

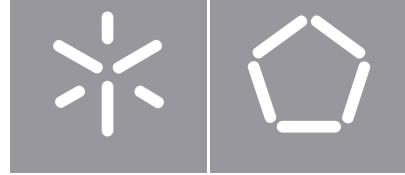


**Universidade do Minho**  
Escola de Engenharia

Gonçalo Braz Afonso

**OCR para documentos estruturados antigos**  
**Old structured documents OCR**





**Universidade do Minho**  
Escola de Engenharia

Gonçalo Braz Afonso

**OCR para documentos estruturados antigos**  
**Old structured documents OCR**

Dissertação de Mestrado  
Mestrado em Engenharia Informática

Trabalho efetuado sob a orientação de  
**José João Antunes Guimarães Dias Almeida**

# **Direitos de Autor e Condições de Utilização do Trabalho por Terceiros**

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada.

Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositórioUM da Universidade do Minho.

## **Licença concedida aos utilizadores deste trabalho:**



**CC BY-NC-SA**

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

# **Agradecimentos**

A consolidação do atual trabalho servirá, expectavelmente, como um marco da fase de estudo que, mais ou menos duradoura, a todos marca.

Alcançar este ponto é prova não apenas do meu esforço, mas de todos os que sempre me apoiaram, em especial os meus pais. A educação e formação que me incutiram e os ideais que me constituem são, por base, fruto da sua dedicação e carinho.

Também a boa disposição e companheirismo inabaláveis da minha irmã não poderiam ser ignorados.

De igual forma agradeço à minha avô, que forneceu os momentos mais ternos da minha infância e juventude, e que sempre teve confiança nas minhas capacidades.

Agradeço aos professores e escolas que me formaram durante o caminho e serviram como alicerces da minha aprendizagem. Ao meu orientador, prof. José João por ter sido um guia paciente e elucidador durante este ano de trabalho, pelas inúmeras ideias que sugeriu e tornaram este um trabalho mais interessante, pelas palavras de experiência e as oportunidades que forneceu.

Aos meus amigos que me acompanharam durante este percurso, que forneceram momentos insubstituíveis e laços preciosos, uma saudação calorosa.

A todos os que direta ou indiretamente me marcaram, 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.

Universidade do Minho, Braga, outubro 2024

Gonçalo Braz Afonso

# Resumo

A digitalização de documentos permitiu uma nova forma de salvaguardar informação para a posteridade, evitando a sua perda pelo deterioramento físico destes. De forma a posteriormente transcrever estes documentos, permitindo uma consulta, processamento e manipulação mais simples, o uso de software de **OCR** é essencial. Esta tecnologia é, no entanto, dependente em diferentes níveis das características do seu alvo, nomeadamente: qualidade da imagem, complexidade da estrutura do documento, linguagem do texto.

Documentos mais antigos, em especial jornais por possuírem estruturas complexas, apresentam por este motivo resultados que diferem bastante do seu conteúdo original; tanto a nível do texto reconhecido, como da sua organização para os diferentes outputs disponíveis (ex.: txt simples). A tarefa de extrair informação destes documentos, como o isolamento e extração de artigos, torna-se assim complexa e propensa a erros.

Este trabalho propõe então a criação de uma solução, focada neste tipo de documentos, que afetando os dados que rodeiam o processo de **OCR**, i.e. imagem, resultados de **OCR** e texto; permita resolver problemas que este apresente, particularmente na dimensão dos resultados de **OCR**, e assim proporcione formas de melhorar a transcrição final.

Na base da solução, está o modelo **OCR Tree** que procura servir como representação universal dos resultados de OCR ou, por outras palavras, serve como árvore de estrutura e conteúdo documental.

A solução é composta por 3 componentes principais:

- **Toolkit**: com ferramentas para análise e tratamento de imagem; processamento de texto; e especial foco na análise e manipulação de OCR Tree.
- **Pipeline**: aplicação do Toolkit e ferramentas auxiliares numa linha de uso de **OCR**, configurável. Permitirá obter resultados objetivos sobre o Toolkit.
- **Editor Gráfico de OCR Tree**: interface gráfica que permita a manipulação manual facilitada de OCR Tree. Faz uso do Toolkit e da Pipeline nas operações que disponibiliza.

**Palavras-chave**   OCR, Digitalização, Documentos estruturados, Documentos antigos, Segmentação de documentos, Tratamento de imagem, Editor gráfico

# Abstract

The digitization of documents has opened a new way of preserving information for posterity, avoiding its loss through their physical decay. To allow the transcription of these documents, enabling an easier search, indexation and manipulation of them, the use of **OCR** software is essential. This technology is, however, dependent in many ways on the characteristics of its target, namely: the quality of the image, the complexity of the document's structure, the text's language.

Older documents, especially newspapers for having complex structures, result in poor transcriptions that differ from their original content, both in the recognized text, and in the organization of the available final outputs (ex.: simple txt). Extracting information from these documents, such as the segmentation and extraction of articles, becomes thus a complex and error prone task.

This work proposes a solution, focused on these documents, which by targeting the data that encompasses the process of **OCR**, i.e. image, **OCR** results and text; can solve problems that arise in this process, particularly on the issue of **OCR** results, thus providing ways to enhance the final transcript.

The basis of this solution is the OCR Tree module, which aims to serve as an universal representation of **OCR** results or, in other words, serves represents the structural and content tree of a document.

The solution is composed by 3 main components:

- **Toolkit**: with tools for analysis and image treatment; text processing; and special focus on the analysis and manipulation of OCR Tree.
- **Pipeline**: an application of the Toolkit and auxiliary tools in the form of a configurable pipeline surrounding the use of OCR. Will as a result allow gathering objective metrics of the Toolkit.
- **OCR Tree Graphical Editor**: graphical interface that allows for an easy manual manipulation of the OCR Tree. Makes use of the Toolkit and the Pipeline in the provided functionalities.

**Keywords**   OCR, Digitalization, Structured documents, Old documents, Document segmentation, Image processing, Graphical Editor

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Enquadramento e motivação . . . . .	1
1.2	Objetivos . . . . .	2
1.3	Estrutura da dissertação . . . . .	3
<b>2</b>	<b>Estado da arte</b>	<b>5</b>
2.1	Reconhecimento ótico de caracteres . . . . .	5
2.1.1	Introdução . . . . .	5
2.1.2	Breve história e evolução . . . . .	6
2.1.3	Processo OCR . . . . .	7
2.1.4	Desafios . . . . .	8
2.1.5	Tecnologia . . . . .	9
2.2	Pré Processamento para OCR . . . . .	10
	Métricas de avaliação . . . . .	13
2.3	Pós Processamento para OCR . . . . .	14
	Junção de Outputs de OCR . . . . .	15
	Vias lexicais . . . . .	16
	Modelos de erro e Máquinas de estado finitas com pesos . . . . .	16
	Modelos de linguagem baseados em tópicos . . . . .	16
	Modelos de linguagem . . . . .	17
	Machine Learning baseado em características . . . . .	17
	Seq2Seq - Sequência para Sequência . . . . .	17
	Métricas de avaliação . . . . .	18
2.4	Segmentação de documentos . . . . .	18
	Algoritmos dedicados a um layout específico . . . . .	19

Algoritmos que usam parâmetros para descrever um layout . . . . .	19
Algoritmos para segmentação de layout potencialmente não restringidos . . . . .	20
<b>2.5 Trabalho relacionado . . . . .</b>	<b>20</b>
2.5.1 Extração de conteúdo de jornais . . . . .	20
Heurísticas . . . . .	21
Inteligência artificial . . . . .	22
2.5.2 Ordem de leitura . . . . .	22
<b>2.6 Conclusões . . . . .</b>	<b>24</b>
<b>3 OSDOCR Filosofia . . . . .</b>	<b>25</b>
3.1 O problema . . . . .	25
3.2 Modelos . . . . .	29
3.2.1 Imagem . . . . .	29
3.2.2 OCR Tree - estrutura de dados para resultados OCR . . . . .	30
3.3 Arquitetura da solução . . . . .	31
3.3.1 OSDOCR Toolkit . . . . .	32
3.3.2 OSDOCR Pipeline . . . . .	33
3.3.3 OSDOCR Editor . . . . .	34
<b>4 OSDOCR Estruturas de Dados - Implementação . . . . .</b>	<b>36</b>
4.1 OCR Tree . . . . .	36
4.2 Box . . . . .	47
<b>5 OSDOCR Toolkit - Implementação . . . . .</b>	<b>51</b>
5.1 Sumário . . . . .	51
5.2 Processamento de resultados OCR . . . . .	52
5.2.1 Conversão de resultados OCR . . . . .	53
5.2.2 Análise de texto . . . . .	55
5.2.3 Limpeza de OCR Tree . . . . .	58
5.2.4 Categorização de blocos . . . . .	67
5.2.5 Divisão de blocos . . . . .	68
5.2.6 Cálculo de ordem de leitura . . . . .	73
5.2.7 Segmentação de resultados . . . . .	77
5.3 Processamento de imagem . . . . .	78

5.3.1	Correção de ângulo de rotação . . . . .	79
5.3.2	Corte de sombra nas margens . . . . .	81
5.3.3	Binarização de imagem . . . . .	82
5.3.4	Identificação de delimitadores . . . . .	84
5.3.5	Identificação de imagens no documento . . . . .	88
5.3.6	Segmentação de documento . . . . .	90
5.4	Processamento de texto . . . . .	91
5.4.1	Correção de hifenização . . . . .	92
5.4.2	Geração de output . . . . .	92
<b>6</b>	<b>OSDOCR Pipeline - Implementação</b>	<b>94</b>
6.1	Configuração . . . . .	94
6.2	Pré-processamento de imagem . . . . .	98
6.3	OCR . . . . .	103
6.4	Pós-processamento de OCR . . . . .	105
6.5	Geração de output . . . . .	108
6.6	Validação de resultados . . . . .	111
<b>7</b>	<b>OSDOCR Editor - Implementação</b>	<b>114</b>
7.1	Sumário . . . . .	114
7.2	Funcionalidades . . . . .	115
7.2.1	Inputs . . . . .	115
7.2.2	Manipulação manual de OCR Tree . . . . .	116
7.2.3	Aplicação local de OCR . . . . .	118
7.2.4	Ferramentas disponíveis . . . . .	119
7.2.5	Outputs . . . . .	125
7.2.6	Operações adicionais . . . . .	126
<b>8</b>	<b>Resultados</b>	<b>130</b>
8.1	OSDOCR Toolkit . . . . .	130
8.2	OSDOCR Pipeline . . . . .	131
8.2.1	Metodologia . . . . .	131
8.2.2	Resultados . . . . .	133
8.2.3	Conclusão . . . . .	140

8.3	OSDOCR Editor . . . . .	141
<b>9</b>	<b>Conclusões e trabalho futuro</b>	<b>144</b>
9.1	O Trabalho . . . . .	144
9.2	Conclusões . . . . .	145
9.3	Perspetiva de trabalho futuro . . . . .	146
<b>A</b>	<b>Detalhes dos resultados</b>	<b>152</b>
<b>B</b>	<b>Ferramentas</b>	<b>159</b>

# Acrónimos

**CNN** Convolutional Neural Network.

**DPI** dots per inch.

**GAN** Generative Adversarial Network.

**LSTM** Long Short Term Memory.

**ML** Machine Learning.

**NLP** Natural Language Processing.

**OCR** reconhecimento óptico de caracteres.

**PSNR** Peak signal-to-noise ratio.

**SSIM** Structural similarity index measure.

# Glossário

**codec** Algoritmo de compressão com o propósito de diminuir a dimensão de uma entidade..

**ground truth** Em OCR, este é o nome dado ao conteúdo do documento original. Usualmente apenas o conteúdo textual é considerado..

**word embeddings** Forma de vetorização de uma palavra. O vetor de uma palavra representa os seus valores num conjunto de características..

# **Lista de Figuras**

1	Aplicação de OCR com passos para melhoria de resultados . . . . .	26
2	Exemplo de pipeline de aplicação de OCR . . . . .	26
3	Exemplo de pipeline de aplicação de OCR com blocos opcionais . . . . .	27
4	Representação visual dos resultados de OCR armazenados em formato HOCR . . . . .	30
5	Arquitetura geral da solução . . . . .	32
6	Arquitetura OSDOCR Toolkit . . . . .	33
7	Arquitetura OSDOCR Pipeline - high level . . . . .	33
8	Arquitetura OSDOCR Pipeline - reduzida . . . . .	34
9	Arquitetura MVC . . . . .	35
10	Árvore de resultados OCR num documento . . . . .	38
11	Exemplos de junção de OCR Tree . . . . .	45
12	Exemplo de jornal com resultados OCR ruidosos e com conflitos. . . . .	52
13	Exemplo de desenho de OCR Tree por cima do documento. . . . .	54
14	Exemplo de desenho de template de jornal. . . . .	54
15	Exemplo de desenho de artigos de jornal. Cada um dos artigos é desenhado com uma cor única. . . . .	55
16	Exemplo de alguns dos casos listados que o toolkit irá abranger. . . . .	58
17	Exemplo de uso de block_bound_box_fix (10 de confiança de texto mínima). . . . .	60
18	Exemplo de uso de text_bound_box_fix (10 de confiança de texto mínima). . . . .	61
19	Exemplo de uso de bound_box_fix_image (10 de confiança de texto mínima). . . . .	62
20	Exemplo de uso de unite_blocks (10 de confiança de texto mínima). . . . .	64
21	Exemplo de uso de split_whitespaces. . . . .	66
22	Exemplo de uso de delimiters_fix. . . . .	67
23	Exemplo de uso de categorize_blocks. . . . .	68

24	Divisão de bloco para separar título do texto (aplicou-se também um ajuste de bounding boxes para acentuar a diferença) . . . . .	69
25	Exemplo de uso de <code>split_block</code> com corte horizontal. . . . .	71
26	Exemplo de uso de <code>split_block</code> com corte vertical. . . . .	72
27	Exemplo de uso de <code>find_text_titles</code> com corte vertical. . . . .	72
28	Exemplo de jornal incorretamente ordenado. . . . .	73
29	Exemplo de cálculo da ordem de leitura de um jornal (exemplo não linear). . . . .	76
30	Exemplo de isolamento de artigos num jornal. Foram encontrados 3 artigos pelo algoritmo, cada um identificado com uma cor única. . . . .	77
31	Exemplo de aplicação de métodos de processamento de imagem do toolkit. . . . .	78
32	Variação dos resultados de OCR em diferentes níveis de rotação de texto. . . . .	79
33	Exemplo de aplicação do algoritmo. . . . .	80
34	Exemplo de documento com inclinação de texto e sombra de bainha. . . . .	81
35	Exemplo de resultado de OCR com diferentes binarizações em local sem ruído muito acentuado. . . . .	83
36	Exemplo de resultado de OCR com diferentes binarizações em local com muito ruído. . . . .	83
37	Página de jornal com delimitadores identificados (a vermelho). . . . .	85
38	Resultados de identificação de linhas verticais e horizontais. . . . .	87
39	Resultados de identificação de delimitadores, aplicados métodos de redução. . . . .	88
40	Exemplo de identificação de ilustrações em jornais com diferentes características. . . . .	89
41	Exemplo de uso de <code>divide_columns</code> numa página de jornal. . . . .	90
42	Exemplo de uso de <code>segment_document</code> em um jornal. . . . .	91
43	Exemplo de uso de <code>fix_hifenzation</code> em um excerto de texto. . . . .	92
44	Pipeline - secção pré-processamento . . . . .	99
45	Exemplo de diferença de OCR numa secção de texto. . . . .	100
46	Exemplo de resultado de passo de remoção das imagens identificadas. . . . .	101
47	Resultado de binarização correspondente a diferentes preenchimentos da zona das ilustrações removidas. . . . .	102
48	Resultado de modelo de denoise do waifu2x. . . . .	102
49	Resultado de uso de diferentes modelos de correção de iluminação. . . . .	103
50	Pipeline - secção OCR . . . . .	104
51	Pipeline - secção pós-processamento . . . . .	105

52	Resultado de passo de ajuste de bounding boxes utilizando diferentes valores de <i>conf</i> . . . . .	106
53	Resultado de aplicação dos passos de limpeza (passos até agora listados) dos resultados OCR. . . . .	107
54	Resultado de aplicação do passo de categorização de blocos. . . . .	107
55	Resultado final de pós processamento dos resultados de OCR . . . . .	108
56	Pipeline - secção output . . . . .	109
57	Resultado de cálculo da ordem de leitura. . . . .	110
58	Resultado de isolamento de artigos. . . . .	110
59	OCR Editor: escolha de inputs. 1 - Input de imagem. 2 - Input de ficheiro OCR Tree . . . . .	115
60	OCR Editor: seleção de blocos . . . . .	116
61	OCR Editor: mover blocos . . . . .	116
62	OCR Editor: redimensionar bloco . . . . .	116
63	OCR Editor: atualizar bloco. 1 - Selecionar bloco. 2 - Modificar dados do bloco. 3 - Guardar alterações. . . . .	117
64	OCR Editor: criar novo bloco. 1 - Ferramenta para criar novo bloco (ID 110). . . . .	117
65	OCR Editor: eliminar bloco. 1 - Selecionar bloco. 2 - Eliminar bloco. . . . .	118
66	OCR Editor: aplicar OCR. 1 - Selecionar bloco. 2 - Aplicar OCR. Resultado em cima - bloco único. Resultado em baixo - blocos resultantes. . . . .	119
67	OCR Editor: divisão de bloco. . . . .	120
68	OCR Editor: junção de bloco. . . . .	121
69	OCR Editor: junção de bloco, texto intercalado. . . . .	121
70	OCR Editor: categorização de blocos. . . . .	122
71	OCR Editor: ordenação de blocos. . . . .	123
72	OCR Editor: manipulação de artigos. . . . .	124
73	OCR Editor: divisão de imagem. . . . .	125
74	OCR Editor: output. . . . .	126
75	OCR Editor: operações adicionais. . . . .	128
76	OCR Editor: janela de configurações. . . . .	129
77	Amostras do conjunto de casos de teste. . . . .	132
78	Valores de confiança média de texto das diferentes pipelines. . . . .	134
79	Valores de similaridade do texto (similaridade por cosseno) com GT das diferentes pipelines.	135

80	Ráios de aparição total de palavras da GT das diferentes pipelines. . . . .	135
81	Ráios de aparição de palavras distintas da GT das diferentes pipelines. . . . .	136
82	Ráios de número de blocos de texto relativo à pipeline simples. . . . .	137
83	Resultados locais de 'o_portuguez_3' para ráio de palavras totais e distintas. . . . .	137
84	Ráios de aparição de linhas da Partial GT das diferentes pipelines. . . . .	138
85	Ráios de acerto da ordem das linhas da Partial GT das diferentes pipelines. . . . .	139
86	Resultados locais de casos de melhoria na ordenação da GT parcial. . . . .	140
87	Exemplo de melhoria substancial de melhoria de acerto na GT parcial. . . . .	140
88	Exemplos de acerto total de ordenação da GT parcial. . . . .	141
89	Valores de confiança média de texto das diferentes pipelines (alargado). . . . .	152
90	Valores de similaridade do texto (similaridade por cosseno) com GT das diferentes pipelines (alargado). . . . .	153
91	Ráios de aparição total de palavras da GT das diferentes pipelines (alargado). . . . .	154
92	Ráios de aparição de palavras distintas da GT das diferentes pipelines (alargado). . . . .	155
93	Ráios de aparição de linhas da Partial GT das diferentes pipelines (alargado). . . . .	156
94	Ráios de acerto da ordem das linhas da Partial GT das diferentes pipelines (alargado). . . . .	157
95	Ráios de número de blocos de texto relativo à pipeline simples (alargado). . . . .	158

## **Lista de Tabelas**

1	Média geral de confiança de texto das pipelines . . . . .	134
2	Média geral de similaridade do texto com GT. . . . .	135
3	Média geral de rácios de aparição total de palavras da GT. . . . .	136
4	Média geral de rácios de aparição de palavras distintas da GT. . . . .	136
5	Média de rácio de número de blocos de texto comparado com pipeline simples. . . . .	137
6	Média de rácio de acerto das linhas da Partial GT. . . . .	138
7	Média de rácio de ordem das linhas da Partial GT. . . . .	139

# **Capítulo 1**

## **Introdução**

Neste capítulo, será realizada uma introdução ao problema que o projeto tenciona abordar, composta por uma contextualização do seu estado atual e os desafios que sobre este são impostos. Além disso, os objetivos do trabalho serão listados e será descrita a estrutura do documento.

