando o DOM atualiza, são carregados todos os nodos, assim como o respetivo CSS. As SPAs contêm uma quantidade de nodos elevada, pelo que o HTML DOM teria de verificar todos os nodos para encontrar aqueles que foram alterados, o que iria reduzir consideravelmente o desempenho da aplicação. O virtual DOM foi pensado para corrigir o problema de desempenho do HTML DOM. Consiste numa abstração do HTML DOM que pode ser atualizado sem afetar o DOM atual. O virtual DOM chama-se ReactDOM. Sempre que acontece uma atualização, o ReactDOM executa um processo, denominado reconciliação, que utiliza um algoritmo para comparar as alterações efetuadas. Deste modo, fica a saber quais os elementos a atualizar, mantendo todos os outros elementos inalterados.

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Taylor" />,
  document.getElementById('hello-example')
);
```

Listagem 3.4: Hello World em ReactJS.

Em ReactJS, as páginas são compostas por vários componentes encapsulados, onde cada componente gere o seu próprio estado. Os componentes são implementados pelo método `render`, que recebe dados de entrada e devolve a respetiva vista. Os dados de entrada

são passados para o componente através de `this.props`. A listagem 3.4 mostra um simples `"Hello <nome>"` em ReactJS, onde o `<nome>` é passado ao componente através do `this.props`.

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState(state => ({
      seconds: state.seconds + 1
    }));
  }

  componentDidMount() {
    this.interval = setInterval(() => this.tick(), 1000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    return (
      <div>
        Seconds: {this.state.seconds}
      </div>
    );
  }
}

ReactDOM.render(
  <Timer />,
  document.getElementById('timer-example')
);
```

Listagem 3.5: Exemplo de um componente ReactJS com estado interno.

Os componentes podem manter o estado interno[3], o qual pode ser acedido através de `this.state`. Quando os dados presentes no estado se alterarem é chamado o método `render`, que desenhará novamente o componente. O código presente na listagem 3.5 mostra um componente que desempenha a função de um temporizador. A variável `seconds`

3 Na terminologia Inglesa, estes componentes são chamados de *stateful components*.

presente no estado interno do componente, é iniciada a zero e incrementada a cada segundo. O método `render` do componente será, então, invocado a cada segundo e a cada segundo será atualizada a vista do componente.

Atualmente, existem diversas *frameworks* de código aberto que permitem desenvolver aplicações Web facilmente, como por exemplo o Angular ou o VueJS. Importa por isso olhar um pouco para os benefícios da utilização do ReactJS em detrimento das alternativas. Pode-se dizer que o ReactJS é mais fácil de aprender, devido à abordagem dividida por componentes, com ciclos de vida bastante bem definidos e à utilização de JavaScript simplificado. A arquitetura *Flux* facilita o controlo do fluxo da aplicação através de um ponto central. Ao invés de passar a informação pelos vários elementos filho, todos os componentes podem ter acesso a um ponto central e trabalharem de acordo com esse ponto de controlo. Este aspeto tem grande importância na simplificação do código. Uma das bibliotecas mais utilizadas para implementar esta arquitetura é o Redux. Os pontos fortes do ReactJS são a facilidade de utilização, a escalabilidade e o desempenho do código gerado. Facebook, Instagram e uma grande comunidade de programadores mantêm o ReactJS atualizado.

4

ANÁLISE DO PROBLEMA

A conversão de interfaces gráficas, presumivelmente elaboradas por *Web designers*, para código é uma tarefa realizada por programadores, sempre que se pretende criar um sítio Web ou uma aplicação Web que exibe conteúdo gráfico. Acredita-se que com o desenvolvimento da presente aplicação, será possível eliminar parte do processo de desenvolvimento de software atual. Assim, em vez de apresentar os *screenshots* do que pretende, o *designer* pode disponibilizar parte do código, ao qual o programador apenas terá de acrescentar o código `JavaScript` e personalizar o aspeto da página gerada.

Os próprios *designers* podem, a partir de esboços pouco detalhados, obter o código pretendido sem grande esforço. Este tipo de ferramenta não evitará a utilização de aplicações como o `Photoshop` para construir esboços, mas diminuiria consideravelmente a complexidade e o tempo de desenvolvimento. Desta forma, a partir de um simples esboço obtém-se parte do produto final. Será muito útil principalmente na conversão de protótipos de baixa fidelidade em protótipos de alta fidelidade.

4.1 LEVANTAMENTO DE REQUISITOS

O levantamento de requisitos tem como finalidade identificar as funcionalidades que um produto deve possuir. Envolve a interação com as partes interessadas, da qual resulta um conjunto de requisitos. Existe um conjunto de técnicas que suporta este processo, com destaque para a análise de domínio, questionários, entrevistas ou reuniões com as diversas partes interessadas.

O processo de levantamento e análise de requisitos permite compreender o domínio do problema, obter o conjunto de requisitos, resolver possíveis conflitos e definir prioridades. O levantamento termina com a verificação dos requisitos, de modo a evitar erros na sua descrição ou falta de requisitos. Para que a recolha de requisitos seja realizada corretamente, é importante que o responsável pela recolha compreenda corretamente o domínio. Este processo envolve a interação com as partes interessadas, o que permite obter conhecimento mais profundo sobre o domínio e, por conseguinte, perceber quais as funcionalidades que elas pretendem que sejam implementadas no produto. O envolvimento das várias

partes interessadas pode gerar conflitos que têm que ser resolvidos nesta fase para evitar problemas futuros, que podem até culminar na rejeição do produto. Uma forma de evitar ou resolver os conflitos, consiste na atribuição de prioridades aos requisitos. Existem várias estratégias de priorização de requisitos, mas uma bastante simples e muito utilizada é a técnica MoSCoW, a qual classifica os requisitos em 4 grupos: os que são de implementação obrigatória (correspondem ao *Must*), os importantes mas não essenciais (correspondem ao *Should*), os desejáveis que valorizam o produto (correspondem ao *Could*) e os que podem não ser implementados (correspondem ao *Wont*).

4.1.1 *Análise de domínio*

A análise de domínio consiste na acumulação de conhecimento e experiência até determinado limite. O limite é atingido quando o domínio deixa de ser abstrato, pode ser sintetizado e corretamente utilizado. Então o domínio do problema pode ser analisado através da consulta de sistemas semelhantes ou da leitura de documentação. Com esta análise não se pretende criar sistemas iguais, mas perceber quais os elementos comuns a todos os sistemas nesse domínio. Durante esta análise utilizaram-se alguns artigos e sítios Web, dos quais se destacam o pix2code (Beltramelli, 2017), o Sketch2Code da Microsoft e o Uizard.io.

O Sketch2Code permite transformar desenhos à mão em código HTML através de Inteligência Artificial. Na página inicial do seu sítio Web é possível encontrar exemplos de esboços e carregar a imagem que o utilizador pretende converter em código HTML. Disponibilizam exemplos de teste e alguma informação sobre as funcionalidades da aplicação. A aplicação permite ainda descarregar o código HTML gerado.

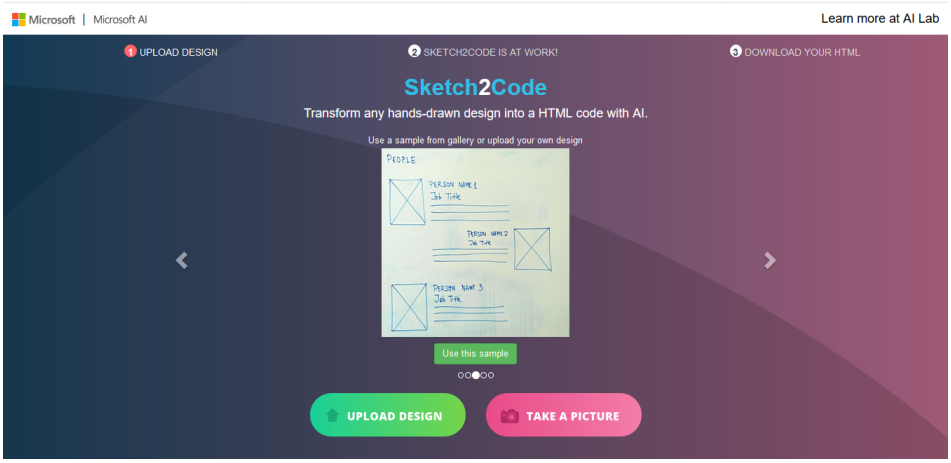


Figura 18.: Página inicial do sítio Web Sketch2Code da Microsoft.

4.1.2 *Entrevistas e Questionários*

As entrevistas permitem capturar os requisitos fundamentais da aplicação junto das partes interessadas mais importantes, assim como identificar os requisitos menos importantes e que podem não ser implementados na versão inicial. Esta é uma das técnicas mais antigas e quase obrigatória em qualquer produto. É importante que o entrevistador prepare a entrevista, para evitar a dispersão e manter o entrevistado focado no problema a resolver. Para que o plano se realize corretamente, é importante que o entrevistador conheça o domínio do problema. Devem ser estudados vários dados importantes para a discussão, como relatórios ou qualquer outro documento ligado ao tema. O entrevistador não deve abusar do uso excessivo de termos técnicos e não fazer da entrevista uma tentativa de persuasão.

Os questionários, tal como as entrevistas, permitem perceber melhor o problema a resolver. Ambas as técnicas esclarecem quais os requisitos fundamentais da aplicação, aqueles que são úteis e, ainda, aqueles requisitos que são menos importantes e, eventualmente descartáveis numa primeira versão. Os questionários são mais adequados quando as partes interessadas se encontram em locais distantes e, por isso, não é possível realizar entrevistas em todos os locais. Também são recomendáveis quando as partes interessadas são em grande número e naturalmente que não podem ser todas entrevistadas. É importante que o tempo gasto a responder ao questionário seja curto. Por isso, os questionários costumam ter questões de escolha múltipla e questões com espaços em branco para a resposta. Os questionários geram maior quantidade de informação, passível de tratamento estatístico, mas menos fiável que as entrevistas.

4.1.3 *Brainstorming*

A principal finalidade da técnica de *brainstorming* consiste na criação de novas ideias, ao potenciar a capacidade criativa dos membros do grupo. É uma técnica de conferência, composta por um grupo de pessoas que tem como objetivo encontrar a solução para um problema específico. As soluções a esses problemas resultam da reunião de ideias dos participantes. Pretende-se com esta técnica, reunir e utilizar a diversidade de pensamentos e experiências vividas dos membros do grupo e, desta forma, encontrar soluções inovadoras. Importa referir que os membros do grupo devem chegar a um consenso após reunir todas as propostas e, desta forma, contribuir para solucionar o problema.

A melhor abordagem para realizar uma sessão de *brainstorming* é trabalhar com um facilitador. O facilitador prepara as questões a resolver e conduz a sessão, além de influenciar na escolha dos participantes. A quantidade de participantes também é importante, pois grupos menores são geridos mais facilmente enquanto que, grupos maiores produzem mais

ideias. Relativamente ao perfil dos participantes, é importante ter membros que não são especialistas no tema, para obter opiniões mais abertas em relação ao tema a discutir.

4.2 REQUISITOS DA APLICAÇÃO WEB DESENVOLVIDA

Não existe uma única "melhor técnica"para capturar requisitos. Para obter melhores resultados é importante aplicar várias técnicas, de preferência as mais adequadas a cada caso.

4.2.1 *Requisitos funcionais*

Descrevem o que o software faz em termos de serviços e tarefas. Cada requisito funcional define uma função do sistema. Uma função é descrita como a entrada do utilizador, o comportamento baseado na entrada e a respetiva saída. Apresentam-se a seguir os requisitos da aplicação Web a desenvolver.

- O utilizador deve obter código HTML funcional após carregar um esboço de página Web;
- O utilizador deve conseguir visualizar e editar o código gerado na aplicação Web;
- O utilizador pode descarregar o código original e o código editado;
- O utilizador edita o código e as alterações devem ser automaticamente transmitidas numa *view*;
- O utilizador deve ter acesso a uma página que forneça informações sobre o funcionamento do modelo de aprendizagem automática e da aplicação;
- O utilizador ao descarregar o código deve obter um ficheiro HTML com o código funcional.

4.2.2 *Requisitos não-funcionais*

São atributos de qualidade do produto, muitas vezes representam restrições que se lhe aplicam. Como por exemplo, o nível de segurança, a disponibilidade, a tecnologia envolvida, a usabilidade ou o tipo de manutenção. Representam qualidades sobre as funções disponibilizadas pelo sistema.

- A comunicação entre *frontend* e *backend* deve ser realizada através de uma API REST;
- A aplicação Web deve ter uma apresentação apelativa e simples de usar;

- O modelo utilizado para converter os esboços em código deve ser desenvolvido com aprendizagem profunda;
- O código gerado pelo modelo deve ser funcional.

4.2.3 *Restrições técnicas*

- Deve utilizar-se a linguagem de programação Python para desenvolver os modelos de aprendizagem profunda;
- A API REST deve ser desenvolvida numa *framework* Python (como Django ou Flask).

4.3 PROTÓTIPOS DA INTERFACE WEB

Os protótipos também são uma técnica de levantamento de requisitos. Embora seja uma opção normalmente dispendiosa e nem sempre disponível, permite validar e enriquecer os requisitos de um produto. Como minimizam os potenciais erros de interpretação das reais intenções das partes interessadas, reduzem os problemas que só seriam detetados depois da implementação. Deste modo, acabam por ajudar a diminuir os custos de desenvolvimento e produção de um produto.

Muitas vezes, um protótipo é um artefacto próximo do produto real e, por conseguinte, ajuda a identificar o que está corretamente delineado, o que pode ser corrigido e o que deve ser descartado. Por vezes, são utilizados protótipos para vender ideias e testar o sucesso de um produto no mercado. A implementação dos mesmos está relacionada com os custos de produção, visto que a partir dos protótipos é possível descobrir falhas e evitar perdas de tempo na construção de produtos que serão revistos no final. Segundo estimativas realizadas por Nielsen (2003) é 100 vezes mais económico efetuar alterações antes da existência de código, do que após a conclusão da implementação.

4.3.1 *Protótipos de baixa fidelidade*

Ao desenvolver um protótipo de baixa fidelidade não é necessário detalhar todas as funcionalidades do produto, visto que serve apenas para melhorar ou alterar um produto que ainda não foi lançado. Este tipo de protótipo possui algumas caraterísticas particulares:

- São pouco detalhados;
- Não possuem interação;
- São fáceis de desenvolver e de corrigir;

- São materializados em papel ou com auxílio de software;
- A sua aparência não é a mesma do produto final.

Geralmente este tipo de protótipo é utilizado para validar os requisitos iniciais de um dado produto. Quando apresentado ao cliente, permite melhorar os requisitos estabelecidos inicialmente, o que permite alterações de acordo com o que ele pretende e evitar desta forma possíveis falhas.

As figuras 19 e 20 constituem o protótipo de baixa fidelidade das páginas Web da aplicação Web desenvolvida.

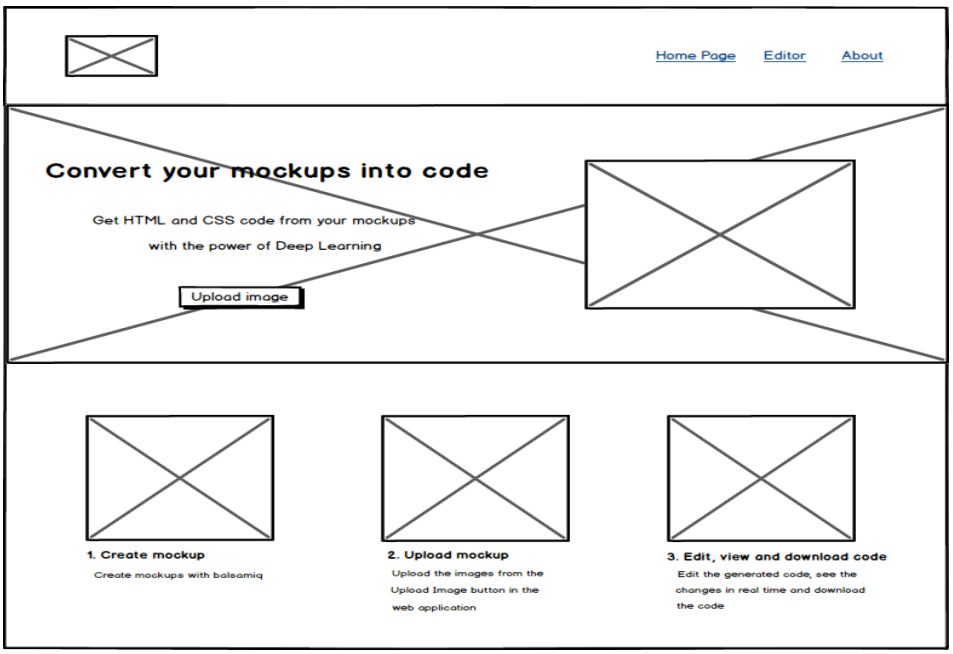


Figura 19.: Protótipo de baixa fidelidade da página principal.