### **1.1 Enquadramento e motivação**

A digitalização tem um papel fundamental na conservação, disponibilização e proliferação de documentos físicos, não só contemporâneos, mas também de eras anteriores à revolução da informação. Esta tecnologia, acoplada a ferramentas de **OCR**, veio trazer uma facilidade de navegação, consulta e manipulação destes documentos que anteriormente não era possível.

A eficácia de **OCR** é no entanto dependente de vários fatores nas imagens ou ficheiros alvo: a qualidade das imagens, como a resolução, estado do documento, coloração, qualidade/tipo de escrita; a estrutura dos documentos - quanto mais complexo, mais difícil é obter a informação de forma automática mantendo a congruência original -; linguagem do texto, sendo que por vezes diferentes tecnologias, como por exemplo **Tesseract**, procuram verificar a sua confiança na deteção com o vocabulário conhecido, o qual pode não coincidir com a época de produção do documento; entre outras.

Estas dependências são especialmente notórias quando se envolvem documentos mais antigos, os quais podem, além de apresentar envelhecimento causado pelo tempo e danos pelas condições de armazenamento, devido às limitações tecnológicas assim como por vezes à falta de convenções de formatação dos documentos, não dispor de uma consistência no formato e texto (estrutura, alinhamento, dimensões dos caracteres, fonte de texto consistente, etc.) usual nos documentos atuais. Estes fatores resultam então num reconhecimento de texto não tão satisfatórios.

Estes documentos antigos são mais comumente, mas não exclusivamente, reconhecidos como anteriores à era digital, sendo que o foco de trabalho será maioritariamente dirigido a documentos desta

época, como jornais, revistas e outros, do século passado ou anteriores. Em especial documentos com estruturas complexas, como é o caso de jornais, onde é possível a segmentação em diferentes partes com conteúdo e propósito distinto e, ao mesmo tempo, uma ordem de leitura complexa i.e., não segue apenas regras simples de posição do conteúdo (texto da esquerda antes do texto da direita e cima antes de baixo), exigindo também noção das características e relação do conteúdo.

Mesmo para ficheiros do tipo **hOCR** ou **PDF**, que já passaram por um processo de reconhecimento de texto, a complexidade da estrutura dos documentos originais ou problemas nos elementos que contém o texto (como por exemplo elementos sobrepostos ou que se intersetam) dificultam a extração e interpretação do seu conteúdo, podendo ser facilmente perdida a lógica original.

A base desta tese será então a criação de um Toolkit, ou conjunto de métodos independentes, que habilite a manipulação e análise de dados envolvidos no processo de OCR, melhorando o resultado final da transcrição.

Complementando esta componente, duas outras serão implementadas através duma sua aplicação: uma pipeline de OCR, permitindo a utilização direta do Toolkit e assim observar a sua efetividade; um editor de resultados OCR que, aplicando diferentes métodos do Toolkit assim como da Pipeline, permitia uma minuciosa manipulação dos resultados, assim como teste do trabalho desenvolvido.

O presente documento pretende então servir como um estudo dos desafios apresentados por estes tipos de documentos perante **OCR**, assim como a procura e implementação de soluções para a melhoria dos resultados na deteção e extração de texto, resultando numa ferramenta que torne este processo mais simples e fiável.

Todo o trabalho desenvolvido estará disponibilizado nos repositórios: [OSDOCR](#) (repositório geral) e [document\\_image\\_utils](#) (tratamento de imagem).

## 1.2 Objetivos

O principal objetivo deste trabalho é a realização de um estudo sobre os problemas apresentados à extração de conteúdo de documentos de estrutura complexa - mantendo a sua lógica original -, assim como a implementação de uma solução para resolver ou mitigar estes desafios, aumentando a confiança na informação extraída. Em termos dos casos alvo do trabalho, será prioridade o estudo de jornais com texto máquina. Tal deve-se ao facto de jornais serem um particular tipo de documento que apresenta mais dificuldades e se encontra em maior procura de soluções e, texto máquina por ser mais comum para este tipo de documento. Esta segunda restrição é menos relevante pois não é uma dificuldade do

trabalho e pode ser resolvida perante a escolha da tecnologia de reconhecimento utilizada.

Especificando, os objetivos do trabalho são:

- Estudar os diferentes softwares de **OCR** disponíveis e as diferenças entre estes.
- Estudar as dificuldades que documentos podem apresentar no processo de reconhecimento de texto.
- Estudar o trabalho desenvolvido sobre a área de tratamento de imagem, identificação de tipo de documento, segmentação de documentos, algoritmos de cálculo da ordem de leitura, melhoria de resultados de **OCR** e métricas de validação de resultados **OCR**.
- Estudar trabalhos com âmbito similar ou relacionado ao presente.
- Estudo da filosofia da solução e desenho da arquitetura da mesma.
- Implementação de uma ferramenta de comando de consola que aplique uma pipeline, configurável, cujo input será um ficheiro - imagem, json, hOCR -, capaz de tratar de problemas deste para melhorar os resultados de **OCR** e, por fim, gere um output que mantenha a lógica e conteúdo do documento original.
- Implementação de um editor gráfico que permita, além de manipular resultados de **OCR**, visualizar e, consequentemente, depurar as outras componentes da solução.
- Avaliar os resultados da solução implementada.

Englobando todo o trabalho prático desenvolvido, a solução implementada será nomeada "Old Structured Documents OCR"(**OSDOCR**).

### 1.3 Estrutura da dissertação

Esta dissertação segue a seguinte estrutura:

- Capítulo 1: Breve contextualização sobre o tema proposto, as dificuldades impostas por documentos estruturados com digitalizações ou condições físicas degradadas nos resultados **OCR**, e a utilidade de uma ferramenta para o tratamento destas. Além disso foram listados os objetivos do trabalho.
- Capítulo 2: Estudo sobre o estado da arte nos tópicos relacionados ao tema da dissertação, as suas dificuldades e soluções destas; estudo de trabalho anteriormente realizado com âmbito similar ao atual ou técnicas relevantes para a construção da solução do problema.
- Capítulo 3: Discussão sobre o problema e desenho da solução e propostas desta.

- Capítulo 5: Discussão da implementação da componente de Toolkit da solução.
- Capítulo 6: Discussão da implementação da componente de Pipeline da solução.
- Capítulo 7: Discussão da implementação da componente de Editor da solução.
- Capítulo 8: Discussão e análise dos resultados das diferentes componentes da solução.
- Capítulo 9: Reflexão sobre o trabalho realizado, os resultados da solução como um tudo, e a experiência obtida, assim como uma breve exploração de caminhos para trabalho futuro do projeto.

## **Capítulo 2**

### **Estado da arte**

Neste capítulo, será feita uma exposição do estado da arte das tecnologias relacionadas com o tema ou relevantes para o projeto, assim como trabalhos relacionados, quer no mesmo tema ou envolvente - algoritmos relevantes para o desenvolvimento -, procurando plantar uma base para o trabalho realizado e futuro, entendendo o que já foi explorado e o que está para vir em alguns casos. O capítulo começa com uma apresentação sobre **OCR** que será a tecnologia pilar do trabalho (2.1), seguido por uma exploração de processos de melhoria dos resultados de reconhecimento usando pré (2.2) e pós processamento (2.3). Procede-se o tema de segmentação de documentos (2.4), terminando com o estudo de trabalho relacionado (2.5).

#### **2.1 Reconhecimento ótico de caracteres**

##### **2.1.1 Introdução**

O reconhecimento ótico de caracteres é a tecnologia base do projeto proposto, estando presente em qualquer instância ou caso de estudo que será explorado, inclusive em exceções que não necessitam a aplicação de reconhecimento de caracteres, como ficheiros do género **hOCR**, pois estes já são um produto de **OCR**.

Na sua essência e como o nome indica, software de reconhecimento ótico de caracteres permitem a deteção e transcrição de texto a partir de imagens, de forma automática e autónoma. Utilizando esta habilidade, abriu-se a possibilidade de tornar os documentos digitalizados ao longo do tempo numa fonte mais útil de informação: navegada, consultada e editada mais facilmente, visto estes serem na maioria dos casos, digitalizados na forma de imagens. A adição do conteúdo destes documentos através da sua transcrição, mesmo que apenas parcialmente correta, permite a adição de, por exemplo, meta-dados ou palavras chaves que auxiliam a sua indexação.

## 2.1.2 Breve história e evolução

Srihari et al. [2003] e Berchmans and Kumar [2014] apresentam a história do reconhecimento ótico de caracteres desde a conceção do seu ideal no século XIX, como uma tecnologia para auxílio de pessoas com impedimentos na leitura, até aos pontos alcançados na última década onde até escrita humana se tornou num desafio, até certo ponto, conquistável. As primeiras instâncias de reconhecimento óptico realizado por máquinas deu-se no final do séc. XIX, mais especificamente em 1870 por Charles R. Carey com a criação de um scanner de retina, mas é necessário ir até meio do século seguinte e pela consequente evolução que decorreu nesta área, para a subárea de reconhecimento de caracteres começar a ver a sua comercialização com a invenção de David Shepard: GISMO, um sistema simples capaz de reconhecer texto .

A génese desta tecnologia começou num formato bastante limitado, sendo capaz apenas de reconhecer um conjunto muito limitado de caracteres de uma fonte específica a um ritmo de 1 carácter por minuto, isto em condições de input bem controladas (papel sem ruído, apenas com o texto a ser reconhecido). Esta é considerada por Berchmans and Kumar [2014] como a primeira geração de **OCR**.

A segunda geração começa a dar os primeiros passos no processamento de escrita humana, como é exemplo o *IBM 1287* na década de 60.

A terceira geração, nas décadas de 70 e 80, introduziu um maior foco no processamento da escrita humana e na capacidade de lidar com problemas na imagem original.

A quarta geração tornou-se capaz de tratar documentos complexos com misturas entre texto e imagens, assim como qualidades de inputs menos favoráveis, documentos com cor e mais precisão com texto manuscrito.

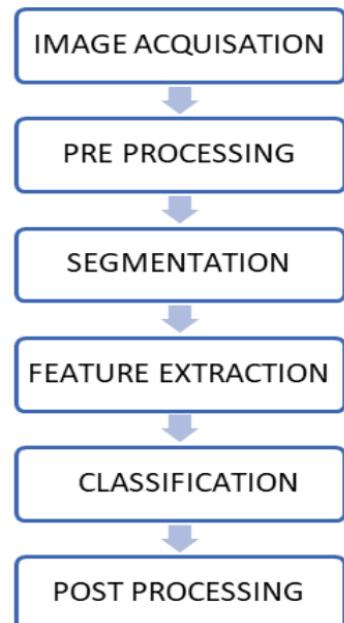
Atualmente com a evolução das técnicas de pré processamento, assim como os algoritmos de reconhecimento e a ascensão da inteligência artificial [Mittal and Garg, 2020], a precisão e flexibilidade dos softwares de **OCR** são capazes de, até em imagens de paisagens, segmentar e reconhecer texto localmente de forma automática e com pouco pré processamento. Além disso, embora o foco anteriormente era em software **OCR** pago e dedicado a um tipo específico de documentos, a implementação de softwares mais geral e de uso aberto tem-se tornado mais vulgar. Em algumas instâncias complexas - documento complexo e linguagem com caracteres fora do latim -, já existe tecnologia capaz de obter taxas de acerto acima dos 95% mesmo para texto escrito à mão [Mittal and Garg, 2020].

### 2.1.3 Processo OCR

Um software **OCR** pode ter reconhecimento online ou offline [Srihari et al., 2003][Berchmans and Kumar, 2014]. O primeiro é reconhecimento em tempo real, em que usualmente o input é obtido num dispositivo dedicado como um tablet digitalizador, no formato de um conjunto de coordenadas, podendo portanto ser mais preciso a custo de menor flexibilidade na entrada. O mais comum, método offline, recebe como um input por norma uma imagem com o documento finalizado. O bitmap desta imagem será utilizado como alvo do reconhecimento de caracteres. O uso deste último método, com tipo de entrada menos controlado, exige uma fase de pré processamento mais minuciosa do que o reconhecimento online.

Neste trabalho, o foco será dado ao reconhecimento offline por ser o mais comum e aquele que permite o tratamento de documentos pré digitalizados. Este pode ser geralmente dividido em 6 partes:

- **Aquisição de input** : imagem a ser reconhecida, incluindo algoritmos de compressão do próprio formato guardado.
- **Pré processamento** : técnicas de manipulação do input para melhorar resultado de **OCR**
- **Segmentação** : segmentação do input, a vários níveis, de modo a isolar o melhor possível os conteúdos relevantes, i.e. o texto.
- **Extração de características** : processo de reconhecimento de características dos caracteres isolados.
- **Classificação** : utilizando as características calculadas é feita a decisão sobre a sua identidade.
- **Pós processamento** : técnicas para melhoria do resultado como, por exemplo, a correção de erros ortográficos. Por vezes pode alterar o documento original se a **ground truth** já contiver estes erros.



O **Pré Processamento** é um passo essencial para o aumento do acerto do reconhecimento de texto, sendo que ele pretende remover imperfeições do input como: baixo contraste das linhas, texto mal delimitado, ruído de imagem, orientação do documento ou do texto (principalmente manuscrito). Em alguns casos mais complexos, com ajuda de inteligência artificial, também é possível a reposição de partes parciais de uma imagem que foram perdidas, ou remoção de elementos como **watermarks**.

A **Segmentação** é usada para isolar o conteúdo útil do resto da imagem podendo envolver vários passos como: segmentação da página para separar texto do resto do conteúdo; segmentação de caracteres, com o intuito de os separar em caracteres individuais, algo que é especialmente difícil com escrita à mão devido à tendência em criar ligações entre caracteres ou mesmo de os unir; tratamento e normalização dos caracteres isolados - normalização do tamanho, filtração morfológica.

A **Extração de Características** (Feature Extraction) trata-se do processo de deteção e cálculo das características dos caracteres, para a criação do classificador (dependendo da arquitetura) e anotação do que distingue o caráter alvo. Este processo é possivelmente o mais aberto para variações e que, juntamente com o classificador, mais influencia o resultado. Diferentes técnicas de extração de características e **Classificação** são utilizadas e foram estudadas durante as últimas décadas: desde *template matching* [Srihari et al., 2003] onde são usados algoritmos para cálculo de similaridade entre um template e o alvo, a segmentação de características, como presença de loops ou traços verticais longos [Srihari et al., 2003], ou distribuições de pixels [Mittal and Garg, 2020]. Para texto humano, este processo torna-se ainda mais complexo devido à necessidade de lidar com múltiplos caracteres invés de singulares. A classificação passava por um processo de comparação do valor das características calculado com diferentes templates porém, mais recentemente, o uso de estratégias no ramo de machine learning são mais comuns: redes neurais, support vector machines e k- nearest neighbor; são alguns dos modelos mais utilizados [Mittal and Garg, 2020] Berchmans and Kumar [2014]. Por vezes, o classificador utiliza conhecimento do léxico de uma linguagem para ajudar na sua classificação, sendo que documentos com linguagem desatualizada poderão sofrer nesse caso.

O **Pós Processamento** é responsável pelo tratamento do output, responsável por mitigar ou corrigir alguns erros do reconhecimento, desde correções ortográficas a posicionamento na página [Mittal and Garg, 2020].

Este trabalho irá ter como foco principal as secções de pré e pós processamento, e segmentação, na procura de aumentar a eficácia do reconhecimento e da organização dos resultados.

## 2.1.4 Desafios

Com a evolução da tecnologia, os problemas foram mudando de foco, tendo passado por um longo período em que a maior prioridade era a capacidade de reconhecimento de caracteres para além de um escopo limitado, tanto em termos de identidade quanto estilo, para a capacidade de tratar a imagem de forma a que o reconhecimento tenha uma maior taxa de acerto [Bieniecki et al., 2007]. Alguns dos maiores desafios atualmente para **OCR** são:

- **documento original** : danos no objeto; texto ilegível ou com um tipo de letra muito complexo; linguagem desatualizada; estrutura complexa; inclinação do texto; distorções da página.
- **imagem** : má iluminação; múltiplas páginas com diferentes orientações; baixa resolução; pouco contraste; ruído.
- **classificador ou extrator de features** não adequado para uma dada linguagem.
- **resultado** : validação quando não se tem a **ground truth** disponível

Dentro destes, o processamento de estruturas complexas será o foco principal e o expectável maior contributo deste trabalho.

### 2.1.5 Tecnologia

Presentemente, com a proliferação permitida pela internet e a globalização, a disponibilização de ferramentas de **OCR**, anteriormente primariamente privilégio de instituições ou empresas, como bancos [Srihari et al., 2003], tornou-se trivial, acessível através de itens do dia a dia como um computador ou telemóvel de forma gratuita, ex.: Google Lens.

Alguns destes softwares que serão utilizados neste trabalho são:

- **Tesseract**
- **Keras-OCR**
- **PaddleOCR**

Os resultados deste tipo software podem ser genericamente descritos como uma lista de caixas, delimitadoras de texto, com conteúdo, i.e. o texto nela contido e, por norma, um nível de confiança no reconhecimento desse texto.

No caso do **Tesseract** [Tesseract], dentro das várias formas que os resultados podem ser apresentados, a lista de caixas pode ser interpretada como uma árvore de blocos, em que cada nível corresponde a um tipo de estrutura no documento: página → bloco → parágrafo → linha → texto.

Usando **PaddleOCR** [PaddleOCR], os resultados são mais simples, divididos apenas pelas linhas de texto detetado.

Já o **Keras-OCR** [KerasOCR] lista um conjunto de caixas em que cada contém uma palavra reconhecida.

Uma outra característica que o **Tesseract** tem é a capacidade de reconhecer, com nível de acerto variável, outros elementos relevantes de um documento, como imagens ou delimitadores. Isto pode, por outro lado, causar erros na interpretação dos resultados por sobreposição ou multiplicação da quantidade de caixas. Além disso o **Tesseract** permite bastantes configurações como: léxico esperado; modo de segmentação; reconhecimento de espaços em branco; etc.

O output deste tipo de software pode ainda ser processado para tomar diversas formas: formatos que apenas retêm o conteúdo como texto simples ou markdown; formatos que mantêm informação sobre os blocos detetados, como hOCR.

A validação do output é na maioria dos casos medida a partir da comparação com a **ground truth**, o que limita a capacidade de testar e treinar (no caso de **ML** supervisionado) modelos visto que os datasets tem de ser criados de forma minuciosa e consumidora de tempo.

Além dos softwares de reconhecimento, é preciso ter atenção ao tipo dos ficheiros de entrada. Estes são usualmente imagens e, dependendo do tipo de **codec** destes, os algoritmos de compressão aplicados poderão diminuir a qualidade de imagem, como é o caso de formatos *lossy* como **JPEG**, potencialmente diminuindo o acerto do reconhecimento do texto. [Darwiche et al. \[2015\]](#), no seu estudo demonstra que mesmo entre diferentes tipos de *lossy* **codec** o seu impacto pode variar significativamente nos resultados de **OCR**, sendo que o formato **JPEG**, um dos mais populares, resultou nas menores taxas de sucesso.

## 2.2 Pré Processamento para OCR

Como foi listado na secção [2.1.4](#), existe uma diversa quantidade de defeitos que os documentos originais e as imagens digitalizadas destes podem ter e cuja presença pode afetar negativamente os resultados de software de **OCR**. O pré processamento pode ser considerado como uma fase de tratamento de imagem para remover estes problemas que deterioram o reconhecimento de texto. De forma a entender os diferentes métodos utilizados e a sua evolução para o presente, foram selecionados os estudos: [\[Bieniecki et al., 2007\]](#),[\[Likforman-Sulem et al., 2009\]](#),[\[Souibgui and Kessentini, 2022\]](#),[\[Lat and Jawahar, 2018\]](#),[\[Dey et al., 2022\]](#),[\[Wei et al., 2018\]](#),[\[Bui et al., 2017\]](#).

Entre o grande leque de diferentes algoritmos e tratamentos que podem ser aplicados nas imagens, em geral, estes podem ser segmentados nos mais comuns [\[Dey et al., 2022\]](#):

- **binarização/thresholding da imagem** : processo de normalização dos pixeis para um de dois valores, mediante um determinado limiar
- **remoção de ruído** : algoritmos para retirar degradações ou sujidades da imagem através de

processos como, por exemplo, suavização da imagem calculando para cada pixel o valor médio da sua vizinhança

- **correções de texto** : alguns casos deste são texto que apresenta um ângulo de rotação, texto com inclinação, distorções locais no texto, *watermarks*
- **super-resolução** : aumentar a resolução da imagem, consequentemente aumentando o seu **DPI**
- **foco da imagem** : acentuação das arestas, diminuir desfocagem
- **transformações morfológicas** : operações sobre a imagem de modo a provocar maior contraste do conteúdo, ou permitir melhor distinção das características, ex.: dilatação do texto para tornar mais fácil a distinção entre regiões com e sem texto.

Para **binarização**, o objetivo principal é distinguir o texto do resto da imagem, daí a alocação dos pixels para 1 de dois valores. Este processo é distinguido principalmente entre o uso de *thresholding* global ou local (ou adaptativo), sendo que o global implica um cálculo das características estatísticas locais dentro da imagem, e é mais adequado para o tratamento de imagens com cor, ou com variações de intensidade dispersas pela imagem [Dey et al., 2022]. Alguns dos algoritmos mais comuns são comparados por Souibgui and Kessentini [2022], onde é evidente a dependência destes nas condições do documento original e da imagem. Um exemplo apresentado demonstra como, numa imagem com uma mancha escura (com o texto ainda distinguível), o algoritmo de Otsu conseguiu gerar uma imagem com pouco ruído e bom contraste, mas a zona da mancha fica completamente preta, comparado ao algoritmo de Niblack que, embora com mais ruído, recuperou algum texto dentro da mancha.

A **remoção de ruído** é possivelmente a área com mais alternativas possíveis e das que mais afeta o resultado do reconhecimento por, a nível de pixels, o ruído interferir com a composição dos caracteres ou criar acumulações de informação extra que serão mal identificadas pelos softwares, como notado por Bieniecki et al. [2007]. O ruído nas imagens pode ser de vários tipos, o que dificulta a forma de o detetar e tratar. Alguns dos outros tratamentos, como a binarização, transformações morfológicas e alguns tipos de super-resolução, também tratam desta questão mesmo não sendo o seu foco principal. Da mesma forma, alguns dos filtros utilizados podem ter outros resultados como o aumento do contraste ou eliminação de distorções. Alguns dos tratamentos mais comuns do ruído são [Dey et al., 2022][Bui et al., 2017][Bieniecki et al., 2007]:

- filtro Gaussiano
- médias não locais

- suavização com filtro de mínimos locais
- suavização com filtro Wiener

Vários destes métodos resultam tanto na acentuação das arestas do texto, como na remoção de lixo ou ruído à sua volta, recuperando o **foco da imagem**.

A **correção de texto** necessita, ao contrário dos outros processos que podem, mesmo sem uma análise prévia do estado da imagem, melhorar o reconhecimento de texto; de uma análise prévia visto que, por exemplo, não se pode aplicar uma rotação na imagem sem saber o ângulo de orientação inicial desta. [Bieniecki et al. \[2007\]](#) faz uma apresentação convincente do efeito de uma rotação de ângulo 15° no reconhecimento do Tesseract, o que impediu o reconhecimento. O método proposto passa pelo computação de uma linha que afete a margem na extremidade esquerda do texto, de modo a calcular a sua inclinação relativa à margem da imagem e assim descobrir o ângulo de rotação do texto. No espectro mais limitado da sua proposta, devido à sua localidade nos documentos, [Bieniecki et al. \[2007\]](#) discute distorções nos documentos como curvaturas resultantes da bainha de um livro. Aqui, em traços gerais, a linha de curvatura do texto é detetada, com a qual é criado um quadrilátero da área afetada, onde será, de acordo com o nível de curvatura na projeção sobre a linha, realizada a correção. Em ambos os casos, os resultados demonstrados para casos de grande deformação, os algoritmos propostos conseguiram tornar a completa falha de reconhecimento para taxas de acerto dentro dos 99%.

A aplicação de **super-resolução** procura auxiliar o processo de reconhecimento ao melhorar a qualidade de imagens de baixa resolução, i.e. aumentar os seus **DPI** e tornar os caracteres mais reconhecíveis. Entre os vários algoritmos utilizados para este propósito, o uso de interpolação tendia a ser o mais comum, porém nem sempre os resultados eram satisfatórios, resultando em imagens transformadas serem desfocadas, ou com os defeitos originais acentuados, especialmente quando o salto era feito a partir de imagens com **DPI** baixo - 100 ou menos [[Lat and Jawahar, 2018](#)]. No entanto, com os avanços na área das redes neurais, em particular na categoria de imagens naturais, modelos como **CNN** [[Wei et al., 2018](#)],[[Lat and Jawahar, 2018](#)] trouxeram uma nova forma treinar algoritmos para tratar imagens de forma adaptativa e com resultados muito melhores do que os algoritmos bem estabelecidos para este problema. Um dos pontos negativos deste tipo de redes é que a criação de datasets de treino é um processo demorado, sendo que para cada imagem de treino (degradada), é necessário uma imagem par com o resultado ideal para validação do resultado. Adicionalmente estes datasets têm de ter casos com características dispersas o suficiente para permitir uma boa generalização do modelo. Um outro modelo que tem vindo a emergir são as **GAN** que, invés de utilizarem uma única rede para gerar conteúdo que depois será validado em cada iteração do treino, utilizam duas redes que competem diretamente: a geradora que

tenta transformar imagens de modo a enganar o discriminador, e este que tenta entender se a imagem de input é a imagem original ou se foi gerada. [Souibgui and Kessentini \[2022\]](#) propõe um modelo deste género que demonstra a sua superioridade tanto em relação a algoritmos baseados em regras, como de modelos baseados em **CNN**.

As **transformações morfológicas** são compostas por vários métodos e propósitos diferentes, nem sempre com o intuito de melhorar a qualidade da imagem, mas para acentuar certas características desta. Por exemplo, técnicas como a dilatação podem ser utilizadas para acentuar regiões de texto de forma a ser possível separar o texto do resto. Por outro lado, técnicas de deteção de arestas, erosão ou *thinning*, diminuem o tamanho dos elementos da imagem, podendo simplificar os caracteres, tornando o seu reconhecimento, ou das suas características (como loops) mais evidentes [[Dey et al., 2022](#)].

Estes diferentes tipos de tratamento podem, na grande maioria dos casos, complementar-se mutuamente e, é costume - inclusive nos estudos referenciados - a criação de pipelines de pré processamento que aplicam estes vários tratamentos de forma sequencial. No entanto, como estes diferentes tratamentos impactam diretamente os dados de input para reconhecimento, nem sempre são benéficos e têm de ser escolhidos com cuidado consoante o estado do sujeito. [Bui et al. \[2017\]](#) demonstra precisamente isto mostrando, por exemplo, que a aplicação de um filtro Gaussiano para a redução de ruído num caso de teste reduziu a taxa de acerto do Tesseract para menos de 1 terço comparado ao resultado sem pré processamento. Isto naturalmente dificulta a criação de pipelines automáticas de pré processamento. Nesse mesmo estudo, é proposto o uso de uma **CNN** que, consoante um número limitado de classes que representam combinações de técnicas de pré processamento, decide a melhor para uma dada imagem. Esta solução resultou numa melhoria considerável, principalmente para o reconhecimento do Tesseract e, mais interessante, a tendência para certas combinações de técnicas com: binarização escolhida 90% das vezes, redução de ruído 35% e acentuação de contrastes 34%. Como mencionado anteriormente, os avanços no tratamento de imagem com uso de modelos de Deep Learning vêm trazer, quando suficientemente generalizados, um método ubíquo para a realização destes vários tratamentos de forma adaptativa. [Souibgui and Kessentini \[2022\]](#) com a **GAN** proposta, demonstra resultados no tratamento de ruído, focagem, binarização e remoção de *watermarks* excelentes, mesmo tendo em conta que o foco principal do modelo era o aumento da resolução da imagem original.

## Métricas de avaliação

No ato de pré processamento, as métricas de avaliação são muitas das vezes subjetivas visto, em geral, se tratar de tratamento de imagem e nem sempre haver uma versão não degradada das imagens dos

documentos. No caso de haver essa versão prística, algumas das métricas mais comuns para testar o tratamento de modelos ou algoritmos são: **PSNR**, que compara o ruído na imagem tratada comparativamente com a original, sendo que valores maiores tendem a significar melhores resultados; e **SSIM**, que tenta ter em conta as similaridades das vizinhanças na imagem, assim como outros aspectos mais relativos a cor e luminosidade, imagens idênticas terão valor 1. Não havendo a possibilidade de testar com uma imagem base, pode-se avaliar o efeito do pré processamento através da variação dos resultados do output ou do pós processamento.

## 2.3 Pós Processamento para OCR

Na generalidade, o tratamento dos resultados de **OCR** ronda em torno das correções sob o texto resultante. Estas correções procuram corrigir erros ortográficos, texto irreconhecível, ou sem sentido (caracteres lixo ou ruído reconhecido).

Correções a nível dos blocos/caixas que englobam o texto reconhecido, são mais orientadas ao tipo de documento e ao seu contexto e serão analisadas com mais atenção nas secções seguintes.

[Nguyen et al. \[2021\]](#) apresentam um estudo extremamente comprehensivo e extenso sobre o estado da arte e o impacto do pós processamento no texto resultante de **OCR**.

Neste estudo, é apresentado primeiramente a importância deste tratamento de texto, não só para aumentar a qualidade das aplicações que o utilizam, exemplo dado no caso de **NLP**: onde taxas de erro por volta dos 7% podem mostrar reduções na qualidade da análise de sentimento de até 30%; mas também no próprio processo de navegação e procura por documentos transcritos por **OCR**: em alguns exemplos os erros de texto não permitiram uma indexação ou reconhecimento de termos de pesquisa correta, não sendo devolvidos na procura.

Os dois principais erros de texto reconhecido são:

- **não palavra** : quando uma palavra reconhecida pelo motor de **OCR** não se encontra no léxico conhecido
- **palavra real** : a palavra reconhecida pertence ao léxico conhecido, porém difere da **ground truth**

Dentre estes dois tipos de erro, o primeiro é consideravelmente mais fácil de detetar e potencialmente corrigir, visto o segundo necessitar de informação adicional, quer seja esta a **ground truth** do documento - o que é raro -, ou uma análise da palavra dentro do seu contexto.

O estudo segue então para a secção das técnicas de pós processamento. Estas são separadas em dois tipos principais: **manuais** e **(semi-)automáticas**.

As técnicas **manuais** entendem total ação humana e são normalmente dirigidas para projetos mais sensíveis a erros mas que, pela necessidade desta mão de obra, são naturalmente mais custosos, demorados e raros. São casos destes, projetos de transcrição de documentos antigos, como é dado exemplo o projeto da biblioteca nacional da Austrália na correção de jornais históricos. Alguns outros casos destas técnicas descritos servem mais para o propósito de avaliação de algoritmos ou criação de casos de teste.

As técnicas **(semi-)automáticas** podem ser agrupadas em dois tipos: **tratamento de palavras isoladas**, e **dependentes de contexto**. Dentro destas, o tratamento de palavras isoladas é focado na correção de problemas de 'não palavra', enquanto as dependentes de contexto procuram resolver os dois tipos de problemas.

Dentro das diferentes técnicas baseadas nas **palavras isoladas**, algumas características servem como fundamentos dos algoritmos:

- **Léxico conhecido**
- **Confiança do reconhecimento**
- **Frequência de utilização de uma palavra, no documento, ou globalmente**
- **Similaridade da palavra errada com as conhecidas no léxico**

Entre algumas destas técnicas, são realçadas:

### **Junção de Outputs de OCR**

A junção de outputs de OCR visa a escolher entre diferentes resultados para uma dada sequência de palavras, com características distintas (nível de confiança no reconhecimento, quantidade de erros,etc.), e escolher dentro destas ou numa sua mistura, o output final.

Numa 1º fase, os outputs são então obtidos, onde para isto várias propostas foram feitas, com as principais sendo:

- Usando o mesmo motor OCR, fazer vários reconhecimentos de um mesmo trecho de texto
- Usando o mesmo motor OCR, fazer vários reconhecimentos de um mesmo trecho de texto, com parâmetros diferentes ou tratamento de imagem diferente
- Usando múltiplos motores OCR, fazer vários reconhecimentos de um mesmo trecho de texto

Numa 2º fase, estes diferentes outputs têm de ser alinhados de forma a poderem ser comparados palavra a palavra. Para isto, algoritmos sob grafos são comuns.

Por último, utilizando um decisor, o output final é escolhido. Este decisor pode tomar várias formas como: modelos de votação, cálculo de similaridade com léxico, modelos [LSTM](#).

Embora esta técnica geralmente resulte em resultados melhores do que o reconhecimento simples, exige um maior gasto computacional, assim como do facto de estar limitado ao dicionário conhecido.

## **Vias lexicais**

Uma outra visão sobre o tratamento do texto, é na procura das palavras mais similares à não palavra detetada e, dentro destas, decidir qual a que tem maior potencial para a substituir. Este cálculo de similaridade pode ser realizado de várias formas, sendo das mais comuns: a distância de Levenshtein, onde se calcula o número de operações mínimas - inserção, remoção ou substituição - a realizar numa palavra para obter outra; variações deste algoritmo; e distância entre n-gramas, que envolve a quantidade de conjuntos de palavras em comum entre as duas palavras comparadas.

Como no caso anterior, estas vias continuam limitadas pelo léxico conhecido, sendo que muito do estudo é dedicado à criação de dicionários mais abrangentes ou adaptados ao documento, ex.: pegando em palavras chaves do documento ou de um tema e criar um dicionário com as páginas mais relevantes de uma pesquisa feita sobre estes.

## **Modelos de erro e Máquinas de estado finitas com pesos**

Os modelos de erro procuram calcular as probabilidades sob as operações nos caracteres da palavra errada e, a partir destes e do léxico conhecido, decidir qual o melhor candidato para substituição.