O protótipo representado na figura 19 fornece uma ideia clara daquilo que se pretende incluir no produto final. A página Web é constituída por uma barra de navegação com o logótipo e as ligações para as restantes páginas Web. O título, o texto e uma imagem ilustrativa da aplicação estão colocados em cima de uma imagem de fundo que a torna mais apelativa. Para converter o esboço em código inclui-se um botão que permite selecionar e carregar a imagem. Na parte inferior, colocou-se uma sequência ilustrativa do funcionamento da aplicação constituída por imagens e a respetiva descrição.

A página do editor tem como finalidade fornecer ao utilizador a capacidade de visualizar, editar e descarregar o código gerado pelo modelo de aprendizagem profunda. O protótipo da figura 20 contém botões para descarregar o código original e o código editado, uma janela para editar o código gerado pelo modelo (à esquerda), uma janela com a representação visual do código HTML (à direita) e a barra de navegação, que é comum à página inicial.

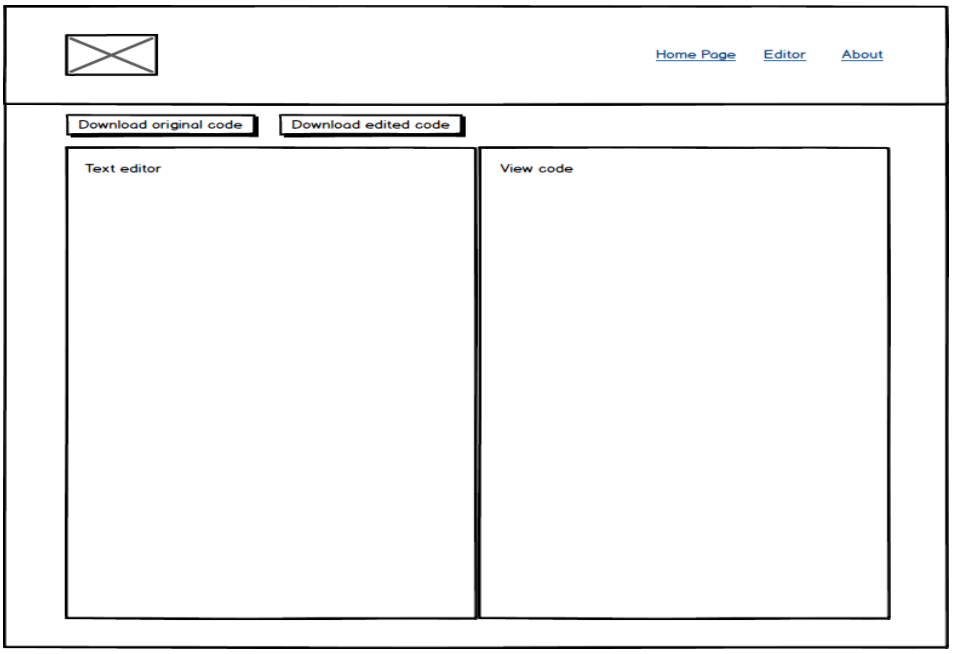


Figura 20.: Protótipo de baixa fidelidade do editor de código.

4.3.2 *Protótipos de alta fidelidade*

Os protótipos de alta fidelidade devem assemelhar-se o máximo possível ao produto final, tanto em termos de funcionalidades (conteúdo, interação e fluxo de navegação) como em aspeto visual. São mais utilizados para vender uma ideia ou realizar testes com utilizadores e, deste modo, melhorar a experiência de utilização dos mesmos.

Estes protótipos baseiam-se nos seguintes princípios:

- Incluir o aspeto visual e funcional da interface;
- A navegação e interação ser igual à do produto final;
- A materialização ser na forma de aplicação.

Quanto mais alta for a fidelidade, melhor será o *feedback* obtido com os testes de usabilidade, visto que o comportamento do utilizador será mais parecido com o comportamento real. O tempo de desenvolvimento é superior, o que torna o custo de desenvolvimento e de correção mais elevado.

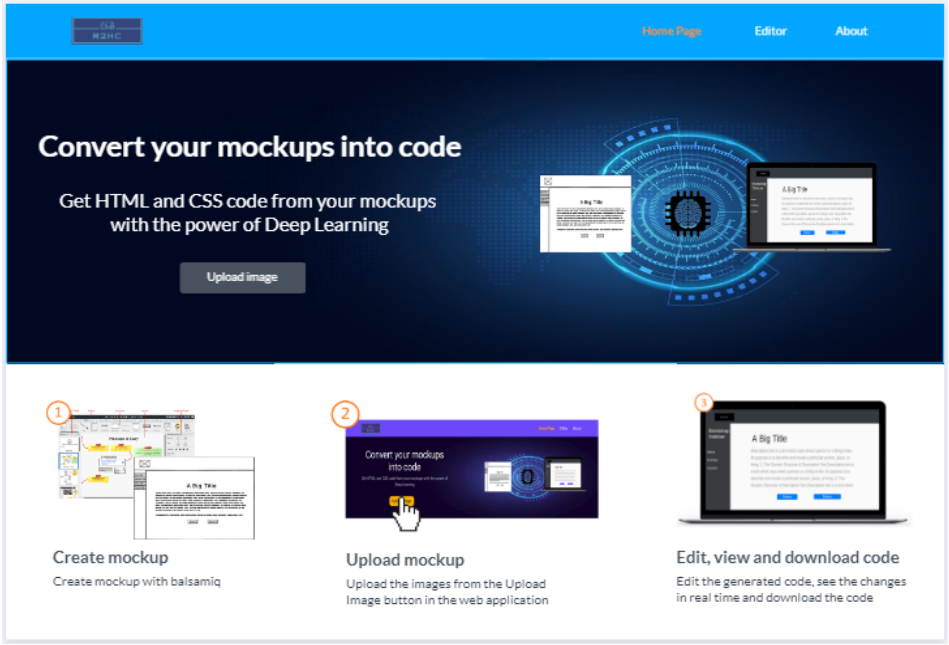


Figura 21.: Protótipo de alta fidelidade da página principal.

O protótipo de alta fidelidade da página principal da aplicação Web desenvolvida encontra-se na figura 21. Constitui uma representação muito próxima daquilo que se pretendia desenvolver no produto final, desde a aparência (estilos, cores, margens e alinhamentos), passando pelo conteúdo e pela interação.

5

DESENVOLVIMENTO DOS MODELOS DE APRENDIZAGEM PROFUNDA

Neste capítulo apresenta-se o conjunto de dados e as abordagens utilizadas no desenvolvimento dos modelos de aprendizagem profunda. A secção 5.1 contém a descrição do conjunto de dados, descrevem-se os elementos HTML suportados, o código DSL utilizado na abordagem 1 e as anotações XML da abordagem 2. Na secção 5.2 encontra-se a descrição detalhada da leitura e armazenamento dos dados, as redes neuronais utilizadas, o código DSL criado, o compilador, a arquitetura do modelo e a métrica de avaliação utilizada na primeira abordagem. Para terminar, a secção 5.3 contém a descrição do YOLO v1 e as várias melhorias introduzidas nas versões seguintes, o algoritmo de *layout* criado, a arquitetura do modelo e a métrica de avaliação desenvolvida na segunda abordagem.

5.1 CONJUNTO DE DADOS

A falta de conjuntos de dados com esboços Web suficientemente complexos, levou à construção de raiz do próprio conjunto de dados. Os esboços de páginas Web, foram desenhados com auxilio da ferramenta `Balsamiq Mockups`, disponível *online* ou em ambiente Windows e Mac OS.

O conjunto de dados desenvolvido suporta parte dos componentes Web mais utilizados no `Bootstrap`, como imagens, vídeos, botões, barras de navegação e tabelas. É constituído por 1000 imagens de treino e 100 imagens de teste, num total de 1100 imagens. Suporta os 15 elementos do `Bootstrap` que se apresentam a seguir. Estes elementos permitem criar grande variedade de páginas Web e servem as necessidades do projeto. A utilização de apenas um subconjunto de elementos está relacionada com a quantidade de esboços que seria necessário criar para suportar todos os elementos do `Bootstrap`. Os elementos contidos no interior da barra de navegação superior são reconhecidos como elementos independentes, ou seja, os elementos no interior da barra são reconhecidos pelo modelo. O mesmo não acontece na barra de navegação lateral pois, não é realizado o reconhecimento dos elementos internos por uma questão de simplificação e redução do tamanho do conjunto de dados.

- Imagem

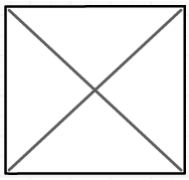


Figura 22.: Representação de uma imagem.

- Vídeo

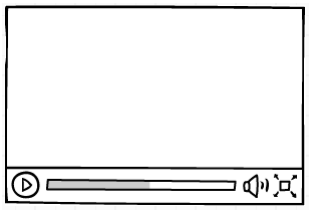


Figura 23.: Representação de um vídeo.

- *Slideshow*

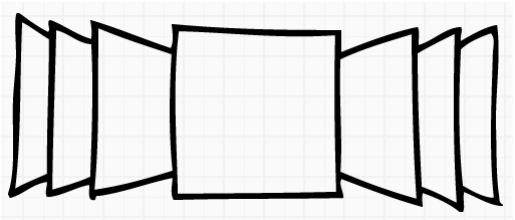


Figura 24.: Representação de um *slideshow*.

- Tabela

Name (job title)	Age	Nickname	Employee
Giacomo Guilizzoni Founder & CEO	40	Peldi	◉
Marco Botton Tuttofare	38		☑
Mariah Maclachlan Better Half	41	Patata	⊟
Valerie Liberty Head Chef	:)	Val	☑
Data Grid Docs			☐

Figura 25.: Representação de uma tabela.

- Bloco de texto

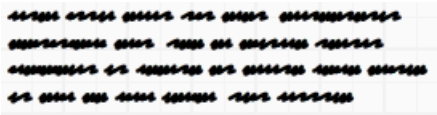


Figura 26.: Representação de um bloco de texto.

- Título

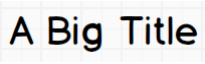


Figura 27.: Representação de um título.

- Botão

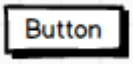


Figura 28.: Representação de um botão.

- Botão de seleção

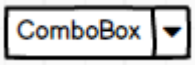


Figura 29.: Representação de um botão de seleção.

- Seletor de data

Figura 30.: Representação de um seletor de data.

- Barra de pesquisa

Figura 31.: Representação de uma barra de pesquisa.

- Barra de navegação superior (*navbar*)

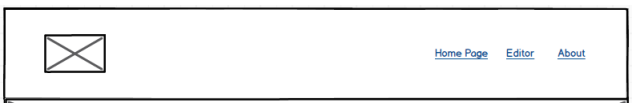


Figura 32.: Exemplo de uma barra de navegação superior.

- Barra de navegação lateral (*sidebar*)

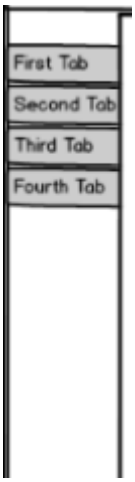


Figura 33.: Representação de uma barra de navegação lateral.

- Ligação

Figura 34.: Representação de uma ligação (*link*).

- Etiqueta

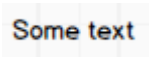


Figura 35.: Representação de uma etiqueta (*label*).

- Caixa para introdução de texto

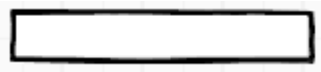


Figura 36.: Representação de uma caixa para introdução de texto.

O conjunto de dados é constituído por imagens a cores, com 3 canais, que resultam diretamente do `Balsamiq Mockups`. As imagens de entrada necessitam ainda de pré-processamento para que sejam aplicadas no modelo com o mesmo tamanho. Utilizou-se um tamanho de 256 x 256 pixéis.

Os modelos desenvolvidos nas duas abordagens têm necessidades diferentes, pelo que foi necessário legendar as imagens da abordagem 1 com código DSL (secção 5.2) e criar anotações XML para a abordagem 2 (secção 5.3).

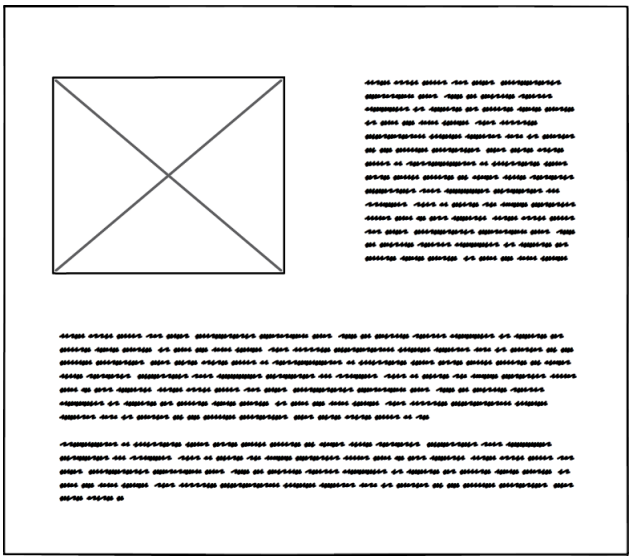


Figura 37.: Exemplo de um esboço presente no conjunto de dados.

As RNNs (primeira abordagem) são o único tipo de rede neuronal que pode receber informação de entrada e guardar em memória parte dessa informação. A informação guardada, pode ser posteriormente utilizada nas entradas seguintes. Para trabalhar com imagens, a arquitetura apropriada tem que incluir CNNs. A utilização de CNNs permite obter a estrutura espacial da imagem de entrada, que é posteriormente fornecida segundo uma sequência temporal à GRU. A mesma imagem é concatenada várias vezes com sequências de código DSL. Este código representa a imagem.

O código DSL utilizado para legendar as imagens, divide os esboços em linhas e colunas. No exemplo da figura 37 existem duas linhas, a primeira linha é dividida em duas colunas (uma imagem e um bloco de texto) e a segunda linha contém apenas uma coluna (texto). O código DSL presente na listagem 5.1 representa o esboço da figura 37.

```
content {
    row {
        col {
            image
        }
        col {
            p
        }
    }
    row {
        p
    }
}
```

Listagem 5.1: Código DSL correspondente ao esboço da figura 37.

O modelo da segunda abordagem (YOLO) aprende com o conjunto de dados a localizar e categorizar objetos, mas para isso necessita de receber essa informação como entrada. O esboço da figura 37 é composto por uma imagem e dois blocos de texto. No ficheiro XML os objetos são descritos pelo nome e a região delimitadora (x_{min}, y_{min}) e (x_{max}, y_{max}), de acordo com a saída da ferramenta `labelImg`. O código XML presente na listagem 5.2, corresponde ao esboço da figura 37.

```
<object>
  <name>image</name>
  <bndbox>
    <xmin>56</xmin>
    <ymin>81</ymin>
    <xmax>358</xmax>
    <ymax>342</ymax>
  </bndbox>
</object>
<object>
  <name>p</name>
  <bndbox>
    <xmin>447</xmin>
    <ymin>86</ymin>
    <xmax>720</xmax>
    <ymax>325</ymax>
  </bndbox>
</object>
<object>
  <name>p</name>
  <bndbox>
    <xmin>64</xmin>
    <ymin>402</ymin>
    <xmax>723</xmax>
    <ymax>628</ymax>
  </bndbox>
</object>
```

Listagem 5.2: Código XML relativo ao esboço da figura 37.

O código XML presente na listagem acima divide-se em 3 objetos (etiqueta *object*). Cada objeto possuí um nome e uma região delimitadora (etiqueta *bndbox*). A região delimitadora é constituída por dois pontos (x_{min}, y_{min}) e (x_{max}, y_{max}).

5.2 ABORDAGEM 1 COM CNN E RNNS

Esta abordagem é muito semelhante à legendagem de imagens com aprendizagem profunda, devido à arquitetura codificador-descodificador. O codificador é composto por duas redes neuronais, uma CNN que recebe uma imagem e uma RNN que recebe texto como entrada. A CNN gera um vetor de caraterísticas como saída. A saída resultante de ambas as redes neuronais contém a informação codificada da imagem e do respetivo código DSL sobre a forma de vetores, concatenados posteriormente num único vetor. A rede neuronal descodificadora que não é obrigatoriamente igual ao codificador, desempenha exatamente o papel oposto do codificador. Recebe como entrada o vetor de caraterísticas concatenado e dá a correspondência mais próxima com base na entrada. O codificador e o descodificador são treinados em conjunto e trabalham na redução da função de custo.