Estes modelos de erro podem ser complementados por máquinas de estados finitas com pesos. Os modelos de erro são utilizados para a escolha de candidatos para substituir o erro, e os pesos da máquina são dependentes de características entre os candidatos e a palavra errada como: comprimento, semelhança, entre outras.

## **Modelos de linguagem baseados em tópicos**

No decisão de candidatos para substituição da palavra errada, outros trabalhos sugeriram o uso de contexto do documento de forma parcial, i.e. calcular o tópico do documento a partir da análise deste e utilizar esta informação como um variável extra nas fases de decisão de candidatos. Assim, palavras que sejam numa perspetiva global mais raras não serão tão facilmente descartadas como nos métodos anteriores.

Tal envolve no entanto um processo de decisão sobre quais os tópicos que existem, e a adaptação do léxico para criar correspondências entre as palavras e estes dados tópicos.

Dentro dos métodos **dependentes de contexto**, são notados os ramos:

### **Modelos de linguagem**

Partindo como base os modelos anteriormente descritos, complementam os modelos com o cálculo da probabilidade de distribuição de sequências de palavras, sendo estas parte do documento. Assim, para cada palavra, utilizando os seus vizinhos, será calculada a probabilidade daquela sequência ocorrer. Este cálculo pode ser calculado utilizando léxicos já definidos, ou complementando estes com as frequências dos n-gramas de palavras dentro do documento. Estes modelos caem dentro do ramo estatístico.

Um outro tipo é conseguido através de redes neurais que a partir do texto criam **word embeddings**, o que permite calcular a similaridade entre palavras tendo em conta as suas características. Com esta habilidade, as sequências de palavras do texto reconhecido podem ser sujeitas ao cálculo da probabilidade de ocorrerem e, caso este seja muito baixo, poderá ser sinal de um erro de palavra real.

### **Machine Learning baseado em características**

Neste caso, o contexto é utilizado dentro de uma quantidade de características mais limitado mas também mais robusto do que na alternativa anterior. Algumas destas características tendem a ser:

- Frequência da palavra - nos casos de treino e no próprio documento
- Frequência dos n-gramas com a palavra - nos casos de treino e no próprio documento
- Peso de confusão - conseguido através dos casos de treino
- Confiança do motor OCR na palavra

### **Seq2Seq - Sequência para Sequência**

Esta alternativa, tem como noção que este problema de correção é uma questão que pode ser resolvida por tradução máquina, correspondendo à transformação numa sequência de palavras, numa outra idêntica ou semelhante, na mesma linguagem.

Estes modelos, ao contrário dos mencionados de modelos de linguagem - que recebendo uma sequência de palavras analisavam a probabilidade de uma outra ser a próxima na sequência, ou sugeriam a próxima palavra -, recebem uma sequência de palavras e devolvem também uma sequência de palavras.

O estudo termina com uma análise das tendências destas diferentes áreas, onde se pode notar uma aderência maior para tecnologias de inteligência artificial, juntamente com a tendência para a união dos dois ramos de tratamento de texto (semi-)automático nas soluções.

### Métricas de avaliação

Como acontece no caso do pré processamento, o pós processamento necessita de uma **ground truth** para poder ser avaliado. Contra esta, diferentes medições podem ser feitas, como a percentagem de caracteres ou palavras totais nos dois textos, ou a taxa de acerto tendo em conta alinhamento dos textos. Algumas características como a quantidade de *whitespaces* e indentação também podem ser relevantes para certos tipos de documentos. A quantidade de substituições realizadas, mais propriamente de palavras não reais, também pode ser importante para avaliar o software de reconhecimento, embora este erro possa ser resultado de um léxico não apropriado para o documento, ex.: documentos históricos.

## 2.4 Segmentação de documentos

A segmentação de documentos é um processo que visa decompor o documento nas suas várias secções ou elementos. A sua aplicação pode variar dependendo do objetivo concebido: separação do texto de elementos não texto, retirando informação prejudicial para **OCR**; divisão do conteúdo do documento em várias secções para que possam ser analisadas ou extraídas isoladamente. Por este mesmo motivo, este processo tem aplicabilidade tanto antes como depois de feito o reconhecimento de texto.

[Eskenazi et al. \[2017\]](#) faz um estudo comprehensivo sobre as diferentes metodologias de segmentação de documentos, apresentando as suas diferentes características, evolução e tendências. As diferentes técnicas podem ser divididas de forma comum como:

- **top-down** : divisão a partir da página em blocos mais pequenos
- **bottom-up** : a partir de uma escala mais pequena, ex.: pixels, componentes conectadas; os elementos são aglomerados em conjuntos maiores até completar a página
- **híbrido**

O estudo decide invés seguir por uma divisão em 3 grupos de acordo com a evolução da capacidade dos algoritmos de segmentar diferentes tipos de documentos.

- **Dedicado a um esquema (layout) específico**

- **Capaz de lidar com layouts descritos por um conjunto de parâmetros**
- **Layout potencialmente não restrito**

### **Algoritmos dedicados a um layout específico**

Estes algoritmos, têm como objetivo a segmentação de um dado tipo de esquema. Estes tendem a ser mais rápidos e, embora os menos versáteis, apresentam para o seu tipo de alvo resultados difíceis de superar. Dentro destes, existem 3 ramos principais de algoritmos.

O primeiro, são os algoritmos que assumem as características do esquema e, usando estas, criam gramáticas que os descrevem, ou criam perfis do esquema que são projetados nas imagens de input para aplicação de heurísticas de alinhamento e análises probabilísticas para deteção de erros ou desalinhamentos

O segundo ramo, foca-se no uso de filtros para realce de regiões de um layout. Estes são normalmente utilizados para documentos em que as linhas do texto sejam retas e horizontalmente alinhadas. Os casos mais frequentes destes algoritmos aplicam morfologias, como sequências de erosão e dilatação de forma a identificar imagens, ou Run-Length Smoothing para a formação de manchas em áreas densas com conteúdo.

O último ramo, investiga cálculo das linhas que delimitam uma página de forma que uma segmentação em blocos ocorre naturalmente. Exemplos deste passam pela transformação das linhas do texto em linhas retas; criação de linhas delimitadores através do espaço em branco no documento; transformação de Hough para deteção de linhas.

### **Algoritmos que usam parâmetros para descrever um layout**

Estes algoritmos são mais flexíveis na sua capacidade em lidar com diferentes tipos de documento do que o grupo anterior. Este grupo trabalha com um conjunto de características dos elementos do documento, permitindo assim, com o uso deste contexto extra, mais versatilidade para lidar com irregularidades. Também este grupo pode ser dividido em alguns ramos.

O mais comum destes é o *clustering* onde, partindo de elementos base, como componentes conectadas, procura criar agrupamentos destes, representantes de elementos de ordem superior, como por exemplo imagens, segundo um conjunto de características destes elementos base: geométricas, de textura (distribuição dos pixels), cor, vizinhança, etc.. Algoritmos híbridos também são usuais, como o uso da informação de uma primeira segmentação em blocos antes de partir para o clustering.

Um outro ramo, trata de fazer o agrupamento de elementos no documento original a partir da otimização de funções de custo. São exemplos destes, algoritmos que iteram sobre a realização de segmentação numa página, onde se estima se a realização de novas segmentações diminui o custo da função.

Por último, e segundo mais popular, estão os algoritmos de classificação. Estes, a partir de um grupo de classes predefinido, pretende atribuir uma delas aos elementos do documento. Este ramo, ao contrário do clustering, é marcado por todos os algoritmos necessitarem de treino. Vários trabalhos deste género, realizam um processo inicial de decisão das melhores features utilizando **ML**.

### **Algoritmos para segmentação de layout potencialmente não restringidos**

As técnicas mais recentes de segmentação estudadas em [Eskenazi et al. \[2017\]](#), englobam um leque de projetos de junção de algoritmos antigos de forma híbrida ou combinada, e algoritmos utilizando redes neurais. Estes últimos são pouco abordados neste estudo, mas trabalhos recentes, não apenas na segmentação de documentos, mas também em termos de segmentação de imagem, tendem para a utilização de **CNN**. Exemplo deste é [He et al. \[2017\]](#), que usa múltiplas redes convolucionais para a segmentação de uma página entre texto e não texto, e dentro do não texto em figura ou tabela.

## **2.5 Trabalho relacionado**

Nesta secção serão estudados trabalhos cujo objetivo, ou orientação, se assemelha com os objetivos deste projeto, sendo portanto casos de estudo e inspiração relevantes. Os temas abordados são: extração de conteúdo de jornais ([2.5.1](#)), e cálculo de ordem de leitura ([2.5.2](#)).

### **2.5.1 Extração de conteúdo de jornais**

Como argumentado no primeiro capítulo, um dos tipos de documentos que mais sofre no processo de reconhecimento de texto são os jornais. Tal deve-se ao facto de estes terem estruturas complexas, interpoladas com imagens, anúncios, elementos de texto chamativos, porções de texto em conteúdos irregulares, e outros elementos invulgares que dificultam a criação de heurísticas ou treino de modelos para a sua análise e tratamento.

Nesta secção será feita uma análise de diferentes trabalhos na área de segmentação e extração de conteúdo de jornais. O objetivo da segmentação é geralmente o isolamento dos artigos do jornal.

Estes trabalhos podem ser divididos em dois tipos principais: baseado em **heurísticas**, ou utilizando **modelos de machine e deep learning**, maioritariamente **CNN**.

## **Heurísticas**

[Chaudhury et al. \[2009\]](#) foi um projeto proposto por elementos da Google que, embora não tão recente como outros, oferece uma visão sobre heurísticas generalizadas para uma grande quantidade de dados. Neste trabalho, a parte relevante do processo, i.e. depois da obtenção da imagem do jornal, passa primeiro por um tratamento da imagem, utilizando uma binarização local baseada em morfologias para reconstrução de gradiente cinzento, o que permite a identificação de um contraste entre o conteúdo do documento e o fundo, e consequentemente saturar o fundo. Em seguida, é realizada uma segmentação em blocos através das linhas e "valetas-trechos" do fundo que separam o texto. Depois, é realizada uma análise dos blocos para fazer uma classificação entre títulos e texto considerando o tamanho de fonte e proporção da área do bloco. Os títulos são considerados iniciadores de artigos. Por último, é feita o agrupamento dos blocos em artigos através de duas regras principais: título comum, onde blocos por baixo de um mesmo título fazem parte do mesmo artigo; bloco órfão, onde as exceções à regra anterior são tratadas, juntando blocos órfãos a blocos não órfãos que não tenham blocos por baixo deles e tenham a margem de baixo abaixo da sua margem superior.

O artigo admite acertos de 90% porém, não suporta estes resultados com números de casos de teste ou identificação de um dataset. Além disso, esta segmentação não tem em conta elementos não-texto nos jornais.

[Chatthuranga and Ranathunga \[2017\]](#) apresenta uma outra proposta focado numa primeira segmentação da página utilizando linhas calculadas através de tratamento de imagem, mas elabora no anterior através de uma extensão destas linhas baseado em regras e uma posterior análise da sua distribuição. Ao contrário do trabalho anterior, as imagens são consideradas na segmentação dos artigos e os resultados são apresentados em conjunto com a informação dos testes.

[Bansal et al. \[2014\]](#) propõe um método híbrido em que são utilizadas heurísticas e grafos para extração do contexto dos blocos, que depois serão classificados usando regressão. Como usual, começam com uma primeira fase de tratamento de imagem para a limpar - binarização, remoção de delimitadores, separar texto de figuras -, e segmentar os blocos em texto e não-texto. Numa segunda fase, para cada bloco é calculada a sua vizinhança, sendo que esta é calculada de acordo com um limite de profundidade de adjacência. Para a classificação de blocos e classificação de artigos a profundidade por eles usada é diferente, sendo superior no caso dos artigos. Por fim, os artigos são classificados através de um modelo que tem informação da sua vizinhança, assim como características geométricas do bloco. Este

grafo é denominado de modelo hierárquico de ponto fixo. Chathuranga and Ranathunga [2017] sugerem uma variação deste modelo utilizando um modelo de Markov de 2 dimensões que permite a retenção de possíveis ordens de leitura como contexto adicional.

## **Inteligência artificial**

Almutairi and Almashan [2019], Meier et al. [2017], Barman et al. [2021] demonstram o poder das **CNN** em tarefas de segmentação, sendo capazes de generalizar problemas como linguagens não ocidentais e blocos de conteúdo não retangulares. Esta última habilidade é conseguida através da aplicação de máscaras ao nível dos pixels invés dos blocos. Além da capacidade de extração de características (visuais) destes modelos, dependendo da sua arquitetura, eles podem ser treinados para realizarem técnicas de tratamento de imagem diretamente [Meier et al., 2017]. Barman et al. [2021] complementa a arquitetura das **CNN** mais genéricas, com a capacidade de analisar características sobre o contexto dos blocos através da modificação da arquitetura para computar **word embeddings** do texto reconhecido.

Por último, é importante realçar o esforço do projeto Europeana [Europeana] na educação e incentivo sobre o processo de digitalização de jornais históricos utilizando **OCR**.

### **2.5.2 Ordem de leitura**

No sentido de permitir outras estratégias de extração de conteúdo, passando pela reorganização deste à partida, ou das segmentações resultantes deste, esta secção irá abordar algumas estratégias de cálculo da ordem de leitura.

Na maioria dos documentos, considerando linguagens e estruturas que partilhem as características do português, a ordenação de leitura é relativamente trivial, podendo ser feita uma ordenação topográfica com base em regras geométricas simples como: um bloco está antes dos blocos diretamente por baixo dele; um bloco está antes de blocos à sua direita. Tal não é o caso para documentos que utilizam estruturas mais complexas, ou o contexto do conteúdo como guia da sua ordem de leitura, como jornais, revistas, tabelas, etc.

Este é um problema que está inherentemente ligado ao processo de segmentação de páginas, visto que este último é que provisionará os elementos que depois têm de ser organizados numa ordem de leitura. Dependendo da granulação da segmentação, múltiplos algoritmos de cálculo da ordem de leitura podem ser aplicados para cada nível, como seria o caso de ordenar os diferentes artigos num jornal, e posteriormente ordenar dentro de cada artigo os seus blocos de conteúdo.

[Breuel \[2003\]](#) propõe um algoritmo generalista para o cálculo da ordem de leitura de documentos, utilizando um ordenação topológica dos blocos com apenas duas regras: um bloco *a* está antes do bloco *b* se ambos estiverem horizontalmente alinhados (coordenadas x mais à esquerda e mais à direita sobrepõem-se nos dois blocos, tendo em conta uma certa folga) e *a* está acima de *b*; *a* está antes de *b* se *a* estiver completamente à esquerda de *b*, e não houver nenhum elemento verticalmente entre *a* e *b* que no seu comprimento englobe os dois. Nesta proposta, eles trabalham com blocos ao nível das linhas de texto, porém seria aplicável para blocos de segmentos de texto. Em termos de implementação e lógica, a proposta é bastante simples e competente na generalidade dos casos, porém, como mencionado na introdução da secção, a falta de consideração sobre o contexto impede que certos conflitos entre potenciais ligações de blocos sejam resolvidos da maneira correta.

Bandas desenhadas, tal como jornais, possuem uma estrutura muito variável na sua disposição, mas também na ordem de leitura correta, intendendo por vezes proporcionar experiências ou emoções ao leitor através do modo como o conteúdo é apresentado. São portanto, um bom caso de estudo para o tratamento de jornais. [Kovanen and Aizawa \[2015\]](#) implementaram um algoritmo dedicado a este mesmo tipo de entretenimento. A base da solução definida é o uso de grafos e a sua ordenação. Estes são usados para realizar duas ordenações diferentes, uma primeira sob os diferentes painéis de uma página, e um segundo sob as caixas de texto na página. O método de ordenação é simples, sendo novamente geométrico, usando o vizinho mais próximo. Esta simplicidade, é compensada tanto com uma segmentação que é proposta por eles e dedicada a este tipo de documento, e também pela dupla ordenação que, para cada caixa de texto, limita a quantidade de candidatos disponíveis de acordo com a ordem calculada dos painéis.

Numa abordagem não heurística, [Quirós and Vidal \[2021\]](#) utilizam **ML** para ordenação de documentos históricos com estrutura simples e regular, mas que incluem anotações no canto das páginas que alteram a ordem de leitura do texto, tornando-a mais irregular. A proposta passa pelo cálculo da probabilidade entre pares de blocos, que representam a sua hierarquia, por parte de um Multi Layer Perceptron. Embora os resultados sejam notáveis na generalidade do dataset proposto, eles notam dificuldades para páginas compostas por tabelas, onde a estrutura é mais complexa. Isto deve-se principalmente ao facto de estas serem uma minoria nos dados de treino. Realça-se aqui a possibilidade de adaptar este, e similares métodos de modelos inteligentes, a partir da criação de datasets dedicados a certos tipos de documentos.

## **2.6 Conclusões**

O estudo realizado para a produção deste capítulo, permitiu uma clarificação das bases relativas ao trabalho com motores de **OCR**, as boas práticas e procedimentos comuns no seu tratamento e melhoria, e os efeitos que, em especial, pré processamento de imagens e pós processamento de texto podem ter sob o resultado final. Além disso, na secção [2.5](#), o estudo de trabalhos com temática similar ou sobre técnicas relevantes para o projeto atual, permitiu uma melhor percepção sobre os fluxos da solução destes e, simultaneamente, entender os maiores desafios com que se deparam - solucionados e por solucionar. Além disso, realça-se que tal como em várias outras áreas tecnológicas, tem nos últimos anos havido uma imergência de soluções com recurso a Inteligência Artificial, com principal sucesso no ramo de tratamento de imagem.

O acumular deste estudo permitiu então um desenho da solução final e do plano de tarefas mais coerente e fundado nos produtos e evolução da área em que se insere, i.e. reconhecimento óptico de caracteres para extração de conhecimento.

## **Capítulo 3**

# **OSDOCR Filosofia**

Neste capítulo é realizada uma reflexão sobre o problema e as consequentes propostas que são deste retiradas. Serão descritas decisões e modelos fundamentais da solução, assim como a arquitetura desta.

### **3.1 O problema**

Na sua essência, o problema abordado pode ser descrito como a aquisição de resultados não satisfatórios na aplicação de OCR sob documentos antigos com texto estruturado.

Como se analisou no capítulo 2, estes resultados podem ser melhorados através de múltiplas interações em diferentes partes do processo do reconhecimento de texto: quer seja antes deste ser realizado OCR, através de processamento de imagem; durante a deteção de texto, na configuração proposta ao motor OCR; ou após com o tratamento dos resultados e do texto.

Estas diferentes interações são, na generalidade, aplicações sob um input inicial, de forma a transformá-lo e para resolver particularidades deste. Exemplos disto são: aplicação de técnicas de binarização que reduzem ruído ou imperfeições numa imagem; melhoria de resolução de uma imagem; correções ortográficas de texto; etc.

Estas interações podem então ser consideradas como um conjunto de ferramentas que podem ser aplicadas ao input do nosso problema de forma a abordar diferentes defeitos deste de acordo com as suas características; até porque, como também foi descrito para algumas destas, a sua utilização indiscriminada pode deteriorar, ao invés de melhorar, o produto final. As ferramentas comumente utilizadas não são explicitamente dedicadas ao problema de OCR em concreto, aumentando a sua utilidade e adaptabilidade para outras situações. Tal reforça o intuito base do trabalho, da criação de um toolkit que proporcione um conjunto de funcionalidades independentes, com a intenção final de procurar melhorar os resultados da transcrição de texto.

Dentro destas ferramentas, é notável a escassez de oferta dirigida para a análise e manipulação dos

resultados OCR, mais especificamente, os dados entre a realização de OCR e output do texto reconhecido.

Podemos então declarar aqui as primeira proposta:

**Proposta 1: A solução para o problema deve ser composta por um conjunto de ferramentas independentes, i.e. um toolkit.**

**Proposta 2: O Toolkit deverá abordar os desafios do contexto de OCR, nomeadamente: processamento de imagem; processamento de texto; e especial ênfase no processamento de resultados OCR.**

Durante a análise do estado da arte, notou-se que para soluções que procuram melhorar a aplicação de OCR, um processo com 3 partes principais pode ser identificado, sendo estas: pré-processamento de OCR, composto por processamento do input para realização de OCR, usualmente uma imagem; OCR, configurando o motor de OCR de acordo com as características do input, como a linguagem esperada do texto ou o dpi da imagem; pós-processamento de OCR, aplicando transformações nos resultados como, remover blocos de ruído, ordenar blocos identificados ou aplicar correções no texto.

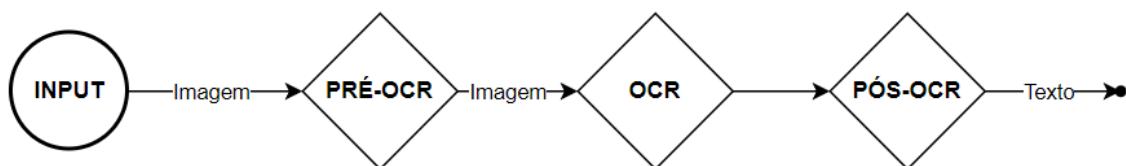


Figura 1: Aplicação de OCR com passos para melhoria de resultados

Observando a figura 1, podemos identificar que a estrutura das soluções para aplicação de OCR é uma pipeline.

Estas pipelines podem ser elaboradas como uma sequência de aplicações das ferramentas acima mencionadas, como por exemplo:

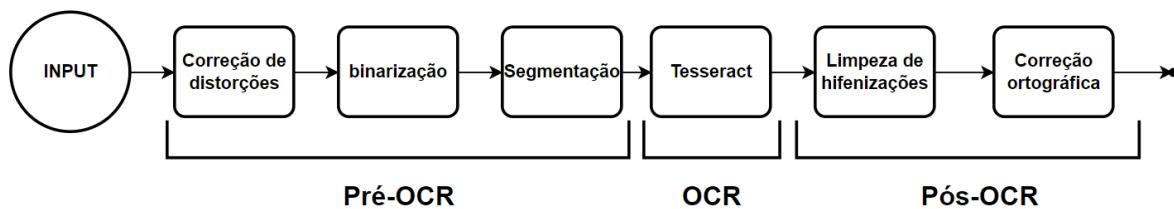


Figura 2: Exemplo de pipeline de aplicação de OCR

Assumindo então a proposta da criação de um toolkit, e a necessidade de verificar a eficácia deste e criar casos de uso deste, torna-se claro que será útil o desenvolvimento de uma pipeline de aplicação de OCR, que faça uso do toolkit desenvolvido, assim como de outras ferramentas disponíveis e que abordem questões em falta no toolkit.

Seguindo o estudo realizado, esta pipeline será composta por 3 partes principais: pré-OCR, OCR e pós-OCR.

**Proposta 3: A solução para o problema deve possuir uma pipeline de aplicação de OCR que faça uso da toolkit desenvolvida.**

**Proposta 4: A pipeline desenvolvida deve ser composta por 3 partes principais: pré-processamento de OCR, OCR e pós-processamento de OCR.**

Como discutido, o uso de ferramentas independentes é interessante visto que diferentes inputs irão necessitar de tratamentos distintos de forma a obter os melhores resultados. Desta forma, é interessante que a pipeline desenvolvida não seja totalmente sequencial, de modo a aumentar a sua utilidade.

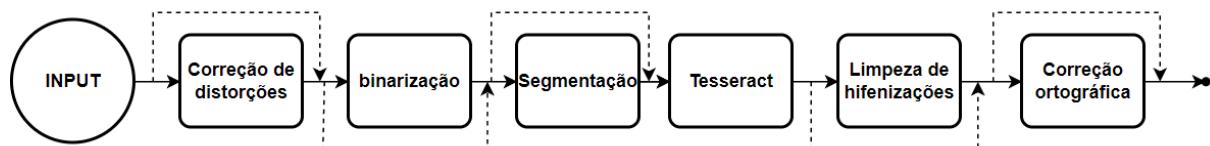


Figura 3: Exemplo de pipeline de aplicação de OCR com blocos opcionais

**Proposta 5: A pipeline deverá ser configurável. Permitirá, dentro dos blocos disponíveis, escolher aqueles que são aplicados.**

Esta pipeline, seguindo a estrutura explicada de aplicação de OCR, apresentará 3 partes principais onde, a primeira - pré-OCR - irá lidar com um input do tipo imagem e terá como output uma imagem; a segunda - OCR - terá como input uma imagem e devolverá os resultados de OCR; e a última - pós-OCR - irá ter como input os resultados de OCR, e terá como output a transcrição do texto da imagem.

Através deste último output, podem ser obtidas métricas objetivas sobre os resultados da pipeline quando comparados com uma que, indiretamente, serão também avaliações do Toolkit. A implementação de um módulo dedicado a esta avaliação demonstra a sua utilidade através da sua possibilidade de apreciação dos resultados e depuração das ferramentas.

Nota-se uma ambiguidade no output da 2º parte e no input da 3º, os resultados de OCR. Tal, deve-se ao facto de motores OCR, como o Tesseract, permitirem vários tipos de output. É então relevante a criação de uma estrutura de dados universal para a representação de resultados OCR.

Como estudado no capítulo 2, tais representações já possuem um standard um formato de ficheiro, como **HOCR**, portanto a estrutura de dados escolhida deve ser baseado nestes, ou ser convertível para o standard.

**Proposta 6: A pipeline deverá incluir um módulo de validação, de modo a permitir apreciar os resultados desta e apoiar a depuração do Toolkit.**

**Proposta 7: Criação de uma estrutura de dados universal para a representação dos resultados de OCR.**

**Proposta 8: A estrutura de dados universal para a representação dos resultados de OCR deve ser baseada ou convertível num formato standard.**

No estudo do estado da arte, foi possível compreender que a criação de algoritmos ubíquos para todos os tipos de documentos é um empreendimento com diminutas chances de sucesso, sendo que para muitos problemas, como por exemplo o cálculo da ordem de leitura, mesmo focando num só tipo de documento como jornais, os resultados podem não ser satisfatórios devido à diversidade de estruturas existentes. Consequentemente, é esperado que a pipeline não seja sempre suficiente na resolução do problema. Deste modo, servindo também como outro caso de uso do toolkit, a criação de uma ferramenta que permite ao utilizador um maior manuseamento dos resultados de OCR é uma proposta para a solução.

Tendo em conta a forte qualidade visual desta área de trabalho, onde se procura processar e analisar imagens de documentos, a utilidade e usabilidade desta ferramenta de edição é elevada ao definirmos que o editor seja gráfico.

Além do refinamento através de uma manipulação mais minuciosa, este editor facilitará a realização de testes e depuração das outras componentes da solução, por sua vez, suavizando o processo de aprimoramento delas e, recursivamente, o seu.

**Proposta 9: A solução do problema deverá possuir um editor gráfico dos resultados de OCR, que permita aplicar ferramentas propostas pelo toolkit.**

Em suma, as propostas definidas para o desenho da solução são:

- 1** A solução para o problema deve ser composta por um conjunto de ferramentas independentes, i.e. um toolkit.
- 2** O Toolkit deverá abordar os desafios do contexto de OCR, nomeadamente: processamento de imagem; processamento de texto; e especial ênfase no processamento de resultados OCR.
- 3** A solução para o problema deve possuir uma pipeline de aplicação de OCR que faça uso da toolkit desenvolvida.
- 4** A pipeline desenvolvida deve ser composta por 3 partes principais: pré-processamento de OCR, OCR e pós-processamento de OCR.
- 5** A pipeline deverá ser configurável. Permitirá, dentro dos blocos disponíveis, escolher aqueles que são aplicados.
- 6** A pipeline deverá incluir um módulo de validação, de modo a permitir apreciar os resultados desta e apoiar a depuração do Toolkit.
- 7** Criação de uma estrutura de dados universal para a representação dos resultados de OCR.
- 8** A estrutura de dados universal para a representação dos resultados de OCR deve ser baseada ou convertível num formato standard.
- 9** A solução do problema deverá possuir um editor gráfico dos resultados de OCR, que permita aplicar ferramentas propostas pelo toolkit.

## 3.2 Modelos

Realizada uma acessão mais concreta do problema e das soluções que para este são sugeridas, segue-se uma discussão sobre os elementos fundamentais que servirão como base do trabalho a realizar. No caso deste problema, os de maior relevância são os tipos de dados que serão processados: imagens para processamento e realização de OCR; e estruturas de dados para representação dos resultados OCR.

### 3.2.1 Imagem

A área de computação de visão é extensa e complexa, apresentando já a sua medida de standards e ferramentas disponíveis. Neste sentido, no propósito deste problema, apenas sobra escolher o tipo de estruturas de dados que se pretende usar de forma a gerar uma consistência no projeto.

Sendo que a componente prática do projeto será maioritariamente produzida utilizando Python, optou-se pela utilização das estruturas de dados oferecidas pelo **open-cv** devido à comprehensiva biblioteca de métodos disponíveis, utilidade para o problema e reputação na área.

### 3.2.2 OCR Tree - estrutura de dados para resultados OCR

Os resultados de OCR podem tomar múltiplas formas; tomando como exemplo a utilização de Tesseract, este permite a geração de output diretamente no formato de texto, dicionário, pdf, hocr, etc.. Deste modo, é necessário decidir o tipo de formato que se pretende para ser trabalhado pelas ferramentas da solução.

Tendo em conta os tipos de problemas que se pretendem resolver - remover ruído detetado como texto, calcular ordem de leitura, categorizar elementos detetados -, é necessária informação além do texto detetado, sendo útil informações geométricas sobre este ex.: posição na imagem, dimensões de elementos detetados. Deste modo, é útil observar o standard HOCR.



Figura 4: Representação visual dos resultados de OCR armazenados em formato HOCR

Como podemos observar na figura 4, é possível criar uma representação visual dos resultados de OCR a partir de ficheiros HOCR, mais digerível do que o seu formato base. Este tipo de representação disponibiliza uma importante ferramenta de debugging de métodos que transformem os resultados OCR, assim como uma melhor compreensão sobre a natureza dos resultados OCR como uma estrutura de dados do tipo árvore. Esta representação visual é apenas possível devido à informação extra armazenada sobre o texto reconhecido, como as bounding boxes.

A partir deste tipo de ficheiro, temos então que a informação mais relevante que podemos obter é:

- posição e dimensões do texto detetado, através do armazenamento de bounding boxes.
- diferentes níveis de deteção de texto, permitindo uma estruturação base deste.

- pagina
  - bloco
  - parágrafo
  - linha
  - palavra
- identificadores dos elementos.
  - nível de confiança do texto detetado.

No propósito da criação de uma classe de dados para o papel da estrutura de dados universal para representar os resultados OCR, seguiu-se com um desenho baseado numa árvore, denominada **OCR Tree**, e que irá possuir, como atributos base, os acima listados.

Esta classe irá, para cumprir standards, possuir um conversor de e/para HOCR.

Maior detalhe sobre esta está disponível nos capítulos seguintes, dedicados à implementação da solução.

### 3.3 Arquitetura da solução

Descritas as decisões fundamentais sobre a solução e os modelos base que a irão compor, expõe-se nesta secção a arquitetura desta, na sua generalidade e das suas partes.

A arquitetura geral da solução é composta por 3 componentes principais, correspondentes às propostas anteriores: o "OSDOCR Toolkit", que será um conjunto de ferramentas desenvolvidas para melhorar os resultados de OCR; a "OSDOCR Pipeline" que será uma aplicação do toolkit, assim como de algumas soluções já existentes, no formato de uma pipeline versátil; "OSDOCR Editor" que será uma segunda aplicação do toolkit, com o intuito de permitir um manuseamento mais delicado dos resultados OCR.

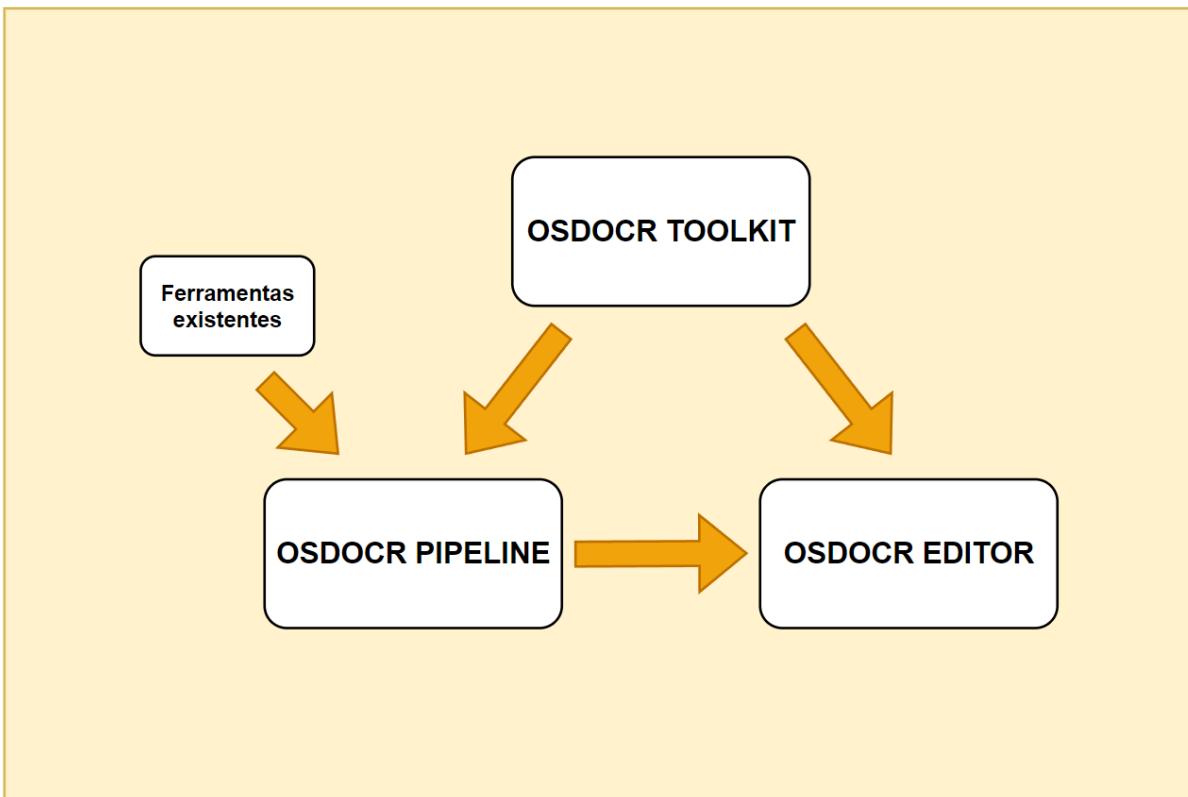


Figura 5: Arquitetura geral da solução

A maioria do código desenvolvido foi utilizando **Python**, com algumas instâncias de **C**.

### 3.3.1 OSDOCR Toolkit

A componente de OSDOCR Toolkit servirá como base para as outras duas componentes da solução e, como o nome desta indica, servirá como o produto principal a ser importado para projetos futuros.

Esta é composta por 3 módulos fundamentais, cada um dedicado a uma dada área: "Imagen", disponibilizando ferramentas de processamento e análise de imagens de documentos; "OCR Tree", com ferramentas que permitem manipulação de resultados OCR representados pela classe OCR Tree; "Texto", para o processamento de texto e geração de output textual.



Figura 6: Arquitetura OSDOCR Toolkit

### 3.3.2 OSDOCR Pipeline

A componente de OSDOCR Pipeline é um exemplo de uma aplicação do toolkit criado no caso de uso clássico para a aplicação de OCR. Este apresenta portanto os 3 procedimentos principais deste uso clássico: pré-processamento OCR, tratamento e análise da imagem de input; OCR; pós-processamento OCR, tratamento e interpretação dos resultados.

Como proposto, esta componente torna-se mais útil se tiver um maior nível de versatilidade. Deste modo, mesmo os procedimentos gerais de pré e pós processamento são opcionais, como se pode ver na figura 7.