A arquitetura da primeira abordagem combina assim CNNs com RNNs para receber uma imagem como entrada e gerar uma legenda para essa imagem. Neste caso, a legendagem dos esboços realiza-se com código DSL em vez da habitual descrição textual num idioma como Português ou Inglês, por exemplo. A CNN é usada para extrair e criar um vetor de caraterísticas. Este vetor representa a imagem e será fornecido como entrada à RNN descodificadora, juntamente com sequências de código DSL proveniente da RNN codificadora. As RNNs são normalmente utilizadas em problemas com dependências temporais, dado que permitem guardar informação de estados anteriores e utilizá-la em previsões futuras. Ou seja, as RNNs têm memória.

Tendo em consideração a complexidade e o tamanho do código HTML e CSS, surgiu a necessidade de criar uma DSL que facilite a geração automática de código. A DSL é uma linguagem de domínio e, por isso, construída especificamente para resolver um problema. Como o utilizador pretende obter código HTML e CSS, foi necessário criar um compilador que traduza o código DSL na linguagem pretendida. Para este efeito, utilizou-se a ferramenta ANTLR4 em Python.

5.2.1 *Pré-processamento do conjunto de dados*

Como o conjunto de dados foi cuidadosamente construído para a presente dissertação, o pré-processamento acabou por ser uma tarefa simples. As imagens são pré-processadas ao abrir, para garantir que todas apresentam o mesmo tamanho. Depois são redimensionadas para 256×256 pixéis. Relativamente ao ficheiro com o código DSL, o pré-processamento envolve apenas a inserção de duas etiquetas, uma inicial e outra final, para que o modelo compreenda quando o código inicia e termina.

Dada a quantidade de dados com que se trabalha neste conjunto de dados, a memória disponível na máquina utilizada para treinar os modelos era insuficiente. Deste modo, foi

necessário encontrar uma solução que permitisse realizar a tarefa com eficiência. Utilizou-se um gerador de dados que permite evitar o esgotamento de memória da máquina pois, este gera o conjunto de dados em tempo real e fornece os dados separadamente ao modelo. O gerador de dados é uma função semelhante a um iterador. Não guarda dados em memória e itera sobre um elemento de cada vez, o que pode resultar numa execução mais lenta. Desta forma as imagens e o código não esgotam a memória, uma vez que são lidos e fornecidos ao modelo iterativamente. Consegue-se assim reduzir a quantidade de memória necessária para processar enormes conjuntos de dados.

5.2.2 *Linguagem específica de um domínio*

As linguagens específicas de um domínio são linguagens de programação, com expressividade limitada, focadas num determinado domínio aplicacional. Entende-se por expressividade limitada, o facto de apenas servir os requisitos mínimos para o domínio onde é aplicada. O oposto são as linguagens de propósito geral, como Java, C ou Python.

O facto da linguagem de programação ser construída especificamente para o domínio do problema, facilita a sua interpretação, uma vez que é composta por elementos e relações que representam diretamente a lógica do problema. Pode ser utilizada por pessoas que conhecem o domínio e que não têm de ser programadores, principalmente quando existe uma visualização gráfica do domínio do problema. Entre as DSLs mais conhecidas estão o Structured Query Language (SQL) e o CSS. O SQL é usado para lidar com dados relacionais e CSS para definir estilos em HTML.

A utilização de uma DSL na primeira abordagem foi essencial na medida em que não seria possível o modelo aprender a gerar código HTML e CSS corretamente devido à sua complexidade. A DSL simplifica o código a gerar e facilita o trabalho do modelo de aprendizagem profunda.

```
content {
    row {
        col {
            p
        }
    } row {
        col {
            image
        }
        col {
            p
    }}
}
```

Listagem 5.3: Exemplo de código DSL criado.

O código incluído na listagem 5.3 representa um simples esboço de duas linhas. Na primeira linha, contém uma coluna com um bloco de texto (p), enquanto que a segunda linha contém duas colunas. A primeira coluna tem uma imagem e a segunda um bloco de texto (p). Devido à simplicidade da DSL desenvolvida, o código HTML gerado fica limitado a estar alinhado por linha e por coluna.

5.2.3 *Arquitetura do modelo desenvolvido*

A arquitetura varia da fase de treino para a fase de inferência. Durante a **fase de treino** o modelo recebe como entrada um vetor concatenado com as caraterísticas da imagem e o respetivo código DSL, enquanto que na fase de inferência o vetor de entrada contém apenas a imagem e a etiqueta de início.

O modelo segue uma arquitetura codificador-descodificador, normalmente utilizada em tradução automática ou em legendagem de imagens. Durante a codificação transformam-se as entradas, neste caso a imagem e o respetivo código DSL, em vetores de comprimento fixo. Na descodificação é interpretado o vetor codificado resultante da codificação. A descodificação varia conforme a fase em que o modelo está a ser utilizado. Durante o treino, o descodificador recebe dois vetores concatenados que utiliza para aprender a relação entre a imagem e o respetivo código DSL (figura 38).

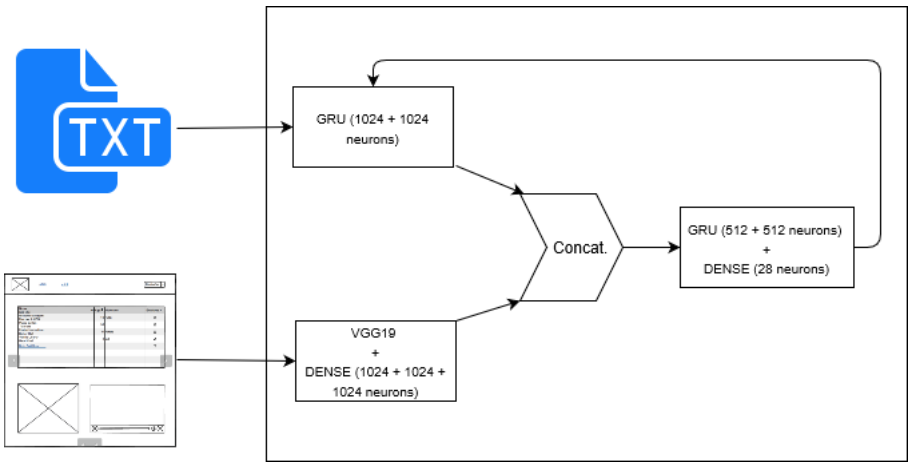


Figura 38.: Arquitetura do modelo durante a fase de treino.

De acordo com a figura 38, o modelo recebe como entrada o código DSL e o respetivo esboço. O código DSL entra numa RNN e o esboço numa CNN. Obtêm-se dois vetores de comprimento fixo, posteriormente concatenados e fornecidos a uma segunda RNN. A segunda RNN, aprende a relação entre o esboço da página Web e as sequências de código DSL.

O modelo desenvolvido é composto por uma CNN e duas RNNs com células GRU. A CNN é uma VGG19, seguida de 3 camadas densamente ligadas, cada uma com 1024 neurónios. A primeira GRU responsável pela codificação do código DSL, é constituída por 2 camadas com 1024 neurónios em cada. A segunda rede GRU trata da descodificação e é composta por 2 camadas com 512 células cada. O descodificador termina com uma camada densamente ligada que possuí um número de neurónios igual ao tamanho do vocabulário da linguagem DSL desenvolvida. Foi utilizado o otimizador `Adam`, com uma taxa de aprendizagem de 0.00001.

Durante a **fase de inferência**, o modelo recebe o vetor com a codificação da imagem e a etiqueta inicial, dado que o restante código DSL será gerado (figura 39). À medida que o modelo gera código para a imagem, a sequência resultante cresce até se atingir o número máximo de iterações pré-definido ou até ser gerada a etiqueta final, a qual termina o processo.

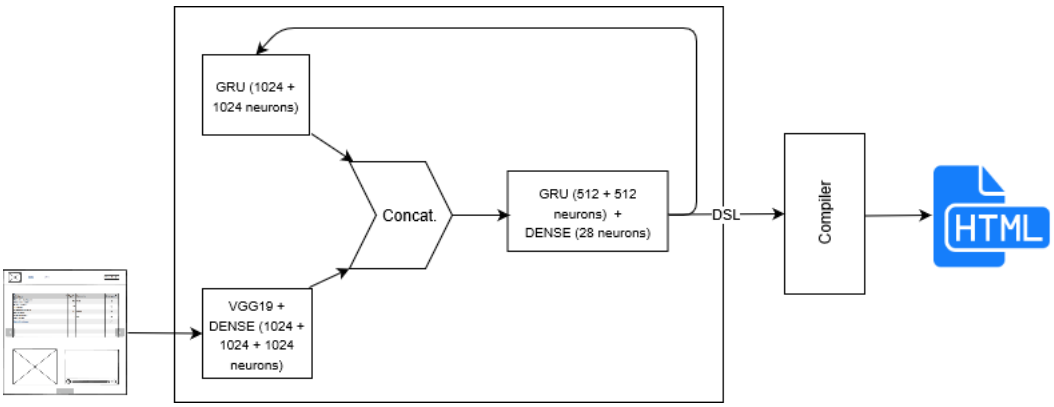


Figura 39.: Arquitetura do modelo durante a fase de predição.

A imagem de entrada é codificada através de uma CNN. Com base nos testes realizados para o conjunto de dados desenvolvido, a utilização de *transfer learning* com uma rede VGG19 proporciona resultados satisfatórios, tendo em conta as limitações do conjunto de dados. A `VGG16` é uma CNN, proposta por Simonyan and Zisserman (2015), constituída por camadas convolucionais, camadas densamente ligadas e camadas de amostragem (*max pooling*) e na saída uma função *softmax*, utilizada na previsão em múltiplas classes. A `VGG19` utilizada no modelo desenvolvido na abordagem 1, é uma variante da versão original que contém mais camadas.

Estas redes possuem grande quantidade de filtros com convoluções 3×3. A utilização de filtros menores reduz consideravelmente o número de parâmetros necessários, quando comparado com outras redes existentes e, consequentemente, reduz consideravelmente o tempo de treino.

O método conhecido por **transfer learning** com extração de caraterísticas permite reutilizar um modelo pré-treinado na resolução de um novo problema. Como base na resolução do novo problema, utiliza-se o resultado do treino realizado num conjunto de dados diferente e "congela-se"as camadas de extração de caraterísticas, ou seja, os pesos dessas camadas não são atualizados. Substitui-se as camadas de classificação para que o modelo aprenda a classificar com base no novo conjunto de dados. Como é difícil obter conjuntos de dados com tamanho suficientemente grande, raramente as CNNs são treinadas do zero. Esta técnica permite obter facilmente melhores resultados e poupanças consideráveis de tempo. A CNN VGG19 utilizada no modelo desenvolvido reutiliza o conjunto de dados `ImageNet`. O conjunto de dados `ImageNet` contém mais de 14 milhões de imagens legendadas, divididas em mais de 20000 categorias, como carro, bola ou gato.

Fine-tuning é um processo que permite reutilizar um modelo treinado para executar uma tarefa semelhante. Através da reutilização de parte da rede, é possível usufruir principalmente da extração de caraterísticas que acontece nas primeiras camadas da CNN, que neste caso é a VGG19. Portanto, a técnica de *fine-tuning* consiste em "congelar"as camadas do topo da rede, de forma a evitar a perda de informação nessas camadas. Para as camadas que não são "congeladas", otimizam-se os pesos para as caraterísticas do novo conjunto de dados, que neste caso em concreto são os esboços das páginas Web.

Embora estas técnicas sejam semelhantes, diferem nas camadas que são "congeladas". Com a técnica de *transfer learning* com extração de caraterísticas as camadas de extração não aprendem e nem se adaptam ao novo conjunto de dados ("congeladas"), enquanto que com *fine-tuning* uma percentagem das camadas de extração aprende sobre o novo conjunto de dados.

As camadas densamente ligadas, colocadas após as convoluções, criam o vetor de caraterísticas da imagem. Este vetor é posteriormente concatenado com o vetor que contém a codificação do código proveniente do codificador da DSL dessa imagem. Por questões de desempenho, utilizaram-se nas camadas RNN células CuDNNGRU do Keras, baseadas na biblioteca cuDNN da NVIDIA. cuDNN é uma biblioteca de primitivas otimizadas para GPU, que disponibiliza versões otimizadas das camadas de convolução, normalização ou ativação (Jeremy Appleyard and Blunsom, 2016).

5.2.4 *Compilador*

Num contexto em que existem inúmeras linguagens de programação, a maioria de alto nível, os compiladores desempenham um papel muito importante. Um compilador é um programa que traduz código de alto nível para código num nível mais baixo. No caso das linguagens mais conhecidas, como C ou Java, o compilador traduz uma linguagem de alto nível que é compreendida pelo utilizador, para uma linguagem de mais baixo nível que a

máquina consegue executar. Ferramentas como o ANTLR4 permitem criar a nossa própria linguagem de programação. O ANTLR4 é um poderoso gerador de *parsers*, utilizado para ler, processar, executar e traduzir texto.

Os compiladores dividem-se em três componentes: *lexer*, *parser* e gerador de código. O papel do *lexer* consiste em dividir o código de entrada em *tokens*. Um *token* pode ser apenas um caractere ou um conjunto de carateres, isto é, uma palavra especial que é posteriormente utilizada pelo *parser*. A separação em *tokens*, é conhecida como a análise léxica.

O *parser* verifica a sintaxe do código do programa. Recebe como entrada um conjunto de *tokens* e cria uma árvore de sintaxe abstrata[1] como saída. As ASTs são estruturas de dados em árvore que representam a estrutura sintática do código fonte, de acordo com uma gramática formal. Cada nó da árvore denota os símbolos não terminais, enquanto que as folhas representam os símbolos terminais, os quais podem ser variáveis ou constantes. A análise da sintaxe, ou *parsing*, consiste na análise do *token* segundo as regras de sintaxe da gramática e posterior geração da AST. O analisador semântico verifica, através da AST, se as regras de semântica da linguagem são cumpridas.

A última etapa consiste na geração de código a partir da AST. Cria-se uma representação intermédia do código, que será otimizada posteriormente. Para terminar, é gerado o código final a partir do código otimizado.

A gramática desenvolvida para solucionar o problema de conversão de esboços em código HTML, é constituída por um conjunto de símbolos terminais, onde se incluem apenas elementos do `Bootstrap` como imagens, vídeos, ligações ou tabelas. Estes símbolos não podem ser divididos em unidades menores, daí a designação de terminal. A barra de navegação (*navbar*) é composta por um conjunto de elementos, como por exemplo, botões, imagens ou títulos. Divide-se em unidades menores e por isso designa-se por símbolo não-terminal. Os símbolos não-terminais são constituídos por combinações de símbolos terminais e não-terminais. Na DSL desenvolvida, os símbolos não-terminais mais utilizados são as chavetas, linhas e colunas. As chavetas são utilizadas para definir os limites do conteúdo existente numa linha, numa coluna ou num elemento não-terminal, como acontece na barra de navegação superior (*navbar*). As linhas e colunas dividem as páginas de forma semelhante às grelhas do `Bootstrap`.

5.2.5 *Métrica de avaliação*

Foi desenvolvida uma métrica especificamente para avaliar o código gerado pelo modelo de aprendizagem profunda. As métricas mais utilizadas em tradução automática, como é o caso do `Bleu score`, comparam apenas o código gerado com o código esperado, palavra a

1 abstract syntax tree (AST), na terminologia Inglesa.

palavra, que no nosso caso iria desprezar o mais importante, ou seja, o número de elementos corretos.

A utilização de humanos para avaliar legendas seria demorada e, consequentemente, apresentaria custos elevados. Na tentativa de resolver este problema, foi proposto um método rápido, barato e que suportasse vários tipos de linguagens, o `Bleu score`. O `Bleu score` é um algoritmo, muito utilizado em tradução automática, que avalia a qualidade do texto gerado. A qualidade é definida pela correspondência entre a saída da máquina e a legenda realizada por um humano. Quanto mais próxima for a legenda gerada da legendagem efetuada por humanos, melhor será a legenda obtida automaticamente.

O `Bleu score` considera as palavras sinónimas uma equivalência errada. Na geração automática de código, esta limitação não constitui um problema porque não existem sinónimos no vocabulário das linguagens de programação. Aqui, cada componente é representado por uma única palavra, o que facilita a utilização do método `Bleu score`. A análise é realizada com base em *n-grams*. Em linguística computacional, os *n-grams* são conjuntos de vários elementos de uma amostra de texto ou voz. Esses elementos podem ser letras ou palavras, de acordo com a aplicação. No exemplo da figura 40 exemplifica-se o funcionamento do método. Na primeira frase (*1-gram*) apenas uma palavra está errada. Ao aumentar o número de *gram* para 2, as palavras erradas distribuem-se por mais *grams* tornando-os errados, como se verifica também com *3-gram* ou *4-gram*.

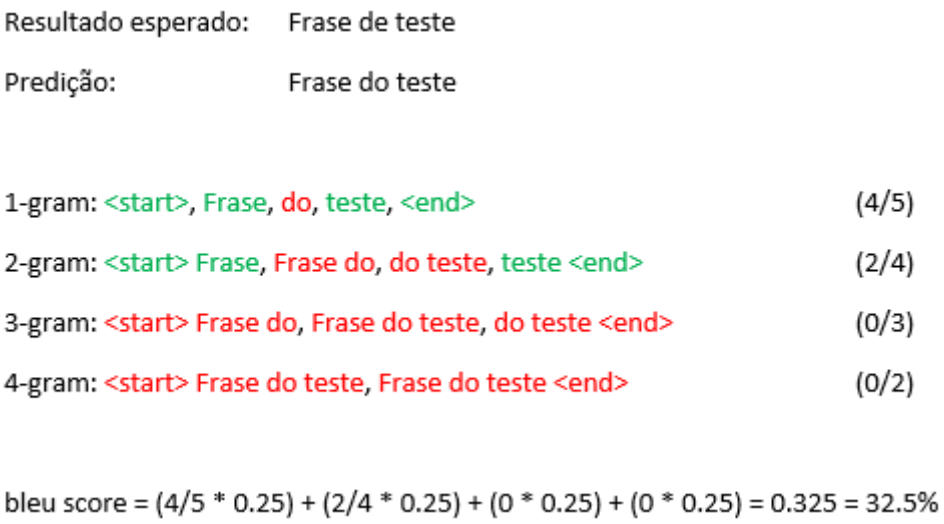


Figura 40.: Cálculo do `Bleu Score`.

Como o método `Bleu score` se foca essencialmente na comparação palavra a palavra da legenda, surgiu a necessidade de criar uma métrica que contabilizasse o número de componentes corretos e o *layout*.

A métrica desenvolvida (equação 28) decompõe-se em várias partes. A primeira parte da métrica compara os elementos gerados com os elementos esperados. Utilizando dicionários distintos, um para a imagem original e outro para a imagem gerada, dado o nome de um elemento e o número de ocorrências na imagem, pode verificar-se quantos dos elementos gerados estão no ficheiro original e obter assim o número de elementos corretos. Gerar elementos a mais ou a menos origina uma penalização.

Na verificação do *layout*, utilizam-se índices que acumulam o número da linha e da coluna. Iterativamente criam-se dois vetores de objetos, um para cada ficheiro, onde cada objeto contém o nome do componente, a linha e a coluna onde está posicionado na página Web. A verificação baseia-se na comparação dos objetos presentes no ficheiro original com os do ficheiro gerado, guardados nos dois vetores separadamente. Com a utilização de uma fila, é possível descobrir se a correspondência entre as chavetas está correta. Quando está presente uma chaveta de abertura, faz-se um *push*, enquanto que a chaveta de fecho resulta na ação contrária, o *pop*. Para que a correspondência entre chavetas seja correta, a fila tem que estar vazia no final.

O resultado final é dado pela percentagem de elementos corretos, pelo número de elementos posicionados na linha e coluna correta (com menor peso), e uma penalização associada à correspondência entre chavetas. A métrica ao focar-se nestes pontos, produz valores mais próximos da realidade no problema de geração automática de código. Para tornar os resultados ainda mais realistas, são atribuídas percentagens diferentes a cada um dos pontos enunciados anteriormente.

Visto que o número de elementos corretos tem grande relevância, atribui-se 80% do peso a esse contributo para a métrica. Os restantes 20% foram distribuídos em 10% para o número de elementos posicionados na linha correta e 10% para o número de elementos colocados na coluna correta. Devido à baixa probabilidade de o modelo gerar chavetas incorretas, existe uma penalização de 5% no resultado final quando a correspondência entre chavetas está errada. A variável binária chavetas_corretas indica se a correspondência entre chavetas está correta ou não (equação 27).

$$penalizacao = 0.95 + 0.05 \times \text{chavetas\_corretas} \tag{27}$$

A equação 28 apresenta a métrica utilizada. A equação permite avaliar a saída do modelo com base numa penalização, no número de elementos gerados corretamente e no tamanho e posicionamento dos elementos na página gerada.

$$penalizacao \times (0.8 \times \frac{\text{corretos}}{\max(\text{gerados, reais})} + 0.1 \times \frac{\text{linha\_correta}}{\text{reais}} + 0.1 \times \frac{\text{coluna\_correta}}{\text{reais}}) \tag{28}$$

A penalização é calculada com a equação 27, a variável *corretos* representa o número de elementos que o modelo gerou corretamente, *gerados* é o número total de elementos gerados pelo modelo, *reais* é o número de elementos que compõem o esboço original, a *linha_correta* e a *coluna_correta* indicam o número de elementos posicionados na linha e na coluna correta, respetivamente.

5.3 Abordagem 2 com YOLO e Algoritmo de Layout

A segunda abordagem adotada para converter os esboços para código resultou do facto de faltar na primeira abordagem (secção 5.2) um adequado tratamento da componente espacial dos esboços, o que dificultou imenso a geração de *layouts* corretos. Para combater esta necessidade utilizou-se um algoritmo que apresenta bons resultados na deteção e localização de objetos, o YOLO. Desta forma, pretende-se descobrir quais os objetos que estão na imagem fornecida pelo utilizador, como por exemplo, imagens, vídeos, ou botões. Além disso, é importante conhecer a localização dos objetos para que o *layout* final seja o mais parecido possível com o esperado. Um dos exemplos mais comuns de utilização futura da deteção de objetos será a condução autónoma. Neste caso é necessário detetar e localizar, em tempo real, pessoas, animais, sinais ou qualquer obstáculo que cause perigo, no menor tempo possível.

Com base na informação obtida a partir do YOLO, desenvolveu-se um algoritmo de *layout* que mapeia os objetos detetados na imagem fornecida em código HTML e CSS. O algoritmo de *layout* aproveita as propriedades de posicionamento e tamanho do CSS para colocar os elementos na página Web. Deste modo, é possível colocar cada objeto numa posição muito próxima daquela que se encontra na imagem original. A utilização dos pontos fornecidos pelo YOLO, permite deduzir a largura e a altura dos objetos detetados e, por conseguinte, obter páginas HTML com aspeto gráfico muito próximo da imagem fornecida inicialmente.

5.3.1 *YOLO V1*

A deteção de objetos é uma área computacional relacionada com a visão por computador e o processamento de imagem. É utilizada na deteção e classificação de objetos em imagens e vídeos. O YOLO é um modelo de aprendizagem profunda para deteção e localização de objetos em tempo real. Neste momento vai na terceira versão, sendo que as duas últimas são melhorias da primeira versão. Com uma única rede neuronal, prevê probabilidades para as classes de objetos e identifica as suas regiões delimitadoras, recorrendo a uma única passagem sobre a imagem.

O YOLO é um sistema extremamente rápido e que apresenta mais erros de localização, mas menor quantidade de falsos positivos. O modelo recebe como entrada uma imagem

e divide-a numa grelha $N \times N$. Cada célula da grelha prevê um único objeto, ou seja, no máximo podem ser detetados $N \times N$ objetos (figura 41).

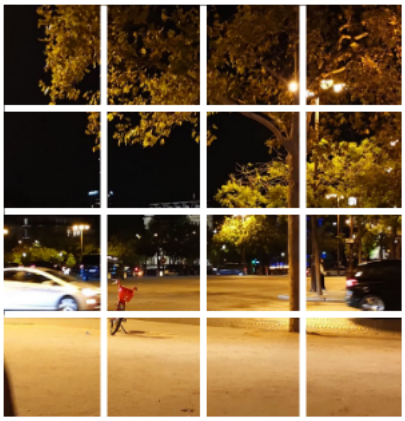


Figura 41.: Imagem dividida numa grelha 4×4.

Em cada célula da grelha prevê-se um número máximo de RD regiões delimitadoras, pelo que a quantidade destas é dada por $N \times N \times RD$. Cada região delimitadora é composta por 4 valores (x, y, w, h) e uma pontuação de confiança. Quando o objeto está presente na região delimitadora, a pontuação resulta do valor da Intersection over Union (IoU) entre a região delimitadora real e a prevista. Caso contrário, a pontuação é nula. Para cada célula da grelha, o YOLO prevê uma probabilidade por categoria. Portanto, a saída tem a forma $N \times N \times (RD \times 5 + NC)$, onde $N \times N$ representa a grelha, RD o número de regiões delimitadoras e NC o número de categorias. Cada célula deteta no máximo um objeto, o que em certos casos pode limitar os resultados obtidos, visto que na mesma célula podem coexistir mais do que um objeto.

Formalmente define-se a pontuação de confiança da região delimitadora com a equação 29, onde $P(objeto)$ é a probabilidade da região delimitadora conter o objeto e $IoU(prev, real)$ é a intersecção sobre a união entre a região delimitadora prevista e real. Quando o objeto não está presente na região, a pontuação de confiança é nula. Caso contrário, o valor da confiança é dado por $IoU(prev, real)$. Deste modo, a confiança reflete a presença de um objeto de qualquer classe.

$$C = P(objeto) \times IoU(prev, real) \tag{29}$$

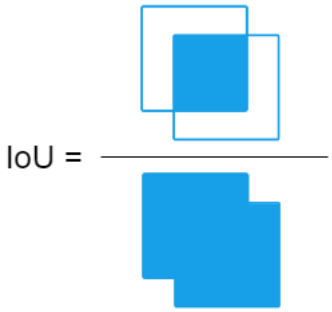


Figura 42.: Ilustração da métrica *interseção sobre a união (IoU)*.

A figura 42 ilustra o significado da métrica IoU. Supondo que existem duas regiões delimitadoras, a região prevista e a região real, a métrica resulta da divisão entre a área de intersecção (área comum) e a área total da união das regiões delimitadoras. Dada a presença de um determinado objeto, a probabilidade condicional da classe $P(classe_i|objeto)$ indica a probabilidade desse objeto pertencer à `classe i`. A pontuação de confiança de uma classe, resulta da multiplicação de C por $P(classe_i|objeto)$ (equação 31).

$$p(i) = P(classe_i|objeto) \tag{30}$$

$$\begin{aligned} pont\_conf\_classe(i) &= C \times P(classe_i|objeto) = \\ &= P(objeto) \times IoU(prev, real) \times P(classe_i|objeto) \end{aligned} \tag{31}$$

O YOLO v1 é constituído por 24 camadas convolucionais e duas camadas densamente ligadas no final. Para obter conformidade na saída, é necessário que o tensor seja reduzido a um tensor com *rank* 1 (*flatenned*) e tamanho $N \times N \times (RD \times 5 + NC)$.

As previsões são ordenadas segundo a pontuação de confiança. Para evitar a existência de múltiplas deteções do mesmo objeto, aplica-se *non-maximal suppression* com intuito de remover deteções repetidas que apresentem menor pontuação de confiança. As regiões delimitadoras que apresentem IoU inferior a 0.5 são ignoradas.

A função de custo utilizada divide-se em três partes: o custo da classificação, o custo da localização e a pontuação de confiança. O erro resultante da classificação está relacionado com a presença do objeto na célula. Quando o objeto não está na célula, o erro é nulo. Quando se verifica o contrário, ou seja, o objeto está presente na região, o erro resulta da soma dos quadrados da diferença entre a probabilidade condicional da classe real e estimada, para todas as classes (equação 32).

$$\sum_{i=0}^{N^2} 1_i^{objeto} \sum_{c \in classes} (p_i(c) - \widehat{p}_i(c))^2 \tag{32}$$

Na equação anterior, N^2 representa o número de células da grelha, 1_i^{objeto} é um valor binário (que é 1 quando o objeto está na célula i, 0 caso contrário) e $p_i(c)$ é a probabilidade condicional da classe c (equação 30).

O erro de localização mede a diferença entre as regiões delimitadoras previstas e reais. De acordo com a equação 33 é considerada a posição e o tamanho da região delimitadora.

$$\sum_{i=0}^{N^2} \sum_{j=0}^{RD} 1_i^{objeto} \left[(x_i - \widehat{x}_i)^2 + (y_i - \widehat{y}_i)^2 \right] + \sum_{i=0}^{N^2} \sum_{j=0}^{RD} 1_i^{objeto} \left[(\sqrt{w_i} - \sqrt{\widehat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\widehat{h}_i})^2 \right] \tag{33}$$

Na equação 33, RD é o número de regiões delimitadoras, x e y formam o ponto que indica a localização da região, w e h denotam a largura e a altura da região delimitadora, respetivamente. Aplica-se a raiz quadrada, à largura e à altura, para evitar que os erros absolutos não sejam ponderados da mesma forma em regiões com tamanhos diferentes. Deste modo, os erros são mais penalizadores em regiões pequenas do que em regiões maiores. O erro resultante da pontuação de confiança (equação 29) é dado pela equação 34.

$$\sum_{i=0}^{N^2} \sum_{j=0}^{RD} 1_{ij}^{objeto} (C_i - \widehat{C}_i)^2 \tag{34}$$

Onde a variável C_i representa a pontuação de confiança.

É mais frequente as regiões não conterem objetos do que conterem, pelo que o modelo é treinado mais frequentemente para encontrar a textura de fundo do que objetos. Posto isto, cria-se um problema de desbalanceamento. Para combater este problema multiplica-se a equação 34 por um fator $\lambda_{semObjeto}$, de valor 0.5 por omissão, para reduzir o peso do erro (equação 35).

$$\lambda_{semObjeto} \sum_{i=0}^{N^2} \sum_{j=0}^{RD} 1_{ij}^{semObjeto} (C_i - \widehat{C}_i)^2 \tag{35}$$

A expressão do custo total resulta da soma das equações 32, 33, 34 e 35 (equação 36).

$$
\begin{aligned}
custo\_total = & \sum_{i=0}^{N^2} 1_i^{objeto} \sum_{c \in classes} (p_i(c) - \widehat{p}_i(c))^2 \\
& + \sum_{i=0}^{N^2} \sum_{j=0}^{RD} 1_i^{objeto} \left[(x_i - \widehat{x}_i)^2 + (y_i - \widehat{y}_i)^2 \right] \\
& + \sum_{i=0}^{N^2} \sum_{j=0}^{RD} 1_i^{objeto} \left[(\sqrt{w_i} - \sqrt{\widehat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\widehat{h}_i})^2 \right] \\
& + \sum_{i=0}^{N^2} \sum_{j=0}^{RD} 1_{ij}^{objeto} (C_i - \widehat{C}_i)^2 \\
& + \lambda_{semObjeto} \sum_{i=0}^{N^2} \sum_{j=0}^{RD} 1_{ij}^{semObjeto} (C_i - \widehat{C}_i)^2
\end{aligned}
\tag{36}
$$

O YOLO é diferente de todos os outros métodos que tratam o problema da deteção de objetos. A resolução de todo o problema com uma única CNN oferece os seguintes benefícios:

Maior rapidez:
É extremamente rápido quando comparado com os seus antecessores, pois utiliza uma única CNN. As previsões são realizadas com apenas uma passagem pela rede. Deste modo, torna-se uma ferramenta muito útil para processamento em tempo real.

Menos falhas associadas à textura de fundo:
Analisa imagens inteiras durante a fase de treino e de predição, o que implica que seja guardada informação sobre as classes e, consequentemente, sobre a aparência de cada classe. Outras abordagens apresentam vários erros associados à textura de fundo, pois não observam contextos alargados nas entradas. Por este motivo, o YOLO apresenta uma menor quantidade de erros quando comparado com estas abordagens.

Capacidade de generalização:
Aprende a generalizar representações de objetos. Devido à sua elevada capacidade de generalizar, o YOLO apresenta melhores resultados quando confrontado com entradas inesperadas ou ligeiramente diferentes das utilizadas durante o treino. As abordagens anteriores apresentam mais limitações quando sujeitas a imagens ligeiramente diferentes daquelas que foram utilizadas durante a fase de treino do modelo.

5.3.2 *YOLO V2*

A segunda versão do YOLO introduziu melhorias de precisão e desempenho. Nesta versão, introduziram camadas de normalização de *batch*. A normalização diminui a variação dos valores das camadas intermédias, o que melhora a estabilidade da rede. Estas camadas ajudam a reduzir o *overfitting* em geral e, consequentemente, aumentam a precisão do modelo. De acordo com a informação disponibilizada em Redmon and Farhadi (2016), as camadas de *dropout* deixaram de ser necessárias ao modelo para evitar o *overfitting*.

A mudança para um classificador com maior resolução, em que o tamanho da imagem de entrada passou de 224×224 para 448×448, contribuiu para a melhoria de precisão nesta versão. O treino é composto por duas fases. Inicialmente treina-se um classificador como uma rede VGG16 e imagens de 224×224. Posteriormente, efetua-se a troca das camadas densamente ligadas (presentes no final do classificador) por convoluções e o modelo é treinado para detetar objetos em imagens de 448×448. A mudança facilita a aprendizagem e aumenta a precisão do modelo.