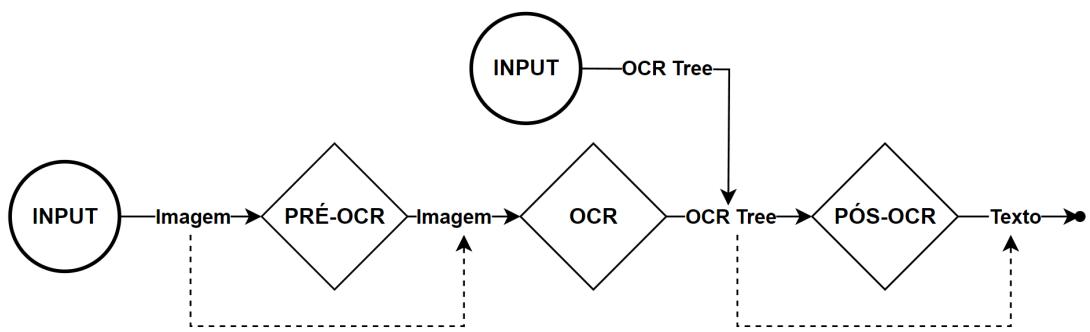


Figura 7: Arquitetura OSDOCR Pipeline - high level

Como também se pode verificar na figura, os tipos de dados manipulados na pipeline são contidos -

havendo apenas 3 tipos usados -, auxiliando a possibilidade de a configurar.

Cada um dos procedimentos é composto por blocos elementares, que, mantendo a mesma lógica de versatilidade, são opcionais.

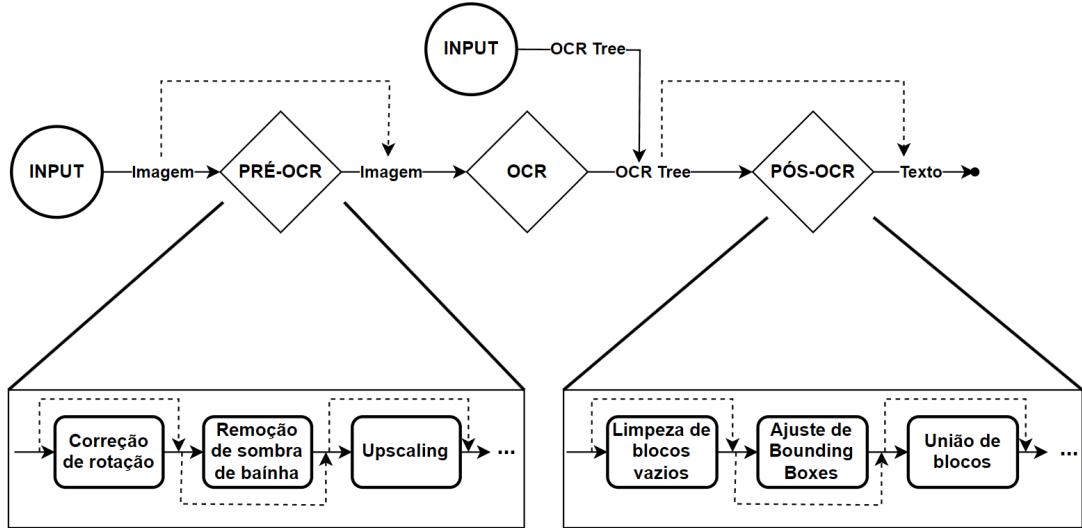


Figura 8: Arquitetura OSDOCR Pipeline - reduzida

Alguns destes blocos, principalmente no que toca a manipulação de imagem, fazem uso de soluções externas disponíveis, ex.: upscaling de imagem utilizando modelos de Deep Learning open-source.

Maior detalhe sobre estes blocos será discutido no capítulo de implementação da pipeline.

### 3.3.3 OSDOCR Editor

O OSDOCR Editor é um GUI de arquitetura relativamente simples, sendo que apresenta uma proposta bastante focada, a de manipulação de OCR Tree.

Este segue uma arquitetura **MVC** (Model View Controller), tendo sido desenvolvido a **View** geral utilizando a biblioteca **PySimpleGui**. Para complemento desta, e devido à permitida compatibilidade, foi utilizado **Matplotlib** no desenvolvimento do canvas visualizador e de manipulação da OCR Tree.

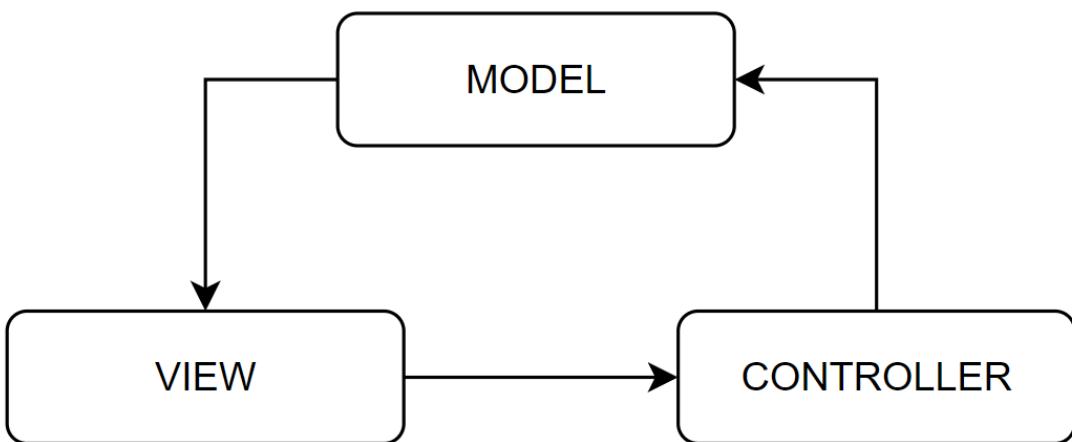


Figura 9: Arquitetura MVC

O **controlador** neste GUI provêm - para além do Matplotlib e do PySimpleGui - das ferramentas disponibilizadas pelo OSDOCR Toolkit e, nalguns casos como por exemplo de aplicação de OCR numa área focada de uma imagem, da OSDOCR Pipeline.

Os constituintes principais do **modelo** são: imagem de input; OCR Tree; abstrações de OCR Tree, ex.: artigos (lista de listas OCR Tree).

## **Capítulo 4**

# **OSDOCR Estruturas de Dados - Implementação**

Neste capítulo, será descrito em detalhe a implementação dos modelos principais de dados usados na base do projeto. Estes são a estrutura OCR Tree, utilizada para representação de resultados OCR; e a estrutura Box, utilizada maioritariamente para a representação de Bounding Boxes, mas com a particularidade de possuir uma coleção de métodos que permitem a sua manipulação.

### **4.1 OCR Tree**

Como o produto final do projeto intende aceitar diferentes tipos de resultados OCR, i.e. resultantes de diferentes motores OCR ou de ficheiros como hOCR que já possuem os resultados, existe uma necessidade de converter estes diferentes formatos num único tipo que mantenha a informação base pretendida.

Estruturas de dados standard como [\[HOCR\]](#) ou [\[ALTO\]](#) apresentam um resultado final semelhante e com capacidade base de armazenamento de meta-dados superior porém, sendo baseados em XML, tornam a sua manipulação mais complexa e, em múltiplos casos, a informação proporcionada é excessiva ou gera conclusões erradas quando gerado de output automático (ex.: atribuição de classes caption a blocos que são títulos). Assim sendo, embora tenha sido desenvolvido um conversor de e para HOCR, para o atual projeto, optou-se pela criação de uma estrutura de dados própria.

Deste modo, tomando como inspiração os atributos dos resultados do Tesseract no modo de dicionário [\[Tesseract\]](#), foi implementada uma estrutura de dados no formato de árvore.

A escolha de uma estrutura de árvore permite a hierarquização de blocos de acordo com o seu nível, quer exista uma divisão de nível à partida, como é o caso do Tesseract que segue: página → bloco → parágrafo → linha → palavra; ou apenas um único nível, semelhante ao Keras-OCR.

Todos os algoritmos desenvolvidos, inclusive os métodos para visualização (métodos de debugging e GUI desenvolvido), assumem e trabalham com os dados de OCR no formato desta estrutura de dados.

As características mais relevantes desta estrutura são:

- **Level** : Nível/altura do nodo.
  - documento : 0
  - página : 1
  - bloco : 2
  - parágrafo : 3
  - linha : 4
  - palavra : 5
- **(page|block|par|line|word)\_num**: Identificação da ordem (dentro de outras caixas(ex.: linha), se aplicável)
- **text** : Texto do bloco, normalmente apenas preenchido ao nível da palavra
- **conf** : Confiança no texto
- **id**
- **type** : Tipo do bloco, ex.: delimitador, título
- **children**
- **box**: Bounding box do nodo, representado pela estrutura Box, que também possui métodos para transformações e verificações geométricas ou de características.
- Características de texto: ex.: texto iniciado (start\_text); texto não terminado (end\_text).

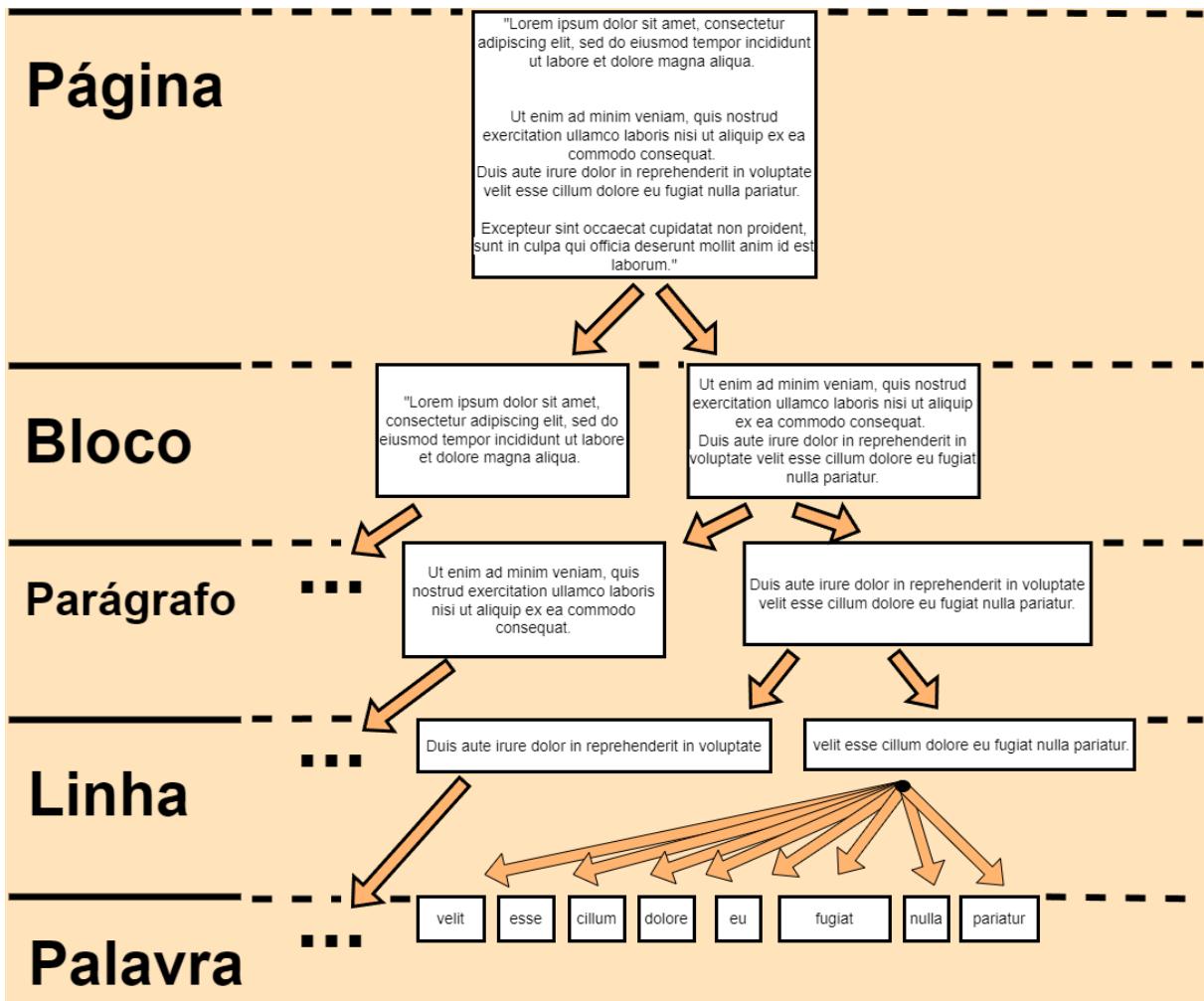


Figura 10: Árvore de resultados OCR num documento

Construtores da classe são capazes de admitir outros atributos não base de modo a expandir a utilidade da estrutura. Construtores disponíveis: iniciação por argumentos, dicionário, ficheiro JSON e ficheiro HOCR.

Da mesma forma, conversores para estes ficheiros compreendidos para iniciação também foram desenvolvidos.

A classe possuí por métodos de transformação e análise sobre a árvore OCR que facilitam a manipulação dos resultados OCR.

Segue-se uma lista dos métodos mais relevantes disponíveis da classe.

## **id\_boxes**

**Descrição:** Adiciona identificador aos blocos.

**Argumentos:**

- level : lista de níveis onde adicionar identificador
- ids (opt): dicionário de ids a utilizar caso não se queira iniciar no 0.
- delimiters (opt): flag para identificar delimitadores
- area (opt): argumento do tipo Box, que restringe os nodos a identificar a uma dada área
- override (opt): flag para reescrever id se já existe.

## **calculate\_mean\_height**

**Descrição:** Calcula a altura média das caixas de um dado nível.

**Argumentos:**

- level : nível a calcular
- conf (opt): valor de confiança de texto no caso de apenas serem relevantes caixas com certa confiança (aplicável apenas para nível de texto)

## **is\_text\_size**

**Descrição:** Verifica se um nodo se encontra dentro do tamanho de texto.

**Argumentos:**

- text\_size : tamanho de texto a comparar
- mean\_height (opt): altura do bloco, caso já tenha sido calculado
- range : margem de erro aceitável (relativo)
- level : nível das caixas usado caso seja necessário calcular a altura média
- conf : confiança do texto a utilizar para calcular a altura média

## **calculate\_character\_mean\_width**

**Descrição:** Calcula a largura média de letra.

**Argumentos:**

- conf (opt): valor de confiança de texto

## **is\_empty**

**Descrição:** Verifica se um nodo é vazio.

**Argumentos:**

- conf : confiança de texto a utilizar para considerar palavras válidas
- only\_text : flag que dita se o tipo do bloco influencia o resultado, i.e. blocos de tipo "image" não são vazios

## **text\_is\_title**

**Descrição:** Verifica se um nodo é potencial título.

**Algoritmo:** Caixa não é texto vertical e é maior do que o tamanho normal de texto.

**Argumentos:**

- normal\_text\_size : tamanho de texto considerado como normal
- conf : confiança de texto a utilizar para considerar palavras válidas
- range : margem de acerto aceitável (relativo)
- level : nível usado para calcular o tamanho médio do bloco

## **is\_delimiter**

**Descrição:** Verifica se um nodo é potencial delimitador.

**Algoritmo:** Caixa já é do tipo delimitador, ou é vazia e segue a regra:

$box.width \geq box.height * 4 \mid box.height \geq box.width * 4$ .

**Argumentos:**

- conf : confiança de texto a utilizar para considerar palavras válidas
- only\_type : flag que dita se usa apenas o tipo do nodo para a verificação

## **is\_image**

**Descrição:** Verifica se um nodo é potencial imagem.

**Algoritmo:** Caixa já é do tipo imagem ou, é vazia, não é um delimitador e é 3 vezes mais alta do que o tamanho de texto.

**Argumentos:**

- conf : confiança de texto a utilizar para considerar palavras válidas
- text\_size : tamanho de texto a utilizar para comparação com altura da caixa
- only\_type : flag que dita se usa apenas o tipo do nodo para a verificação

## **get\_boxes\_in\_area**

**Descrição:** Obtém todas as caixas numa dada área.

**Argumentos:**

- area : área de interesse
- level : nível dos nodos a ir buscas. Se nível == -1, obtém todos os nodos
- conf : confiança de texto a utilizar para considerar nodos válidos
- ignore\_type : tipos de nodo a ignorar

## **is\_vertical\_text**

**Descrição:** Verifica se um nodo é texto vertical.

**Argumentos:**

- conf : confiança de texto a utilizar para considerar palavras válidas

**Algoritmo:**

---

### **Algorithm 1** Verificação de texto vertical

---

```
1: if nodo não é vazio then
2:   lines
3:   if len(lines) == 0 then return False
4:   end if
5:   // Linha única
6:   if len(lines) == 1 then
7:     words
8:     // Palavra única
9:     if len(words) == 1 then
10:       if altura da palavra >= 2 * largura da palavra then return True
11:     end if
12:     // Múltiplas palavras
13:   else
14:     // Verifica se a maioria das palavras coincidem horizontalmente
15:     widest_word <- calcula palavra mais larga
16:     overlapped_words = 0
17:     for word in words do
18:       if word == widest_word then
19:         continue
20:       end if
```

```

21:         if word.box.within_horizontal_boxes(widest_word.box,range=0.1) then
22:             overlapped_words += 1
23:         end if
24:     end for
25:     if overlapped_words/len(words) >= 0.5 then return True
26:     end if
27: end if
28: // Múltiplas linhas
29: else
30:     // Verifica se a maioria das linhas coincidem verticalmente
31:     tallest_line <- calcula linha mais alta
32:     overlapped_lines = 0
33:     for line in lines do
34:         if line == tallest_line then
35:             continue
36:         end if
37:         if line.box.withinvertical_boxes(tallest_line.box,range=0.1) then
38:             overlapped_lines += 1
39:         end if
40:     end for
41:     if overlapped_lines/len(lines) >= 0.5 then return True
42:     end if
43: end if
44: return False

```

---

## **prune\_children\_area**

**Descrição:** Atualiza dimensões dos filhos de um nodo para se encaixarem dentro de uma área.

**Argumentos:**

- area : área de interesse

## **boxes\_below** (método semelhante para as outras direções)

**Descrição:** Dada uma lista de OCR Tree, devolve aqueles que se encontram por baixo do bloco atual.

Os blocos filtrados podem intersestar ou estar dentro do bloco comparado.

**Argumentos:**

- blocks : lista de blocos a filtrar

## **boxes\_directly\_below** (método semelhante para as outras direções)

**Descrição:** Dada uma lista de OCR Tree, devolve aqueles que se encontram diretamente por baixo (nos limites das suas coordenadas de x) do bloco atual. Blocos filtrados não estão dentro do bloco comparado e não podem estar diretamente por baixo dos outros blocos.

### **Argumentos:**

- blocks : lista de blocos a filtrar

## **join\_trees**

**Descrição:** Junta duas OCR Tree com o mesmo nível numa única árvore. Tem dois métodos principais de junção das árvores: vertical, operação mais simples em que apenas se juntam as duas listas de ramos dos nodos raiz (assume-se que uma das árvores é mais alta do que a outra e não se intersetam); e horizontal, onde se procura juntar árvores que têm interseção no eixo y, sendo necessário verificar as posições que os filhos devem tomar e se certos filhos devem ser unidos num único (podendo resultar numa junção de linhas resultando em texto intercalado).

Recebendo o argumento 'auto' como orientação, calcula a orientação a usar. Se o bloco estiver dentro do que quer unir ou o interseca no eixo horizontal, orientação 'horizontal'; senão é 'vertical'.

Antes de realizar a junção faz uma verificação das posições dos blocos, sendo que para orientação 'horizontal' precisa que o bloco principal seja o mais à esquerda e, para 'vertical' tem de ser o mais acima. Troca os blocos, se necessário, de acordo com esta verificação.

### **Argumentos:**

- tree : árvore a juntar
- orientation : orientação da junção - vertical, horizontal ou auto.

### **Algoritmo:**

---

#### **Algorithm 2** Junção horizontal

---

- 1: tree\_children
- 2: self\_children
- 3: // no último nível, filhos são ordenados da esquerda para a direita
- 4: **if** último nível da tree **then**
- 5:     tree\_children ← ordenar lista da esquerda para a direita
- 6: **end if**
- 7: **for** child in tree\_children **do**
- 8:     **if** não é o último nível **then**

```

9:    self_children ← ordena de cima para baixo
10:   if child pode ser inserida no topo ou fundo da lista then
11:      self_children ← insere child no início ou fim
12:   else
13:      // procura slot para inserir, ou nodo com quem unir
14:      joined = False
15:      for i in range(len(self_children)) do
16:          if posição válida e child não intersetava com nodo i ou nodo i+1 then
17:              self.children ← adiciona child entre os dois nodos
18:              joined = True
19:          else if intersetava com nodo i then
20:              if interseção em pelo menos 70% da altura da caixa then
21:                  if nodo i tem filhos then
22:                      // join recursivo
23:                      self_children[i].join_trees(child,orientation=orientation)
24:                  else
25:                      self_children ← insere child depois do nodo i
26:                  end if
27:                  joined = True
28:              else
29:                  // procura local mais baixo para inserir (por poder intersetar com varios blocos)
30:                  for j in range(i,len(self_children)) do
31:                      if nodo j mais alto do que child then
32:                          self_children ← insere child depois do nodo j
33:                          joined = True
34:                      end if
35:                  end for
36:                  if not joined then
37:                      self_children ← procura local mais alto válido para inserir
38:                      joined = True
39:                  end if
40:              end if
41:          end if
42:          if joined then
43:              break
44:          end if
45:      end for
46:      end if
47:  else

```

```

48:     self.children ← adiciona child na primeira posição cujo right seja maior do que o seu
49:   end if
50:   self._children ← atualiza lista de filhos
51: end for

```

---

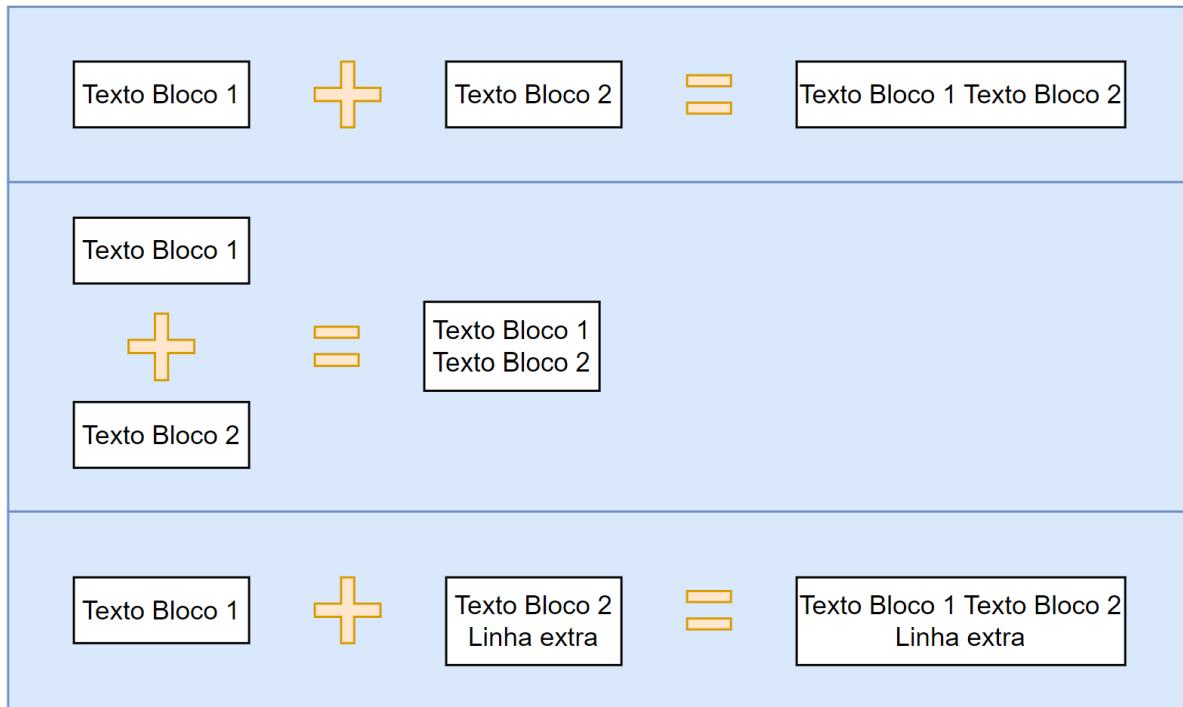


Figura 11: Exemplos de junção de OCR Tree

## **remove\_blocks\_inside**

**Descrição:** Remove os blocos dentro do bloco com dado id. Blocos removidos são do mesmo nível que o bloco com dado id.

**Argumentos:**

- id : id do bloco a limpar
- block\_level : nível do bloco a limpar

## **update\_position**

**Descrição:** Atualiza a posição da bounding box de um nodo e dos seus filhos. Especialmente útil para o editor OCR.

**Argumentos:**

- top : valor a atualizar verticalmente
- left : valor a atualizar horizontalmente
- absolute : flag que indica se operação é de tipo absoluta, i.e. bounding box vai ser diretamente atualizada com estes valores, ou relativa, aos valores da bounding box serão somados os argumentos

## **update\_size**

**Descrição:** Atualiza o tamanho da bounding box de um nodo e dos seus filhos nas arestas (filhos interiores não serão alterados). Especialmente útil para o editor OCR.

### **Argumentos:**

- top : valor a atualizar ao topo
- left : valor a atualizar à esquerda
- bottom : valor a atualizar ao fundo
- right : valor a atualizar à direita
- absolute : flag que indica se operação é de tipo absoluta, i.e. bounding box vai ser diretamente atualizada com estes valores, ou relativa, aos valores da bounding box serão somados os argumentos

## **update\_box**

**Descrição:** Atualiza diretamente valor da bounding box do nodo e dos filhos. Especialmente útil para o editor OCR.

### **Argumentos:**

- top : valor a atualizar ao topo
- left : valor a atualizar à esquerda
- bottom : valor a atualizar ao fundo
- right : valor a atualizar à direita
- children : flag que indica se é para se aplicar ajuste direto no nodo, ou apenas ajustar de forma a não sair da bounding box do pai.

## **scale\_dimensions**

**Descrição:** Escala dimensões da bounding box do nodo e dos seus filhos. Especialmente útil para o editor OCR.

**Argumentos:**

- `scale_width` : escalar de valores do eixo horizontal
- `scale_height` : escalar de valores do eixo vertical

## 4.2 Box

A estrutura de dados Box é utilizada maioritariamente para encapsular os dados das bounding boxes dos resultados de OCR. Embora em geral este tipo de dados seja geralmente fornecido por métodos de módulos de manipulação de imagens na forma de tuplo, a utilização de uma classe dedicada permite o desenvolvimento e utilização de métodos para sua manipulação de forma mais simples e organizada.

Tal como a estrutura de dados OCR Tree, esta classe apresenta construtores e conversores de ficheiros diferentes tipos: argumentos simples, dicionário, ficheiro JSON.

Os principais atributos da estrutura são:

- **left** : Valor mais à esquerda da caixa.
- **right** : Valor mais à direita da caixa.
- **top** : Valor mais em cima da caixa (menor do que bottom por ser baseado em manipulação de imagem).
- **bottom** : Valor mais em baixo da caixa.
- **width** : Comprimento da caixa.
- **height** : Altura da caixa.

Realça-se que os atributos desta classe são esperados no formato de inteiros, devido a ter como foco o seu uso no contexto do espaço de imagens.

Segue-se uma lista dos métodos mais relevantes disponíveis da classe.

### update

**Descrição:** Atualiza os valores dos atributos de posição da caixa. Atributos de posição são mantidos válidos, i.e. `left <= right` e `top <= bottom`. Altura e comprimento são atualizados automaticamente.

**Argumentos:**

- `top` : valor a atualizar ao topo

- left : valor a atualizar à esquerda
- bottom : valor a atualizar ao fundo
- right : valor a atualizar à direita

## **move**

**Descrição:** Soma valores aos atributos de posição da caixa.

**Argumentos:**

- x : valor a somar nos atributos de posição horizontais
- y : valor a somar nos atributos de posição verticais

## **within\_vertical\_boxes** (método semelhante para direção horizontal)

**Descrição:** Verifica se caixa e caixa a ser comparada estão alinhadas verticalmente, podendo considerar uma margem de acerto. Verificação é realizada nos dois sentidos, i.e. caixa 1 alinhada com caixa 2 ou vice-versa.

**Argumentos:**

- box : caixa a comparar
- range : valor relativo da altura da caixa, a servir como margem para considerar na verificação

## **is\_inside\_box**

**Descrição:** Verifica se caixa a ser comparada está dentro da caixa. Caixa a ser compara tem de estar completamente dentro para resultado afirmativo.

**Argumentos:**

- box : caixa a comparar

## **intersects\_box**

**Descrição:** Verifica se caixa a ser comparada intersetam com a caixa.

**Argumentos:**

- box : caixa a comparar
- extend\_vertical : flag para indicar se verificação deve ser feita longo do eixo x (ex.: utilizado para verificar se caixa comparada está diretamente por acima da caixa)

- extend\_horizontal : flag para indicar se verificação deve ser feita apenas longo do eixo y (ex.: utilizado para verificar se caixa comparada está diretamente à direita caixa)
- inside : flag para indicar se verificação de caixa dentro conta como interseção

## **intersect\_area\_box**

**Descrição:** Calcula a caixa de interseção entre a caixa e uma caixa a comparar.

**Argumentos:**

- box : caixa a comparar

## **remove\_box\_area**

**Descrição:** Remove área da caixa. Procura remover área aplicando as menores modificações possíveis. Apenas realiza modificações, se área fornecida está dentro da caixa.

**Argumentos:**

- area : área da caixa a remover

## **get\_box\_orientation**

**Descrição:** Método naive para obter orientação da caixa (horizontal, vertical ou square) de acordo com a diferença entre a sua altura e comprimento.

## **join**

**Descrição:** Une duas caixas.

**Argumentos:**

- box : caixa a unir

## **distance\_to**

**Descrição:** Calcula distância entre duas caixas. Procura dois pontos mais próximos de acordo com argumentos dados e utiliza distância euclidiana para calcular a distância.

**Argumentos:**

- box : caixa a comparar

- border (opt): borda da caixa a ter em conta. Valores disponíveis: "left", "right", "top", "bottom", "closest". Se "closest" for fornecido, procura a menor distância entre bordas. Se nenhum valor for fornecido, utilizado os pontos centrais das caixas.

## **distance\_to\_point**

**Descrição:** Calcula distância entre a caixa e um ponto. Procura calcular a menor distância da caixa ao ponto (tendo em conta a diferença entre o ponto e as bordas).

**Argumentos:**

- x : valor x do ponto
- y : valor y do ponto

## **vertices**

**Descrição:** Retorna uma lista dos vértices da caixa na forma de tuplos (x,y) seguindo de cima para baixo, esquerda para a direita.

## **closest\_edge\_point**

**Descrição:** Calcula a borda mais próxima entre a caixa e um ponto. Utilizado no editor de resultados OCR para operações de divisão de blocos.

**Argumentos:**

- x : valor x do ponto
- y : valor y do ponto

## **Capítulo 5**

# **OSDOCR Toolkit - Implementação**

Neste capítulo será discutida a componente de Toolkit, premissa base do tema da dissertação. Esta componente consiste num conjunto de ferramentas focado na melhoria dos resultados obtidos da aplicação de **OCR** em documentos antigos, com especial interesse em jornais.

Estas ferramentas são então pertinentes para os diversos passos do processo convencional de **OCR**, i.e. pré-processamento, OCR e pós-processamento; atendendo tanto a processamento de imagem, processamento de resultados de OCR e texto, e validação de resultados.

### **5.1 Sumário**

- Processamento de resultados OCR 5.2
  - Conversão de resultados OCR 5.2.1
  - Debugging 5.2.1
  - Análise de texto 5.2.2
  - Limpeza de OCR Tree 5.2.3
  - Categorização de Blocos 5.2.4
  - Divisão de blocos 5.2.5
  - Cálculo de ordem de leitura 5.2.6
  - Segmentação de resultados 5.2.7
- Processamento de imagem 5.3
  - Correção de ângulos de rotação
  - Corte de sombra nas margens
  - Binarização de imagem
  - Identificação de delimitadores
  - Identificação de imagens no documento

- Segmentação de documento
- Processamento de texto 5.4
  - Limpeza de hifenização
  - Geração de output

## 5.2 Processamento de resultados OCR

O resultado da aplicação de OCR em imagens de baixa qualidade ou complexas como é o caso de jornais, é na generalidade propício a um output ruidoso e com vários defeitos em diferentes níveis. Alguns destes são: imprecisões nas bounding boxes dos blocos; texto reconhecido com erros; ruído identificado como texto; blocos que deveriam ser separados.

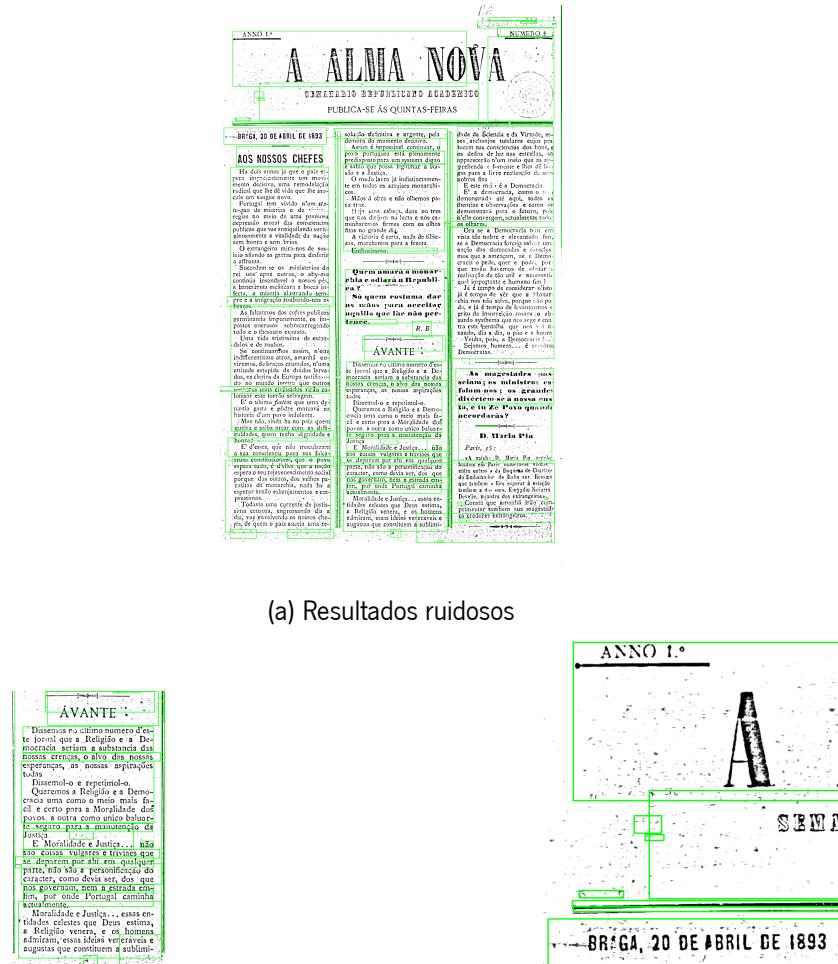


Figura 12: Exemplo de jornal com resultados OCR ruidosos e com conflitos.

Deste modo, surgiu a oportunidade de adicionar ao *toolkit* funcionalidades que fossem capazes de abordar estes problemas.

Esta secção trata particularmente de funcionalidades aplicadas sobre os resultados de OCR, i.e. após a aplicação de reconhecimento de caracteres, não tendo influencia no input desse procedimento.

### 5.2.1 Conversão de resultados OCR

Como base para a manipulação dos dados, como mencionado anteriormente, foi utilizada a estrutura de dados OCR Tree. Consequentemente, um conjunto de conversores desta classe foram desenvolvidos, nomeadamente:

- JSON : de e para JSON. Inspirado no output de Tesseract para dicionário.
- HOCR : de e para HOCR. Standard para armazenamento de dados resultantes de OCR.
- Texto : para texto.
- MD : para markdown.

## Debugging

Uma das características fundamentais de OCR é a sua capacidade para visualização fácil de resultados. Uma das mais úteis ferramentas de debugging geradas foi a reconstrução da OCR Tree sobre a imagem original.