A utilização de um conjunto de regiões delimitadoras pré-definidas, permitiu remover todas as camadas densamente ligadas existentes na versão 1. Estas regiões são definidas para capturar a escala e a proporção dos objetos presentes no conjunto de dados, pelo que devem ser tomadas medidas que facilitem a sua deteção. Segundo Redmon and Farhadi (2018), o YOLO v1 numa fase inicial do treino é suscetível a gradientes instáveis. Isto deve-se principalmente à realização aleatória das previsões das regiões delimitadoras, o que pode resultar apenas em certos casos. Deste modo, é importante que no início do treino sejam definidas as formas mais comuns do conjunto de dados. Em vez de utilizar regiões delimitadoras aleatórias, prevê-se apenas compensações que ajustam cada região. As compensações permitem ampliar ou reduzir o tamanho da região e adaptar-se desta forma ao objeto. As regiões são restringidas para manter a diversidade de formas e tornar o treino mais estável.

O YOLO estima 5 valores, t_x, t_y, t_w, t_h e t_o, que estão associados à previsão inicial do YOLO. t_x e t_y representam o ponto central e, t_w e t_h a largura e a altura da região delimitadora, respetivamente. O último valor (t_o) representa a pontuação de confiança da região. Como foi referido anteriormente, a imagem está dividida numa grelha com $N \times N$ células. Nas equações 37 e 41, a função σ restringe os intervalos de deslocamento, c_x e c_y são o vértice superior esquerdo da célula que está a ser avaliada, p_w e p_h são a largura e altura da região delimitadora pré-definida (ancora). As variáveis b representam a região delimitadora sujeita aos ajustes de posição e tamanho. Deste modo, b_x, b_y, são o ponto central, b_w a largura e b_h a altura da região delimitadora prevista como resultado final.

$$b_x = \sigma(t_x) + c_x \tag{37}$$

$$b_y = \sigma(t_y) + c_y \tag{38}$$

$$b_w = p_w e^{t_w} \tag{39}$$

$$b_h = p_h e^{t_h} \tag{40}$$

$$P_r(objeto) * IoU(b, objeto) = \sigma(t_o) \tag{41}$$

A figura 43 representa graficamente o significado das equações 37 a 40. A região a tracejado, representada pela variável p, corresponde à região delimitadora prevista inicialmente. Após o cálculo das compensações, obtém-se a região delimitadora representada pela variável b (linha contínua). Deste modo, as regiões delimitadoras mantêm a diversidade de formas, mas adaptam-se aos objetos.

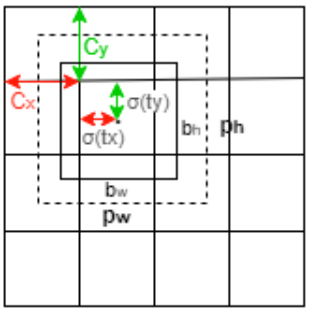


Figura 43.: Representação das compensações no cálculo de regiões delimitadoras.

5.3.3 *YOLO V3*

A terceira versão do YOLO introduziu alterações na arquitetura do modelo, resultando numa rede maior e mais precisa. Apesar do crescimento, o modelo contínua a permitir a deteção de objetos em tempo real. Esta versão aplica classificação *multi-label* na deteção dos objetos. Inicialmente, o YOLO utilizava uma camada *softmax* para realizar a previsão e encontrar a classe com maior probabilidade. Com a função `softmax`, a soma das probabilidades das classes tem de ser obrigatoriamente igual a 1. Com esta alteração a suposição de

que as classes são mutuamente exclusivas deixa de existir, ou seja, o objeto presente na imagem não pertence exclusivamente a uma classe. Por exemplo, quando existem classes como animal e gato, o objeto não pertence exclusivamente a uma classe, mas a ambas. Apenas prevalecem pontuações superiores a um determinado nível de decisão.

O YOLO V3 introduziu uma rede nova na extração de caraterísticas, constituída por camadas com convoluções 3×3 e 1×1 sucessivas, com atalhos e significativamente maior do que a anterior. Os atalhos são ligações diretas entre camadas não sucessivas, uma ideia introduzida pela arquitetura `Resnet` (He et al., 2015). A rede é constituída por 53 camadas convolucionais e, por isso, é conhecida por `Darknet-53`. Esta rede tem muito mais potencial que a anterior *Darknet-19*, mantendo uma eficiência computacional superior à `Resnet-101` e `Resnet-152`. Apesar do maior número de camadas, a `Resnet` é menos eficiente e realiza menos operações de vírgula flutuante por segundo. Portanto, tiram menor partido dos GPUs.

Nesta segunda abordagem da presente dissertação, escolheu-se a versão 3 do YOLO para desenvolver o modelo de deteção de objetos. Os bons resultados deste modelo com conjuntos de dados pequenos e a capacidade de funcionar em tempo real são pontos a favor, visto que o modelo suporta uma aplicação Web cuja alta escalabilidade é muito importante.

5.3.4 *Algoritmo de Layout*

O YOLO deteta objetos, no nosso caso elementos HTML, e estima a sua localização aproximada. Através das propriedades do CSS, os elementos podem ser posicionados por coordenada (secção 3.3). Desta forma, é relativamente fácil mapear objetos reconhecidos pelo modelo em código HTML. O modelo de aprendizagem profunda gera dois pontos que definem a região delimitadora de um objeto (x_{min}, y_{min}) e (x_{max}, y_{max}). Na ausência de informação mais precisa, utiliza-se a informação da região delimitadora como localização do objeto.

As propriedades *top* e *left* do CSS permitem colocar o objeto na posição (x_{min}, y_{min}), tal como a largura e altura são definidas com as propriedades *width* e *height*. A utilização destas propriedades permite gerar páginas Web graficamente muito parecidas às imagens fornecidas como entrada. Uma desvantagem desta abordagem é a menor responsividade em *smartphones*. Para garantir que a largura e a altura se adaptam ao tamanho do dispositivo, utiliza-se a função *calc*. Esta função é utilizada em cálculos simples e flexibiliza o uso das propriedades mencionadas. Suporta expressões matemáticas de adição (+), subtração (-), multiplicação ($\times$) e divisão ($\div$). A *viewport* é a área onde o *browser* desenha o conteúdo da página. Recentemente, apareceram 4 novas unidades para a *viewport*, Viewport Width (vw), Viewport Height (vh), Viewport Minimum (vmin) e Viewport Maximum (vmax), que funcionam de forma semelhante ao % usado em CSS para definir tamanhos.

As imagens são convertidas para um tamanho de 256×256 à entrada do modelo, tanto em treino como em predição. Resulta daqui a necessidade de mapear as coordenadas e o tamanho do objeto para a medida da *viewport*. No cálculo das propriedades *left*, *top*, *width* e *height*, as equações 42 a 45 convertem o tamanho da imagem (256×256) para o tamanho da *viewport*, que varia conforme o dispositivo em que está a ser utilizado.

$$left = (x_{min} \times 100vw) \div 256 \tag{42}$$

$$top = (y_{min} \times 100vh) \div 256 \tag{43}$$

$$width = (width \times 100vw) \div 256 \tag{44}$$

$$height = (height \times 100vh) \div 256 \tag{45}$$

Nas equações anteriores, (x_{min}, y_{min}) é o ponto superior esquerdo da região delimitadora e *width* e *height* a largura e altura, respetivamente. A largura e altura deduzem-se a partir dos pontos (x_{min}, y_{min}) e (x_{max}, y_{max}), através da subtração de pontos. As equações recorrem à função `calc`, dado que os pontos (x_{min}, y_{min}) e (x_{max}, y_{max}) variam de imagem para imagem, o que obriga o código a adaptar-se à imagem de entrada. Estas propriedades permitem fazer a colocação dos elementos HTML exatamente nas posições geradas pelo modelo.

Para melhorar o funcionamento do algoritmo foi criada uma funcionalidade que permite detetar sobreposições de regiões delimitadoras. Considerou-se haver sobreposição sempre que a área sobreposta é superior a metade da área da menor região. Quando isto acontece apenas pode prevalecer um dos objetos, aquele cuja pontuação de confiança é superior.

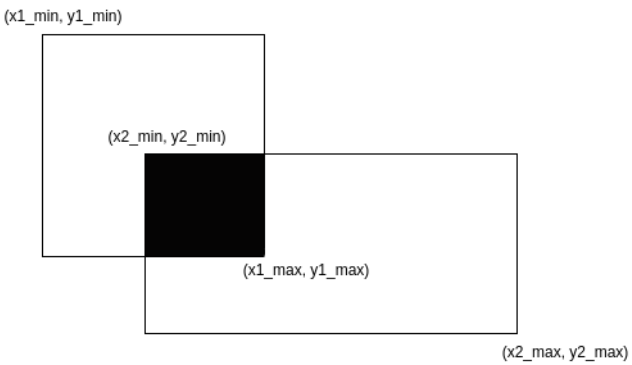


Figura 44.: Interseção de dois retângulos.

Perante a existência de dois retângulos, as medidas dos lados da interseção obtém-se com as equações 46 e 47. Destas equações resulta o comprimento dos lados da intersecção e, consequentemente, a área de intersecção ($dx \times dy$).

$$dx = min(x1_{max}, x2_{max}) - max(x1_{min}, x2_{min}) \tag{46}$$

$$dy = min(y1_{max}, y2_{max}) - max(y1_{min}, y2_{min}) \tag{47}$$

Para cada elemento HTML foi definido um *template* com código HTML e CSS. O código dos *templates* possuí etiquetas que indicam as zonas substituíveis pelos valores gerados no YOLO. Considera-se como substituível as propriedades referentes à posição e ao tamanho no código CSS dos elementos HTML. Assim, garante-se que a colocação, no ficheiro de saída, dos elementos gerados pelo YOLO, bem com as devidas propriedades, está de acordo com o pretendido.

Os ficheiros HTML seguem uma estrutura bem definida, o que impossibilita a escrita desordenada no ficheiro. Após efetuar as substituições, o código dos elementos é guardado em *strings*. No final são escritas no ficheiro de forma ordenada para garantir a estrutura correta do código e resultarem ficheiros de código funcional.

5.3.5 *Arquitetura do Modelo*

Tal como aconteceu na primeira abordagem, o modelo desenvolvido apresenta arquiteturas diferentes durante as fases de treino e de predição. Durante o **treino** o YOLO recebe um ficheiro com a identificação das imagens, os objetos de cada imagem e as respetivas regiões delimitadoras (figura 45). Além da identificação da imagem, o modelo recebe a seguinte informação para cada objeto: $x_{min}, y_{min}, x_{max}, y_{max}$ e a classe do objeto. Cada imagem pode conter um ou mais objetos. No final do treino os pesos do modelo treinado são guardados num ficheiro, para serem utilizados na fase de predição.

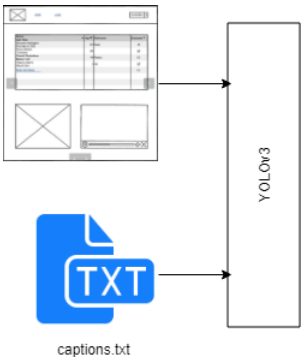


Figura 45.: Arquitetura do modelo da abordagem 2 em treino.

O modelo desenvolvido utiliza a versão 3 do YOLO. O único hiperparâmetro ajustado foi a taxa de aprendizagem, tendo-se usado o valor $1 \times e^{-6}$, e o otimizador selecionado foi o `Adam`. Os hiperparâmetros que definem a capacidade da rede, como o número de camadas ou o número de neurónios por camada, não foram alterados.

Durante a **fase de predição**, o modelo recebe apenas uma imagem para a qual deve gerar código HTML e CSS (figura 46). Tendo em conta que o modelo apenas prevê as regiões delimitadoras e a classe a que o objeto pertence, surge a necessidade de utilizar o algoritmo de *layout* para converter a saída do YOLO para código (subsecção 5.3.4). O algoritmo de *layout* recebe a lista de regiões delimitadoras geradas pelo YOLO, que contém a informação da localização e a classe do objeto ($x_{min}, y_{min}, x_{max}, y_{max}$, classe), e o ficheiro com os *templates* de elementos HTML. O algoritmo converte os objetos para código HTML e CSS.

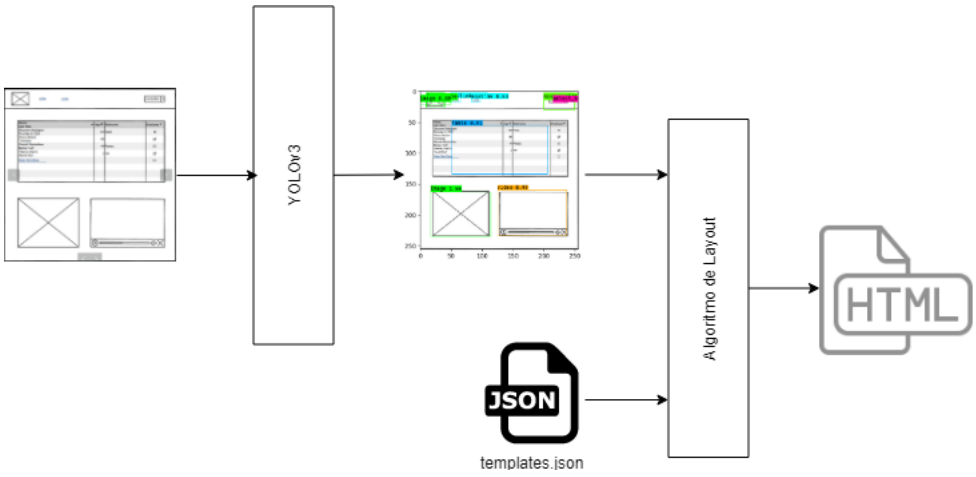


Figura 46.: Arquitetura do modelo da abordagem 2 na fase de predição.

No final, obtém-se sempre um ficheiro com código funcional. Para simplificar o manuseamento do resultado final por parte dos utilizadores, os códigos HTML, CSS e `Bootstrap` são colocados num único ficheiro HTML. O apêndice A contém um exemplo do código funcional gerado pelo modelo. O código desenvolvido está disponível no seguinte URL: `https://github.com/tiagoboucas/mockups2html_code/tree/master/model_2`

5.3.6 *Métrica de Avaliação*

A métrica desenvolvida para a segunda abordagem permite avaliar a qualidade do ficheiro HTML gerado. A qualidade diz respeito à correspondência entre a imagem fornecida como entrada e a página HTML e CSS gerada pelo modelo como saída. A métrica baseia-se no número de elementos corretos e nas interseções das regiões delimitadoras estimadas com as reais.

Com o valor da IoU das regiões delimitadoras (figura 42) encontra-se a percentagem de coincidência entre as áreas e, deste modo, é possível penalizar o erro na geração de regiões delimitadoras.

Procurou-se que as métricas das duas abordagens fossem semelhantes. Para isso, tal como na abordagem 1, atribuiu-se um peso de 80% ao número de elementos corretos e 20% à localização dos objetos (equação 49). Sabendo que as áreas da região delimitadora estimada e real se intersetam, garante-se através da IoU que os erros de tamanho e posicionamento serão contabilizados na métrica final.

$$ious = \frac{\sum_{i=0}^{N} IoU_i}{N} \tag{48}$$

A equação 48 fornece a média das IoUs das N regiões delimitadoras reais. O algoritmo compara as regiões delimitadoras reais com a estimadas e, percebe quais das regiões estimadas correspondem às regiões reais. A comparação realiza-se com base na categoria e IoU das regiões.

$$0.8 \times \frac{obj\_corretos}{max(obj\_reais, obj\_pred)} + 0.2 \times ious \tag{49}$$

Na equação anterior o valor dado pela função *max* penaliza tanto os objetos gerados a mais, como os objetos gerados a menos.

5.4 DESENVOLVIMENTO DA APLICAÇÃO WEB

Inicialmente utilizou-se o comando `create-react-app` para criar uma aplicação de página única (SPA). Esta abordagem oferece uma instalação moderna e prática sem a necessidade de qualquer configuração. O comando `create-react-app` cria uma aplicação em React com o *template* do Facebook e os recursos de compilação. A estrutura da pasta gerada é ilustrada como na figura 47.

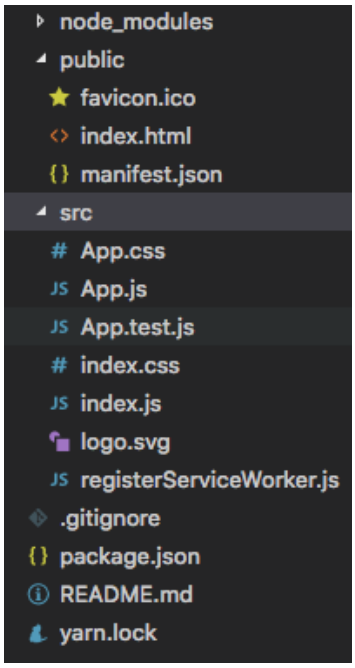


Figura 47.: Ficheiros resultantes do comando `create-react-app`.

ReactJS utiliza JSX, uma extensão da sintaxe do JavaScript com etiquetas muito semelhantes ao HTML. Esta extensão combina HTML com JavaScript, ou seja, incorpora as etiquetas HTML diretamente no código JavaScript. Pode-se utilizar apenas JavaScript, mas JSX permite código mais simples e legível.

O `redux` não se restringe a ReactJS. É possível escrever aplicações com `redux` em Angular, Vue.js, jQuery ou JavaScript. O `redux` funciona como estado da interface, que emite atualizações de estado em resposta a ações. Dada a simplicidade da aplicação, a manutenção de estado na aplicação resume-se a guardar o código gerado pelo modelo ou possíveis alterações realizadas pelo utilizador. Deste modo, o utilizador pode navegar pelas páginas sem a preocupação de guardar o código gerado, uma vez que ele fica guardado no editor tal e qual como o deixou.

ReactJs introduziu o conceito da arquitetura baseada em componentes, um método que encapsula peças individuais de larga interface em microssistemas autossustentáveis. Os componentes podem ser vistos como pequenas peças independentes, que em conjunto constituem a interface do utilizador. São utilizados no mesmo espaço, mas interagem independentemente. Possuem estrutura, métodos e API própria. A utilização desta arquitetura facilita a reutilização de código, na medida em que os componentes são reutilizáveis e fáceis de introduzir em interfaces para as quais não foram criados.

A interface da aplicação Web divide-se em 4 componentes: o cabeçalho, uma *frameview* que mostra as alterações no código realizadas através do editor em tempo real, um cartão em que se descrevem os passos a realizar ao carregar uma imagem na página principal e

o `codemirror`, um componente externo utilizado como editor de texto. A aplicação é constituída apenas por 2 páginas, a página principal e o editor de código. As páginas seguem muito de perto o protótipo apresentado na seção 4.3. A página principal é constituída por um componente `header` e 3 cartões (componente `card`), que funcionam como tutorial sobre a utilização da aplicação.

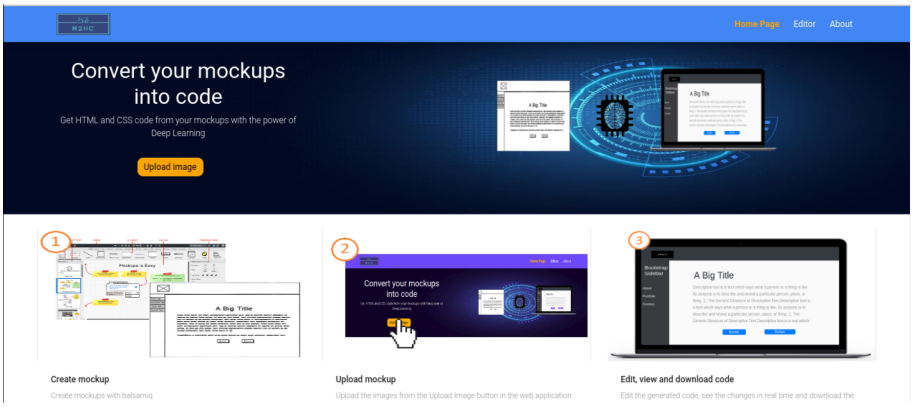


Figura 48.: Página principal desenvolvida em ReactJS.

A página principal desenvolvida é muito semelhante ao protótipo de alta fidelidade proposto inicialmente. A página do editor de código acabou por sofrer alterações à posição dos botões que permitem descarregar o código original e o editado. A posição central dos botões, inicialmente colocados à esquerda, melhora o aspeto global da página. O editor é composto pelo `header`, um editor de texto (`codemirror`) e uma `frameview`. O utilizador pode criar, editar ou remover conteúdo do editor de texto e, automaticamente, visualizar as alterações no componente *frameview*. Esta funcionalidade é realmente interessante, na medida em que poupa tempo aos utilizadores, pois evita a abertura de páginas HTML para visualizar o conteúdo do ficheiro.

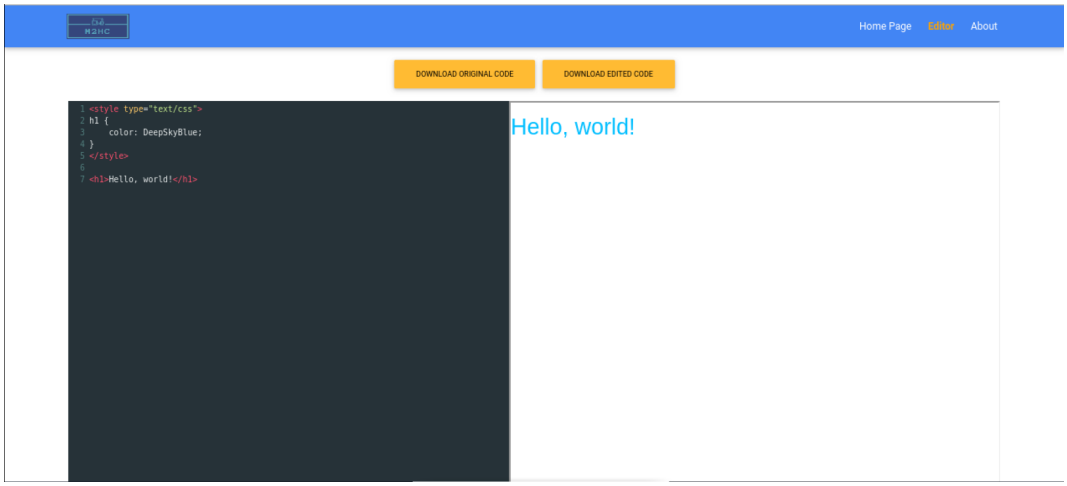


Figura 49.: Editor de código desenvolvido em ReactJS.

Para terminar, a gestão de páginas foi realizada com `React Router`, uma simples API que mantém a interface sincronizada com os URLs. Os recursos disponibilizados pela API são poderosos, caso da correspondência de rotas dinâmicas. Definiram-se apenas duas rotas, uma para a página principal (`'/'`) e outra para o editor (`'/editor'`), devido à simplicidade da aplicação Web.

6

EXPERIÊNCIAS E RESULTADOS

No presente capítulo apresentam-se as experiências realizadas e os resultados obtidos no desenvolvimento da dissertação. As experiências incluem o teste das diferentes redes neuronais convolucionais (VGG16, VGG19, ResNet-50, etc) na extração de caraterísticas, as diferenças da utilização de GRUs ou LSTMs, a redução do tempo de treino com RNNs otimizadas com Compute Unified Device Architecture (CUDA) (via `CuDNN`) e a comparação dos resultados obtidos entre as duas abordagens.

As duas abordagens utilizam o mesmo conjunto de dados e cenários de teste iguais, de forma a possibilitar comparações adequadas. O conjunto de dados nas duas abordagens difere apenas na legendagem das imagens. Na primeira abordagem fornecem-se ficheiros com código DSL como entrada das RNNs, enquanto que na segunda abordagem utiliza-se a descrição e localização das regiões delimitadoras dos objetos. O conjunto de dados foi dividido em 90% para treino e 10% para teste.

6.1 EXPERIÊNCIAS COM A ABORDAGEM 1

A abordagem 1 implementou um método muito semelhante à legendagem de imagens. No nosso caso, a legendagem realiza-se com código DSL que descreve a imagem, ao invés da habitual legenda em língua natural, por exemplo em Português. Para facilitar a tarefa das redes neuronais e obter melhores resultados, optou-se pela utilização de uma DSL, visto que usar código HTML e CSS dificultaria imenso o código gerado pelo modelo.

Com as experiências realizadas pretendia-se selecionar a melhor combinação de redes neuronais, com base na pontuação de semelhança entre os esboços reais e os gerados pelo modelo no subconjunto de teste. A melhor combinação será utilizada na comparação com a segunda abordagem. As várias alternativas envolvem a utilização de diferentes CNNs, RNNs e a utilização de RNNs otimizadas com CUDA. Importava comparar o tempo de treino e a semelhança obtida no subconjunto de teste.

A tabela 2 contém os resultados das várias experiências realizadas com diferentes CNNs e RNNs. A pontuação de semelhança entre os esboços de teste reais e gerados foram recolhidos com a utilização da métrica desenvolvida na subsecção 5.2.5.

CNN *Encoder*	**RNN *Encoder***	**RNN *Decoder***	**Pontuação (%)**
InceptionResNetV2	CuDNNGRU	CuDNNGRU	12.73%
InceptionV3	CuDNNGRU	CuDNNGRU	32.66%
ResNet-50	CuDNNGRU	CuDNNGRU	34.75%
VGG16	CuDNNGRU	CuDNNLSTM	60.36%
VGG16	CuDNNLSTM	CuDNNLSTM	61.97%
VGG16	GRU	GRU	66.73%
VGG16	CuDNNGRU	CuDNNGRU	67.35%
VGG19	CuDNNGRU	CuDNNGRU	71.30%

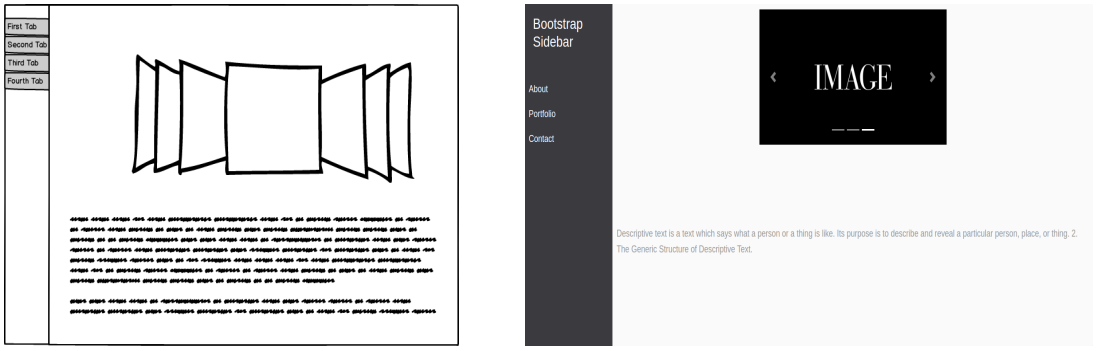
Tabela 2.: Sumário das experiências realizadas na abordagem 1.

A tabela 3 contém as experiências realizadas com um número de iterações (*epochs*) fixado em 20. Estas permitem comparar o tempo de treino sujeito à variação da composição das redes (CNN e RNNs).

CNN *Encoder*	**RNN *Encoder***	**RNN *Decoder***	**Tempo de treino**	**Pontuação (%)**
InceptionV3	CuDNNGRU	CuDNNGRU	239 min	26.16%
VGG16	CuDNNGRU	CuDNNGRU	346 min.	60.27%
ResNet-50	CuDNNGRU	CuDNNGRU	373 min.	20.43%
VGG16	CuDNNLSTM	CuDNNLSTM	373 min.	26.21%
VGG16	CuDNNGRU	CuDNNLSTM	374 min.	56.15%
VGG19	CuDNNGRU	CuDNNGRU	423 min.	63.12%
VGG19	CuDNNGRU	CuDNNLSTM	429 min.	59.09%
VGG16	GRU	GRU	565 min.	63.26%

Tabela 3.: Sumário das experiências com 20 iterações na abordagem 1.

Apresentam-se a seguir alguns exemplos de predição realizados com o modelo desenvolvido nesta abordagem, com o objetivo de avaliar visualmente a qualidade dos esboços gerados. Os exemplos reproduzidos permitem avaliar a maioria dos tipos de elemento presentes no conjunto de dados. Na figura 50, a entrada contém 3 dos elementos mais utilizados em aplicações Web: a barra de navegação lateral, o *slideshow* e texto. Percebe-se pela saída que o modelo não tem em consideração o posicionamento dos elementos e, portanto, o *slideshow* é colocado no ficheiro HTML sem margem superior.



(a) Entrada fornecida (b) Respetiva saída

Figura 50.: Primeiro exemplo de predição na abordagem 1.

Como se pode constatar pela análise da figura 51, o resultado obtido neste exemplo é bastante semelhante à entrada fornecida. A principal diferença está no tamanho e posicionamento dos elementos. Os elementos definidos no ficheiro de *templates* são colocados na página HTML segundo o modelo por linha e coluna, com tamanho e posição pré-definida.

(a) Entrada fornecida (b) Respetiva saída

Figura 51.: Segundo exemplo de predição na abordagem 1.

No exemplo reproduzido na figura 52, a página gerada também reproduz corretamente a maioria dos elementos presentes no esboço de entrada. Tal como acontece em todos os exemplos, o tamanho e a posição dos elementos está definido por omissão no ficheiro de *templates*. Percebe-se assim, porque é que o tamanho está errado e faltam as margens em torno dos elementos HTML, como acontece por exemplo na tabela e na imagem da figura 52.

(a) Entrada fornecida (b) Respetiva saída

Figura 52.: Terceiro exemplo de predição na abordagem 1.

6.2 EXPERIÊNCIAS COM A ABORDAGEM 2

Com a segunda abordagem pretendia-se verificar se uma arquitetura, conhecida por ter um elevado desempenho na deteção e localização de objetos, teria melhor desempenho que a arquitetura utilizada na abordagem 1 na conversão de esboços para código. A arquitetura em causa é o YOLO.

Em cada iteração do treino e da validação com YOLO, obtém-se um valor para o custo (*loss*). O valor do custo quantifica quão o modelo se está a adaptar aos dois subconjuntos de dados e ao contrário da precisão, o valor não é uma percentagem. Como a função de custo resulta da soma dos erros, pretende-se minimizar o seu valor. Os gráficos seguintes mostram o valor do custo no subconjunto de dados de treino (*loss*) e de validação (*validation loss*) ao longo das várias iterações (*epochs*).

Os valores do custo variam muito de uma iteração para a seguinte, o que dificulta a visualização das curvas. Para contornar este problema, decidiu dividir-se o gráfico do custo em dois. O gráfico da figura 53 contém as primeiras 15 iterações. O intervalo de valores é bastante grande, mas permite visualizar a descida do erro e a ausência de interseções entre as curvas das sessões de treino e de validação.

O gráfico da figura 54 possuí um intervalo de valores de custo muito inferior ao gráfico da figura 53 e, por conseguinte, permite visualizar o comportamento das curvas com mais detalhe. Desta forma, é possível perceber o momento em que as curvas se intersetam, ou seja, quando começam a divergir. Este momento é importante para perceber quando o treino deve ser interrompido.

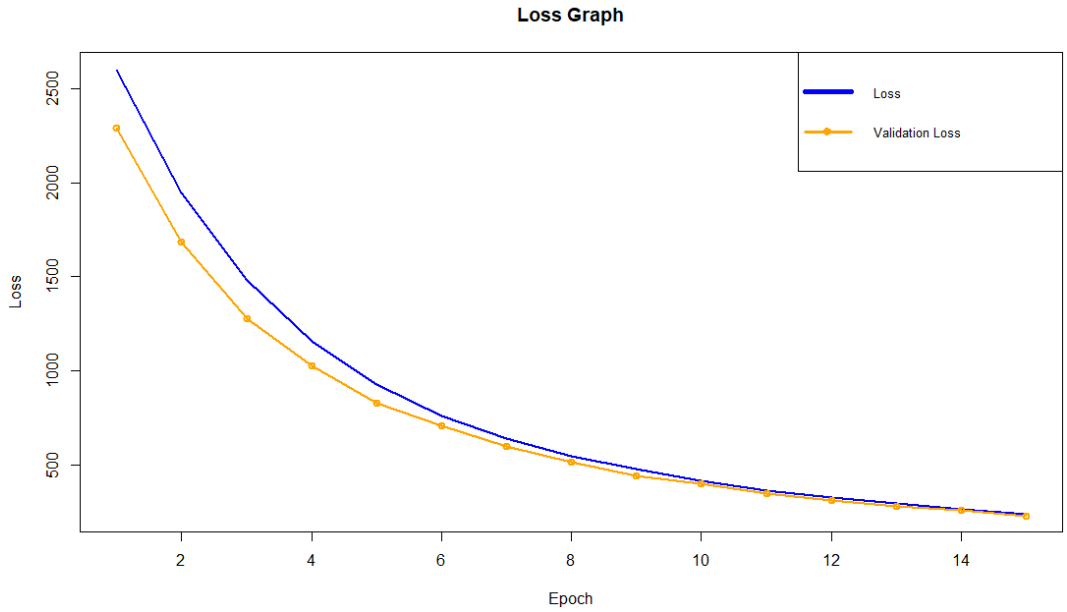


Figura 53.: Valor do erro nas primeiras 15 iterações (grande intervalo de variação).

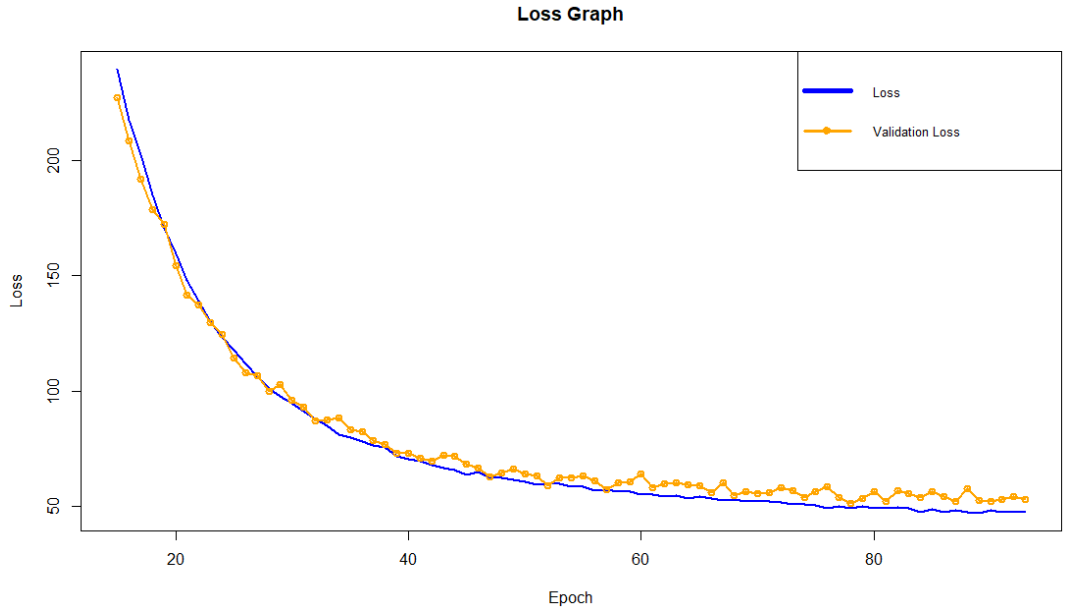


Figura 54.: Valor do erro nas restantes iterações (pequeno intervalo de variação).

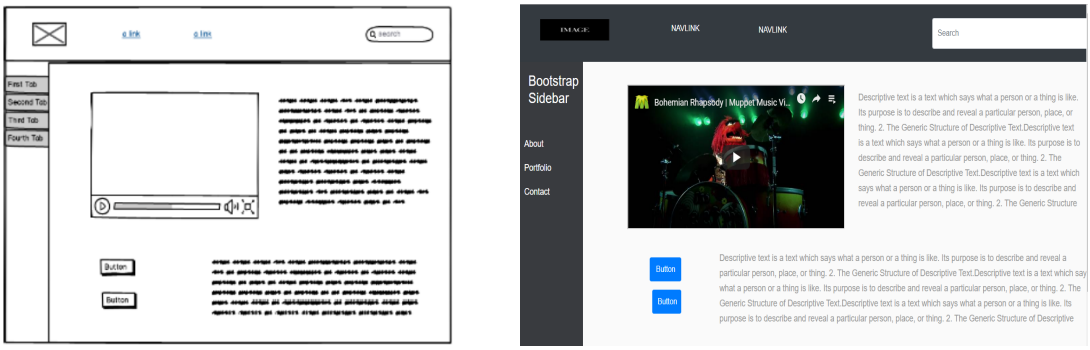
A tabela 4 contém alguns valores da pontuação de semelhança no subconjunto de teste, ao longo das várias iterações durante a fase de treino do modelo. Estes valores foram obtidos a partir de *checkpoints* guardados em cada iteração. A partir dos *checkpoints* é possível efetuar a avaliação do modelo no subconjunto de teste e obter vários resultados intermédios.

Epoch	48	58	68	78	88
Semelhança no subconjunto de teste (%)	59.51%	73.97%	83.02%	88.28%	83.26%

Tabela 4.: Pontuação de semelhança do modelo no subconjunto de teste.

O YOLO e o algoritmo de *layout* desenvolvido permitem obter resultados muito bons. Utilizando a métrica desenvolvida para esta abordagem (secção 5.3.5), em 254 minutos de treino obtém-se uma semelhança de 88.28% entre os esboços reais e gerados.

Apresentam-se de seguida alguns exemplos de predição avaliados e que englobam a totalidade dos elementos do conjunto de dados. Com estes exemplos pretende-se perceber quão corretamente o modelo reconhece os elementos presentes nos esboços e gera as respetivas páginas Web.



(a) Entrada fornecida (b) Respetiva saída

Figura 55.: Exemplo 1 de predição com a abordagem 2.

O exemplo representado na figura 55 demonstra um funcionamento perfeito do modelo. A página Web fornecida como entrada engloba uma grande quantidade e variedade de elementos. Da totalidade dos 15 tipos de elementos presentes no conjunto de dados, aos quais o modelo desenvolvido dá suporte, o esboço de entrada contém 8 elementos diferentes, ou seja, é representativo de mais de metade dos tipos de elementos do conjunto.

O esboço da figura 56 contém 8 tipos de elementos diferentes. Este caso é interessante porque possuí um *slideshow*. Revela-se útil para mostrar a capacidade do modelo para detetar e localizar todos os elementos do conjunto de dados. Também neste caso, o comportamento do modelo é irrepreensível.

(a) Entrada fornecida (b) Respetiva saída

Figura 56.: Exemplo 2 de predição com a abordagem 2.

O exemplo da figura 57 mostra a capacidade do modelo para detetar um dos elementos mais utilizados nas aplicações Web, as tabelas. Este caso destaca ainda a capacidade do modelo para distinguir elementos muito parecidos, como é o caso das caixas de entrada de texto, as etiquetas e os botões. Estes elementos muitas vezes são pequenos, semelhantes e, por isso, difíceis de diferenciar.

(a) Entrada fornecida (b) Respetiva saída

Figura 57.: Exemplo 3 de predição com a abordagem 2.

Com o último exemplo apresentado, pretendia-se perceber qual o comportamento do modelo quando sujeito a esboços invulgares (figura 58). A entrada fornecida é composta por um grande número de botões dispostos de forma irregular na página Web. No entanto, conclui-se que o modelo teve a capacidade de reconhecer corretamente os elementos, assim como o seu posicionamento, com destaque para o último botão ligeiramente desalinhado dos botões colocados acima.

(a) Entrada fornecida (b) Respetiva saída

Figura 58.: Exemplo 4 de predição com a abordagem 2.

6.3 RESULTADOS

Nas seções anteriores apresentaram-se os resultados das experiências realizadas nas duas abordagens. Importa referir que o conjunto de dados utilizado nas duas abordagens contém as mesmas imagens (esboços). A legenda das imagens é que varia entre uma DSL (abordagem 1) e uma descrição em XML dos objetos mais a respetiva localização das regiões delimitadoras (abordagem 2). Apesar das linguagens utilizadas serem diferentes, as legendas são equivalentes, pois descrevem a mesma imagem.

As células CuDNNLSTM e CuDNNGRU são RNNs normais otimizadas via CUDA e, por isso, permitem um treino mais rápido em GPUs. Os GPUs permitem um nível elevado de paralelismo. Executam a maioria das operações de álgebra linear paralelamente e, consequentemente, melhoram o desempenho do treino de redes neuronais. O CUDA fornece uma interface que facilita a exploração do paralelismo disponível nos GPUs. A biblioteca CuDNN disponibiliza às ferramentas de aprendizagem automática uma implementação das operações de álgebra linear otimizada para GPUs da Nvidia. De acordo com as tabelas 2 e 3, confirma-se que para o mesmo número de *epochs* a versão CUDA demora consideravelmente menos tempo, mantendo uma precisão semelhante. Verifica-se também que a precisão final é ligeiramente superior na versão CUDA das RNNs.

As células GRU são relativamente mais recentes que as LSTMs, apresentam desempenho semelhante e requerem menos cálculos. De acordo com os resultados obtidos na experiência 1 (tabela 2), com poucos dados de treino, as GRU treinam mais rapidamente e apresentam melhores resultados do que as LSTMs. Ainda de acordo com a tabela 3, com o mesmo número de *epochs* as GRUs apresentam melhores resultados. A versão CUDA, ainda que seja com mais *epochs*, acaba por apresentar melhores resultados. As LSTMs têm a vantagem de conseguir memorizar informação proveniente de sequências mais longas, devido a terem uma arquitetura mais complexa.

Na primeira abordagem, após várias tentativas com diferentes redes neuronais, com e sem *transfer learning*, e com diferentes hiperparâmetros, obteve-se como melhor resultado 71.30% de semelhança no subconjunto de teste. Este resultado foi obtido com *transfer learning*, onde os pesos foram obtidos com o conjunto de dados `Imagenet`, e com *fine-tuning*, onde os pesos das primeiras camadas não são atualizados.

Foram avaliados vários modelos alternativos, com e sem *transfer learning*, com *fine-tuning* e variação das camadas congeladas, com diferentes otimizadores e taxas de aprendizagem. Neste caso, a utilização de GRUs ou LSTMs tem pouca influência no resultado final. Tanto a ResNet50 como a Inceptionv3 apresentam melhores resultados no conjunto de dados Imagenet do que a VGG16 ou a VGG19. No entanto, quando comparado com a Inception-ResNetV2, esta apresenta ainda melhores resultados do que a ResNet50 ou a Inceptionv3 no Imagenet. A partir dos resultados da tabela 2, nota-se que quanto melhor os resultados no Imagenet, pior os resultados obtidos no modelo desenvolvido. Após várias tentativas não foi possível ultrapassar os 35% de semelhança no conjunto de teste com a ResNet, a InceptionV3 ou InceptionResNetV2. Estes resultados são bastante limitados em comparação com o resultado das VGG.

Devido às limitações de memória do GPU utilizado no desenvolvimento da dissertação, uma Nvidia GTX 1070 com 8GB de RAM, algumas destas redes não puderam ser utilizadas sem *fine-tuning*. Por exemplo ao usar a InceptionResNetV2, que contém 200 camadas convolucionais, foi necessário "congelar" mais de metade das camadas para reduzir a quantidade de memória necessária, o que acabou por limitar as experiências realizadas e, consequentemente, os resultados obtidos. Apenas as redes VGG16 e VGG19 permitiram obter resultados satisfatórios. Estas redes possuem menor quantidade de camadas convolucionais do que as referidas anteriormente.

Na segunda abordagem foi utilizada a terceira versão do YOLO. Fizeram-se testes com diferentes taxas de aprendizagem e com vários otimizadores. Esta abordagem atingiu um excelente resultado de 88.28% de semelhança no conjunto de teste. O código HTML gerado pela segunda abordagem é muito mais semelhante à entrada fornecida do que na abordagem 1. O YOLO encontra a localização exata da região delimitadora e o algoritmo de *layout* coloca o elemento na posição correta com base nas coordenadas, pelo que a representação fica muito mais semelhante. O mesmo não acontece na primeira abordagem, que não preserva as margens nem o tamanho dos elementos. Na primeira abordagem, os elementos têm tamanho e posição atribuídos por omissão, provenientes de *templates*.

7

CONCLUSÕES E TRABALHO FUTURO

Neste capítulo apresentam-se as conclusões e o trabalho futuro. Na primeira secção encontram-se as conclusões que resultaram da realização da presente dissertação, com destaque para as vantagens da segunda abordagem em relação à primeira. A segunda secção sumaria o trabalho futuro, incluindo o que ficou por fazer e o que poderá melhorar no projeto realizado.

7.1 CONCLUSÕES

A presente dissertação incluiu um levantamento bibliográfico sobre aprendizagem automática, focado na aprendizagem profunda com redes neuronais, as tecnologias utilizadas e os trabalhos relacionados. Incluiu ainda o estudo do problema com o levantamento de requisitos e a conceção dos protótipos de baixa e alta fidelidade de uma aplicação Web simples. O resultado do levantamento bibliográfico foi, posteriormente, aplicado no desenvolvimento das duas abordagens. A primeira abordagem, resulta da junção de redes CNN com RNNs e inspira-se em trabalhos sobre legendagem de imagens. A segunda abordagem baseia-se no YOLO para deteção e localização de objetos.

Após efetuar o levantamento bibliográfico, iniciou-se o levantamento de requisitos para a aplicação Web que permite ao utilizador utilizar os modelos de aprendizagem automática que convertem um esboço de página Web em código HTML. Identificar os requisitos foi fundamental para se desenvolver uma aplicação Web de acordo com o que se pretendia.

Idealmente, o conjunto de dados usado para treinar as redes neuronais deveria ser constituído por esboços elaborados à mão. Os esboços seriam digitalizados e legendados com código DSL. Isto porque os traços manuais dificultam imenso a tarefa de aprendizagem. Devido à morosidade necessária para conceber grandes quantidades de esboços, o tamanho do conjunto de dados seria bastante reduzido. Tendo em conta a necessidade de um conjunto de dados maior e com mais diversidade, optou-se pela utilização de uma ferramenta informática, o *Balsamiq mockups*. A utilização de uma ferramenta como o *Balsamiq* torna os elementos muito mais semelhantes, o que facilita a tarefa do modelo.

Os esboços inicialmente feitos à mão foram replicados na ferramenta e os resultados sofreram melhorias drásticas. As duas abordagens utilizaram conjuntos de dados ligeiramente diferentes. Partilham as mesmas imagens, mas com legendas diferentes. Os esboços da primeira abordagem foram descritos com código na DSL desenvolvida, enquanto que na segunda abordagem as legendas são descritas em XML, onde se especifica o nome de cada objeto e a localização da sua região delimitadora.

A primeira abordagem decorreu em várias etapas. As principais tarefas realizadas em cada etapa são as seguintes: construção da DSL, implementação em ANTLR4 do compilador utilizado para converter a linguagem DSL em código HTML funcional, desenvolvimento dos modelos em Keras com CNN e RNNs, e a criação da métrica de avaliação. Em treino, o modelo recebe como entrada uma imagem e o respetivo código DSL. O modelo aprende a reconhecer elementos, linhas e colunas a partir das entradas. Em predição, o modelo recebe apenas uma imagem e gera código DSL para essa entrada. O compilador recebe o código DSL e faz a transformação em código HTML funcional. As caraterísticas dos elementos a inserir no código HTML estão guardadas num ficheiro, que contém um *template* para cada tipo de elemento das páginas Web. Acontece que nesta abordagem os elementos não variam de tamanho e a posição é definida apenas por linha e coluna, o que limita o aspeto final da página gerada. Deste modo, o resultado final é sempre diferente do esboço fornecido como entrada.

A segunda abordagem surgiu do facto de faltar na primeira abordagem um adequado tratamento da componente espacial dos esboços. Nesta utilizou-se a saída do YOLO para identificar os elementos de um esboço, assim como a respetiva localização. O algoritmo de *layout* desenvolvido mapeia os objetos encontrados no YOLO em código HTML e CSS, com base nas coordenadas da região delimitadora, aproximado às coordenadas do objeto. Este algoritmo coloca os elementos num ficheiro HTML, utilizando propriedades do CSS que permitem colocar os elementos dadas as suas coordenadas.

As métricas de avaliação desenvolvidas para cada abordagem aplicam os mesmos pesos: ao número de elementos gerados corretamente é atribuído um peso de 80% e os restantes 20% são atribuídos às dimensões e ao posicionamento dos elementos. Com as métricas desenvolvidas e o mesmo conjunto de dados, é possível comparar os melhores resultados obtidos em cada abordagem. A primeira abordagem atingiu como melhor pontuação de semelhança entre os esboços reais e esperados 71.30%, enquanto que a segunda abordagem alcançou como melhor resultado 88.28%. A segunda abordagem gera código HTML que contém objetos com tamanho e posição corretos, o que resulta naturalmente em páginas Web muito mais parecidas ao esboço de entrada.

Os trabalhos realizados por Beltramelli (2017) e Asıroglu et al. (2019) são semelhantes ao projeto desenvolvido na presente dissertação. O Beltramelli (2017) utiliza uma CNN para extração de caraterísticas e duas RNNs, uma para codificar o código DSL e outra como descodificador, exatamente como foi desenvolvido na abordagem 1. A precisão alcançada foi 77%, ligeiramente acima do obtido na abordagem 1 desta dissertação. O trabalho de Asıroglu et al. (2019) combina um detetor de objetos, semelhante ao YOLO, com uma LSTM bidirecional para descodificar o vetor de caraterísticas. Utilizaram um conjunto de dados relativamente pequeno, contendo imagens de esboços realizados à mão. Obtiveram uma precisão de 73% no conjunto de validação.

A abordagem com YOLO desenvolvida nesta dissertação, cobre uma grande variedade de elementos HTML e atingiu uma semelhança de 88.28% no conjunto de teste segundo a métrica desenvolvida (subsecção 5.3.6). Este resultado muito bom pode ainda melhorar com ajustes em quantidade e diversidade ao conjunto de dados.

7.2 TRABALHO FUTURO

Uma dissertação tem uma dimensão tal que naturalmente permite que o trabalho realizado continue depois de concluída a dissertação. Como trabalho futuro propõe-se a implementação de um algoritmo de *layout* com divisão por linha e coluna, tal como acontece no `Bootstrap`. Uma vez que a abordagem com YOLO fornece as coordenadas das regiões delimitadoras, o algoritmo a desenvolver deve conseguir encontrar as margens corretas, de forma a posicionar os elementos mais aproximadamente ao mapeamento de coordenadas atual e, deste modo, tornar o código gerado corretamente responsivo.

Para corrigir o maior problema encontrado no desenvolvimento desta dissertação, a falta de dados, propõem-se aumentar o tamanho e diversidade do conjunto de dados. Esta medida pretende melhorar a precisão do modelo na deteção correta dos objetos, mas fundamentalmente melhorar a precisão das coordenadas da região delimitadora. Pretende-se também aumentar o número de elementos HTML suportados.

Sugere-se ainda a realização de testes de usabilidade com utilizadores representativos do público-alvo, o que permite ao analista observar e anotar possíveis melhorias na interface da aplicação, tendo em vista torná-la mais intuitiva e apelativa. Esta medida ajudará a tornar a aplicação Web mais simples de utilizar.

Pretende-se ainda criar métricas de avaliação de precisão e *recall*, com o objetivo de avaliar os modelos desenvolvidos com métricas diferentes. A última tarefa consiste em preparar um servidor para colocar a aplicação em produção.

BIBLIOGRAFIA

Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Matthieu Devin Jeffrey Dean, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. 2016.

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *IEEE SIGNAL PROCESSING MAGAZINE*, 34, 2017. ISSN 1053-5888.

Nurshazlyn Mohd Aszemi and P.D.D Dominic. Hyperparameter optimization in convolutional neural network using genetic algorithms. *International Journal of Advanced Computer Science and Applications*, 10, 06 2019. doi: 10.14569/IJACSA.2019.0100638.

Batuhan Asıroglu, Busra Rumeysa Mete, Eyyup Yıldız, Yagız Nalcakan, Alper Sezen, Mustafa Dagtekin, and Tolga Ensari. Automatic html code generation from mock-up images using machine learning techniques. *2019 Scientific Meeting on Electrical-Electronics & Biomedical Engineering and Computer Science (EBBT)*, 2019. doi: 10.1109/EBBT.2019.8741736.

Saumya Bajpai, Kreeti Jain, and Neeti Jain. Artificial neural networks. *International Journal of Soft Computing and Engineering (IJSCE)*, 2011.

Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. Deepcoder: Learning to write programs. *ICLR 2017*, 2017.

Balsamiq. URL https://balsamiq.com/company/. Consultado a 09/07/2019.

Tony Beltramelli. pix2code: Generating code from a graphical user interface screenshot. *arXiv*, abs/1705.07962, 2017.

Bootstrap. History. URL https://getbootstrap.com/docs/4.1/about/overview/. Consultado a 09/07/2019.

Bootstrap. Bootstrap 3 released, August 2013. URL https://blog.getbootstrap.com/2013/08/19/\bootstrap-3-released/. Consultado a 09/07/2019.

Nicola Capece, Ugo Erra, and Antonio Vito Ciliberto. Implementation of a coin recognition system for mobile devices with deep learning. *International Conference on Signal-Image Technology & Internet-Based Systems*, 2016.

Rich Caruana, Steve Lawrence, and Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. *Advances in Neural Information Processing Systems*, 13:402–408, 01 2001.

Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. *Semi-Supervised Learning*. The MIT Press, 1st edition, 2010. ISBN 0262514125, 9780262514125.

Chunyang Chen, Ting Su, Guozhu Meng, Zhenchang Xing, and Yang Liu. From ui design image to gui skeleton: A neural machine translator to bootstrap mobile gui implementation. *The 40th International Conference on Software Engineering, Gothenburg, Sweden*, 2018.

François Chollet. *Deep Learning with python*. Manning, 2 edition, 2017.

Kwetishe Joro Danjuma. Performance evaluation of machine learning algorithms in post-operative life expectancy in the lung cancer patients. *ArXiv*, abs/1504.04646, 2015.

Yuntian Deng, Anssi Kanervisto, Jeffrey Ling, and Alexander M. Rush. Image-to-markup generation with coarse-to-fine attention. *International Conference on Machine Learning*, 2017.

Rahul Dey and Fathi M. Salem. Gate-variants of gated recurrent unit (gru) neural networks. *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2017. ISSN 1558-3899. doi: 10.1109/MWSCAS.2017.8053243.

Cathal Dooley. Data visualisation and machine learning web application with potential use in sports data analytics. Master's thesis, OE Gaillimh, March 2017.

Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. Master's thesis, University Of California, Irvine, 2000.

Flask. Welcome to flask. URL http://flask.pocoo.org/docs/1.0/. Consultado a 09/07/2019.

Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, September 2016.

Amir Ghasemian, Homa Hosseinmardi, and Aaron Clauset. Evaluating overfit and underfit in models of network community structure. *IEEE Transactions on Knowledge and Data Engineering*, PP, 2019. ISSN 2326-3865. doi: 10.1109/TKDE.2019.2911585.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *NIPS*, 2014.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. *arXiv*, 1506.02626, 2015.

Hiroaki Hayashi, Jayanth Koushik, and Graham Neubig. Eve: A gradient based optimization method with locally and globally adaptive learning rates. *arXiv*, abs/1611.01505, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Geoffrey Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15, 2014.

Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. 1997.

MD. Zakir Hossain, Ferdous Sohel, Mohd Fairuz Shiratuddin, and Hamid Laga. A comprehensive survey of deep learning for image captioning. *arXiv*, abs/1810-04020, October 2018.

Tomáš Kociský Jeremy Appleyard and Phil Blunsom. Optimizing performance of recurrent neural networks on gpus. *arXiv*, abs/1604.01946, April 2016.

Daniel Kent and Fathi Salem. Performance of three slim variants of the long short-term memory (lstm) layer. *2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 307–310, 2019.

Keras. Keras documentation. URL `https://keras.io/`. Consultado a 19/08/2019.

Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. *International Joint Conference on Articial Intelligence (IJCAI)*, 2001.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning, 2015.

Piji Li. Optimization algorithms for deep learning. 2017.

Rui Li, Howard D. Bondell, and Brian J. Reich. Deep distribution regression. *ArXiv*, abs/1903.06023, 2019.

Tianyi Liu, Shuangsang Fang, Yuehui Zhao, and Peng Wang. Implementation of training convolutional neural networks. *arXiv*, abs/1506.01195, 2015.

Lili Mou, Rui Men, Ge Li, Lu Zhang, and Zhi Jin. On end-to-end program generation from user intention by deep neural networks. *arXiv*, abs/1510.07211, 2015.

Vikram Mullachery and Vishal Motwani. Image captioning. *arXiv*, abs/1805.09137, December 2016.

Paula C. Useche Murillo, Robinson Jiménez Moreno, and Javier O. Pinzón Arenas. Comparison between cnn and haar classifiers for surgical instrumentation classification. *Contemporary Engineering Sciences*, 10, 2017. ISSN 1351-1363.

Tuan Anh Nguyen and Christoph Csallner. Reverse engineering mobile application user interfaces with remaui. 2015.

Jakob Nielsen. Paper prototyping: Getting user data before you code. April 2003. ISSN 1548-5552.

Michael A. Nielsen. Neural networks and deep learning, 2018. URL http://neuralnetworksanddeeplearning.com/.

Aboelmagd Noureldin, Ahmed El-Shafie, and Mohamed Bayoumi. Gps/ins integration utilizing dynamic neural networks for vehicular navigation. *Elsevier*, 12, 2011.

Ismoilov Nusrat and Sung-Bong Jang. A comparison of regularization techniques in deep neural networks. *Symmetry*, 10, 11 2018. doi: 10.3390/sym10110648.

Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *ArXiv*, abs/1811.03378, 2018.

Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, December 2016. ISSN 1063-6919. doi: 10.1109/CVPR.2017.690.

Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *ArXiv*, abs/1804.02767, April 2018.

Frank F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *ArXiv*, abs/1609.04747, 2016.

David Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. Backpropagation: The basic theory. 1986.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A modern approach*. Prentice Hall Press, 3rd edition, 2009. ISBN 0136042597, 9780136042594.

Nitesh Sharma, Rachit Pabreja, Ussama Yaqub, Vijayalakshmi Atluri, Soon Chun, and Jaideep Vaidya. Web-based application for sentiment analysis of live tweets. pages 1–2, 05 2018. doi: 10.1145/3209281.3209402.

Phil Simon. *Too Big to Ignore: The Business Case for Big Data*. Wiley, 2013.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR 2015*, abs/1409.1556, April 2015.

Lakshminarasimhan Srinivasan, Dinesh Sreekanthan, and Amutha A.L. Image captioning - a deep learning approach. *International Journal of Applied Engineering Research*, 13, 2018. ISSN 0973-4562.

Yash Srivastava, Vaishnav Murali, and Shiv Ram Dubey. A performance comparison of loss functions for deep face recognition. *ArXiv*, abs/1901.05903, 2019.

StackOverflow. Developer survey results 2019. URL https://insights.stackoverflow.com/survey/\2019?utm_source=so-owned&utm#technology-_-web-frameworks_medium=blog&utm_ca\mpaign=dev-survey-2019&utm_content=launch-blog. Consultado a 19/07/2019.

Humayun Karim Sulehria and Ye Zhang. Hopfield neural networks - a survey. *World Scientific and Engineering Academy and Society (WSEAS)*, February 2007.

Alan Turing. *Computing Machinery and Intelligence*. Mind, 1950.

Jaime Cepeda Villamayor. Car features recognition for insurance applications using machine learning. Master's thesis, Universidad de Castilla-La Mancha, July 2017.

Bernard Widrow. An adaptative "adaline"neuron using chemical "memistors". 1960.

Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv*, abs/1502.03044, 2015.

Youjun Xu, Jianfeng Pei, , and Luhua Lai. Deep learning based regression and multiclass models for acute oral toxicity prediction with automatic chemical feature extraction. *Journal of Chemical Information and Modeling*, 57, 10 2017. doi: 10.1021/acs.jcim.7b00244.

Łukasz Kaiser, Aidan N. Gomez, and François Chollet. Depthwise separable convolutions for neural machine translation. *ArXiv*, abs/1706.03059, 2017.

Željko Jovanović, Dijana Jagodić, Dejan Vujičić, and Siniša Randić. Java spring boot rest web service integration with artificial intelligence weka framework. *International Scientific Conference*, November 2017.

A

ESBOÇO E CÓDIGO HTML GERADO PELO MODELO

Este anexo contém o esboço fornecido como entrada ao modelo da segunda abordagem (figura 59) e o código HTML, CSS e Bootstrap gerado. Para simplificar, o código contém os estilos CSS no mesmo ficheiro que o código HTML, dentro da etiqueta de estilo.

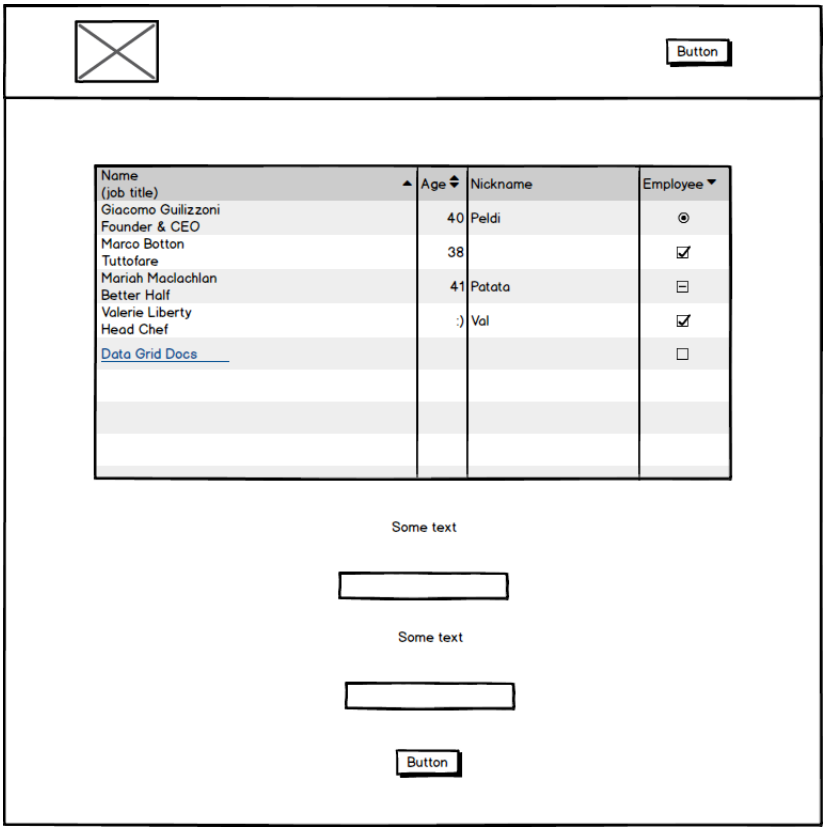


Figura 59.: Esboço fornecido ao modelo.

```
<html lang="en">
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">
<!-- The above 3 meta tags *must* come first in the head; any other head
    content must come *after* these tags -->
<title>Bootstrap Template</title>
<!-- Bootstrap -->
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/
    css/bootstrap.min.css">
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.
    js"></script>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"
    ></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.6/umd/
    popper.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.
    js"></script>
<!--[if lt IE 9]>
<script src="https://oss.maxcdn.com/html5shiv/3.7.3/html5shiv.min.js"></script
    >
<script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
<![endif]--><style>
#table0 {
 position: absolute;
 left: calc(calc(0*100vw)/256);
 top: calc(calc(42*100vh)/256);
 width: calc(calc(255*100vw)/256);
 height: calc(calc(116*100vh)/256);
}
#button1 {
 position: absolute;
 left: calc(calc(125*100vw)/256);
 top: calc(calc(229*100vh)/256);
 width: calc(calc(16*100vw)/256);
 height: calc(calc(13*100vh)/256);
 line-height: 0px;
}
#input2 {
 position: absolute;
 left: calc(calc(95*100vw)/256);
 top: calc(calc(177*100vh)/256);
 width: calc(calc(70*100vw)/256);
 height: calc(calc(8*100vh)/256);
}
#input3 {
```

```
 position: absolute;
 left: calc(calc(99*100vw)/256);
 top: calc(calc(209*100vh)/256);
 width: calc(calc(66*100vw)/256);
 height: calc(calc(7*100vh)/256);
}
#label4 {
 position: absolute;
 left: calc(calc(118*100vw)/256);
 top: calc(calc(159*100vh)/256);
 width: calc(calc(24*100vw)/256);
 height: calc(calc(7*100vh)/256);
}
#label5 {
 position: absolute;
 left: calc(calc(119*100vw)/256);
 top: calc(calc(194*100vh)/256);
 width: calc(calc(25*100vw)/256);
 height: calc(calc(7*100vh)/256);
}
#navbarid { height: calc(calc(29*100vh)/256);
 }
#sidebar {margin-top: calc(calc(29*100vh)/256); }
#button6 {
 position: absolute;
 left: calc(calc(211*100vw)/256);
 top: calc(calc(7*100vh)/256);
 width: calc(calc(16*100vw)/256);
 height: calc(calc(12*100vh)/256);
 line-height: 0px;
}
#image7 {
 position: absolute;
 left: calc(calc(14*100vw)/256);
 top: calc(calc(3*100vh)/256);
 width: calc(calc(41*100vw)/256);
 height: calc(calc(23*100vh)/256);
}
</style>
</head>
<body>
<table id="table0" class="table table-striped">
    <thead>
        <tr><th scope="col">#</th><th scope="col">Col1</th><th scope="col">
            Col2</th><th scope="col">Col3</th></tr>
    </thead>
    <tbody>
```

```
        <tr><th scope="row">1</th><td>------</td><td>------</td><td>------</td
            ></tr>
        <tr><th scope="row">2</th><td>------</td><td>------</td><td>------</td
            ></tr>
        <tr><th scope="row">3</th><td>------</td><td>------</td><td>------</td
            ></tr>
    </tbody>
</table>
<button type="button" class="btn btn-primary" id="button1">Button</button>
<input id="input2" type="text" placeholder="Input text" required
><input id="input3" type="text" placeholder="Input text" required
><label id="label4" for="uname"><b>Label</b></label>
<label id="label5" for="uname"><b>Label</b></label>
<nav id="navbarid" class="navbar navbar-expand navbar-dark bg-dark">
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-
      target="#navbarText" aria-controls="navbarText" aria-expanded="false"
      aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
    <div class="navbar-collapse collapse justify-content-between">
        <ul class="navbar-nav mr-auto">
            <button type="button" class="btn btn-primary" id="button6">Button
                </button>
            <img id="image7" src="https://www.image.ie/images/no-image.png">
        </ul>
    </div>
</nav>
</body>
</html>
```

Listagem A.1: Código gerado pelo modelo da segunda abordagem.