### **draw\_bounding\_boxes**

**Descrição:** Desenha blocos de OCR Tree sob uma imagem.

**Argumentos:**

- ocr\_results : OCR Tree com os blocos a desenhar
- img : imagem a ser modificada
- draw\_levels : lista dos níveis de nodos a desenhar
- conf : confiança mínima de texto dos blocos de nível de texto a desenhar
- id : flag para desenhar id do bloco, caso disponível

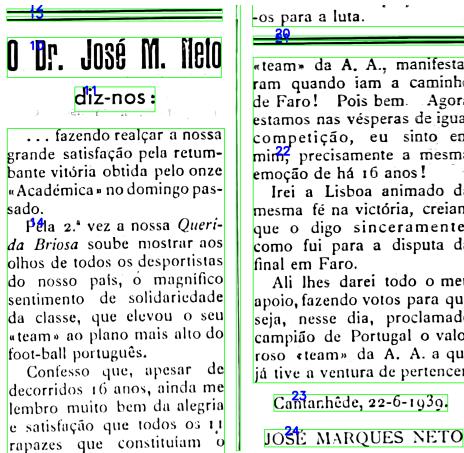


Figura 13: Exemplo de desenho de OCR Tree por cima do documento.

Outros métodos criados, mais circunstanciais, permitem o desenho do template de um jornal e de artigos respetivamente.

### **draw\_journal\_template**

**Descrição:** Desenha o template de um jornal sob uma imagem.

#### **Argumentos:**

- `journal_data` : dicionário com entradas dos segmentos de um jornal (header, columns, footer).
- Cada um com um objeto do tipo Box que dita a bounding box do elemento. No caso das colunas é uma lista de Box.
- `img` : imagem a ser modificada



Figura 14: Exemplo de desenho de template de jornal.

## **draw\_articles**

**Descrição:** Desenha artigos sob uma imagem.

**Argumentos:**

- articles : lista de listas de OCR Tree. Assume-se que cada lista de OCR Tree é um artigo. Cada OCR Tree será um nodo de nível 2
- img : imagem a ser modificada



Figura 15: Exemplo de desenho de artigos de jornal. Cada um dos artigos é desenhado com uma cor única.

### **5.2.2 Análise de texto**

A análise de texto dos resultados OCR permite inferir características sobre o documento que não estão à partida disponíveis na OCR Tree. Foi desenvolvido um conjunto de métodos que cada um infere uma das seguintes métricas a partir da OCR Tree principal:

- Tamanho de texto normal : método **get\_text\_sizes**
- Espaçamento médio de palavras : método **analyze\_text**
- Largura média de letra : método **analyze\_text**

- Colunas do documento : método **get\_columns**

Todas estas métricas podem ser obtidas na chamada do método **analyze\_text**.

Naturalmente, a qualidade do cálculo destas métricas irá depender da qualidade dos resultados OCR, ou no caso de análise de texto, do nível de confiança de texto usado.

Os métodos **get\_text\_sizes** e **get\_columns** são ambos baseados em análise de frequências e procura de picos na curva destas.

### **get\_text\_sizes**

**Descrição:** Analisa as frequências dos tamanhos de linha, pesados pelo número de palavras que as respetivas linhas têm, de modo a obter os tamanhos de letra mais proeminentes por análise dos picos da curva de frequências.

A curva é obtida a partir de um smoothing da lista de frequências e os picos são calculados baseado em proeminência.

Devolve pelo menos um tamanho de letra : `normal_text_size`.

#### **Argumentos:**

- `ocr_results` : OCR Tree a analisar
- `method` : método de smoothing. Opções : WhittakerSmoothen (por defeito), savgol\_filter
- `conf` : confiança de texto mínima. Restringe as palavras utilizadas para o cálculo do tamanho das linhas

#### **Algoritmo:**

---

##### **Algorithm 3** Cálculo de tamanhos de texto

---

```

1: text_sizes = { 'normal_text_size' : 0 }

2: lines ← obtém linhas da OCR Tree

3: line_sizes = []

4: // Cálculo das frequências de tamanhos das linhas

5: for line in lines do

6:   if line não é vazia e não é texto vertical then
7:     lmh ← altura média da linha (arredondado a inteiro)
8:     linhe_sizes[lmh] ← soma 1 + peso (nº palavras de confiança na linha)

9:   end if

10: end for

```

```
11: // smoothing das linhas
12: // WhittakerSmoother : lambda = 1e1; order = 3; data_length= len(line_sizes)
13: // savgol_filter : window_length = round(len(line_sizes*0.1)); polyorder = 2
14: line_sizes_smooth ← smooth de line_sizes usando o método escolhido
15: // cálculo de picos utilizando a função find_peaks do módulo spicy com proeminência a 10% da
    frequência máxima
16: peaks_smooth ← cálculo dos picos
17: text_sizes['normal_text_size'] ← máximo das frequências, i.e. altura mais comum
18: // se houver mais picos, aqueles abaixo do maior serão tamanhos de letra pequena e da mesma
    forma para picos acima do maior
```

---

## get\_columns

**Descrição:** Analisa as frequências das margens das bounding boxes dos blocos de nível 2, pesados pelo número de palavras que os respetivos blocos têm, de modo a obter os pontos esquerdos e direito mais proeminentes por análise dos picos das curvas de frequências.

A curva é obtida a partir de um smoothing da lista de frequências e os picos são calculados baseado em proeminência.

Devolve uma lista do tipo Box com o espaço das colunas encontradas.

Este método é bastante dependente da qualidade dos blocos reconhecidos, sendo que em geral é recomendado usar o método com o mesmo objetivo mas abordagem de análise de imagem, discutido na secção de processamento de imagem.

### Argumentos:

- ocr\_results : OCR Tree a analisar
- method : método de smoothing. Opções : WhittakerSmoother (por defeito), savgol\_filter
- conf : confiança de texto mínima. Restringe as palavras utilizadas para o peso das frequências das margens

**Algoritmo:** Algoritmo semelhante ao método get\_text\_sizes mas com análise das margens esquerda e direita das bounding boxes de nodos de nível 2. São calculados os picos de margens esquerdas e direitas separadamente que são depois pareados de forma a formar o espaço das diferentes colunas.

### 5.2.3 Limpeza de OCR Tree

Como já descrito anteriormente, com a norma dos resultados da aplicação de OCR em documentos antigos, surge a oportunidade de criar um conjunto de métodos que ajudem numa "limpeza" geral dos resultados, de forma a, por exemplo: remover elementos de ruído; unir blocos com características semelhantes; separar texto com espaço vazio pronunciado; ajustar dimensões das bounding boxes, etc..

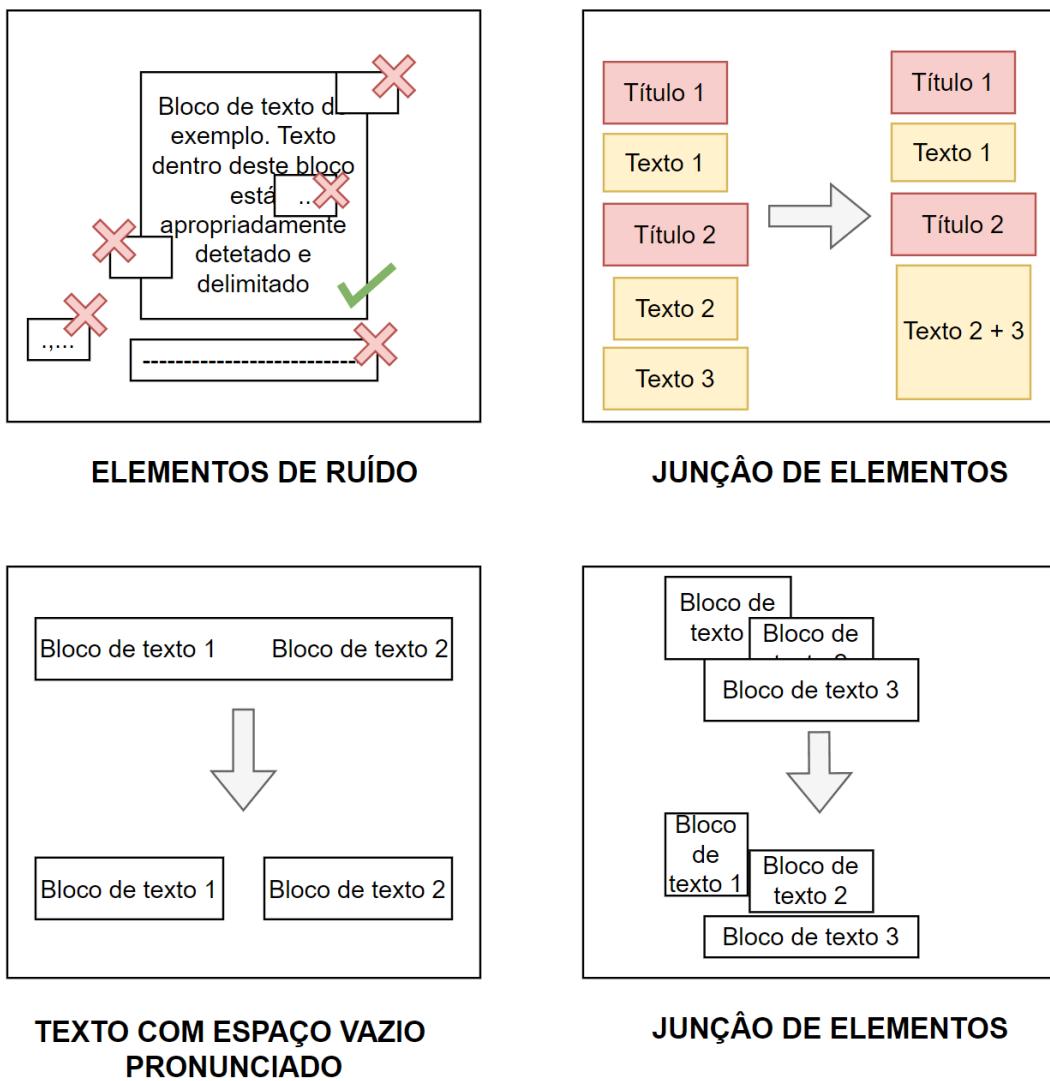


Figura 16: Exemplo de alguns dos casos listados que o toolkit irá abranger.

Descrevem-se então alguns dos métodos propostos para resolver alguns destes problemas.

#### **block\_bound\_box\_fix**

**Descrição:** Ajuste de bounding boxes de nível 2 para eliminar interseções. Procura: remover caixas vazias dentro de caixas de texto; eliminar interseções entre caixas de texto, ajustando de forma a alterar

a caixa menos impactada; unir blocos de texto dentro de outros.

### **Argumentos:**

- ocr\_results : OCR Tree a limpar
- text\_confidence : confiança de texto mínima. Utilizado para verificar se blocos são vazios
- find\_delimiters : flag que indica se verificação por delimitadores é realizada apenas por verificação do atributo "type" ou utilizando o método "is\_delimiter" de OCR\_Tree
- find\_images : flag que indica se verificação por imagens é realizada apenas por verificação do atributo "type" ou utilizando o método "is\_image" de OCR\_Tree

### **Algoritmo:**

---

#### **Algorithm 4** Correção de interseções

---

```
1: blocks ← obtém nodos de nível 2
2: boxes_to_check =
3: checked_boxes = []
4: i = 0
5: current_box = None
6: // Cada bloco será comparado por interseções com todos os outros
7: while i < len(blocks) do
8:     if not current_box and block[i] not in checked_boxes then
9:         if block não é vazio ou é delimitador then
10:             current_box = block[i]
11:             i += 1
12:     else
13:         ocr_results, blocks ← remove bloco vazio
14:     end if
15: end if
16: if current_box and blocks[i].id != current_box.id then
17:     compare_box = blocks[i]
18:     if compare_box vazia dentro de current_box ou vice-versa then
19:         ocr_results, blocks ← remove bloco vazio
20:     else if caixas intersetam-se then
21:         if ambas as caixas têm texto, e intersetam-se em mais de 70% da sua área then
```

```

22:           // Une os dois blocos utilizando join_trees. Usa como base o bloco que tem maior
rácio da sua área em interseção

23:           ocr_results, blocks ← remove bloco não usado como base

24:       end if

25:   else

26:       // Remove área de interseção do bloco com menor rácio de área intersetada, usando o
método remove_box_area

27:   end if

28: end if

29: i += 1

30: if está a verificar último bloco e ainda faltam verificar mais, ou não tem current_box then

31:     // Escolhe próximo bloco a analisar

32:

33:     // Escolhe bloco ainda não verificado, não vazio

34:     if current_box is not None then

35:         i = 0

36:     end if

37: end if

38: end while

```

---

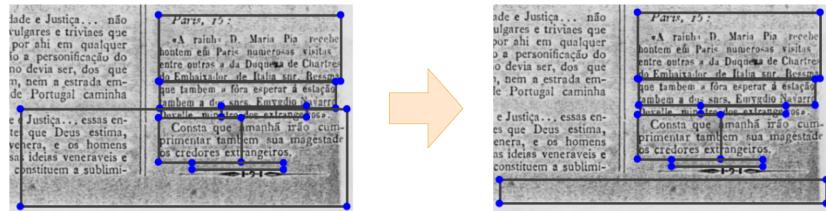


Figura 17: Exemplo de uso de block\_bound\_box\_fix (10 de confiança de texto mínima).

### **text\_bound\_box\_fix**

**Descrição:** Ajuste de bounding boxes para apenas englobar o texto confiáveis dentro delas.

#### **Argumentos:**

- ocr\_results : OCR Tree a ajustar
- text\_confidence : confiança de texto mínima. Utilizado para filtrar blocos de texto confiáveis

### **Algoritmo:**

---

#### **Algorithm 5** Cálculo de tamanhos de texto

---

```
1: blocks ← obtém nodos de nível 2
2: text_blocks ← filtra blocks para apenas ter blocos com texto
3: // Percorre blocos com texto para ajustar as BBs
4: for b in text_blocks do
5:     words ← obtém lista de palavras confiáveis, não vazias
6:     block_min_left ← valor da palavra mais à esquerda
7:     block_max_right ← valor da palavra mais à direita
8:     block_min_top ← valor da palavra mais acima
9:     block_max_bottom ← valor da palavra mais abaixo
10:    if valores calculados ajustam caixa para ser menor then
11:        b.box ← atualiza tamanho da caixa e dos seus filhos
12:    end if
13: end for
```

---

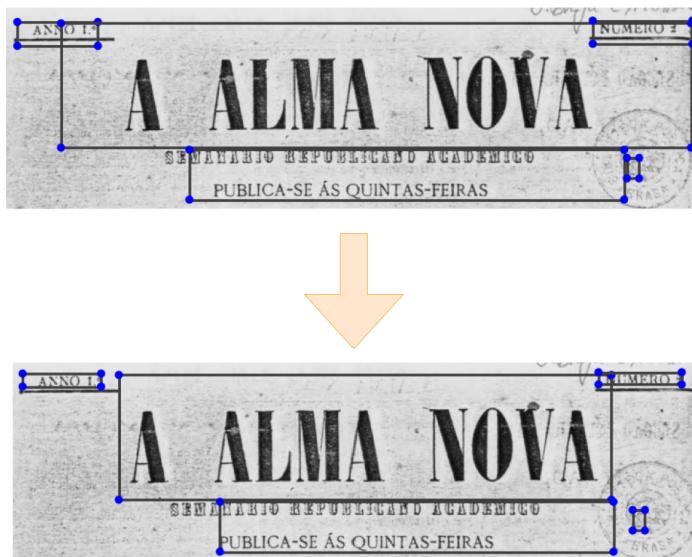


Figura 18: Exemplo de uso de `text_bound_box_fix` (10 de confiança de texto mínima).

### **block\_bound\_box\_fix\_image**

**Descrição:** Ajuste de bounding boxes realizando análise da imagem correspondente. Para cada bloco procura, na correspondente zona da imagem, reduzir as suas dimensões de acordo com os primeiros pixels pretos encontrados.

### **Argumentos:**

- ocr\_results : OCR Tree a ajustar
- target\_image : imagem correspondente à OCR Tree
- level : nível de bloco a ajustar
- text\_confidence : confiança de texto mínima. Utilizado para filtrar blocos de texto confiáveis

### **Algoritmo:**

Para cada aresta de um bloco, procura, na zona correspondente à bounding box, na imagem analisar, seguindo uma direção dependente da aresta, a ocorrência do primeiro pixel preto que tem seu vizinho nessa mesma direção outro pixel preto.

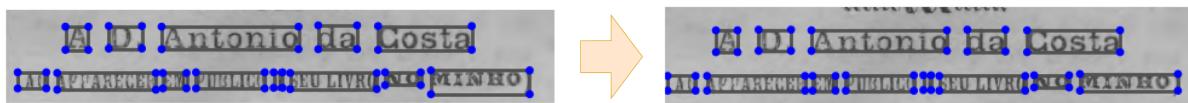


Figura 19: Exemplo de uso de bound\_box\_fix\_image (10 de confiança de texto mínima).

### **remove\_solo\_words**

**Descrição:** Remove blocos com uma única palavra que estão dentro de outros blocos. Estes são blocos que por esta característica podem ser identificados como ruído ou, em caso negativo, iriam causar conflito no output final por não estarem propriamente integrados no bloco devido.

### **Argumentos:**

- ocr\_results : OCR Tree a limpar
- conf : confiança de texto utilizada para análise a nível das palavras

### **unite\_blocks**

**Descrição:** Une verticalmente blocos com o mesmo valor de atributo "type". Apenas aplica união entre blocos adjacentes e horizontalmente alinhados. Nota : blocos de texto vertical, apenas podem unir com blocos de texto vertical.

### **Argumentos:**

- ocr\_results : OCR Tree a limpar
- conf : confiança de texto utilizada para verificação de blocos vazios

### **Algoritmo:**

---

**Algorithm 6** União de blocos

---

```
1: blocks ← obtém nodos de nível 2
2: target_block ← escolhe bloco não visitado
3: // Percorre blocos com texto para ajustar as BBs
4: while não tiver visitado todos os blocos do
5:     united = False
6:     bellow_blocks ← lista de blocos adjacentes, diretamente abaixo
7:     bellow_blocks ← filtra pelos blocos do mesmo tipo que target_block e horizontalmente alinhados
8:     if bellow_blocks then
9:         if target_block é texto vertical then
10:             bellow_blocks ← filtra para apenas manter texto vertical
11:         end if
12:         if len(bellow_blocks) == 1 then
13:             target_block ← une com bloco
14:             ocr_results ← remove bloco unido
15:             // atualiza lista de não visitados
16:         end if
17:     end if
18:     // Se não tiver unido, escolhe novo bloco, senão repete verificação
19:     if not united then
20:         target_block ← escolhe próximo bloco não visitado
21:     end if
22: end while
```

---

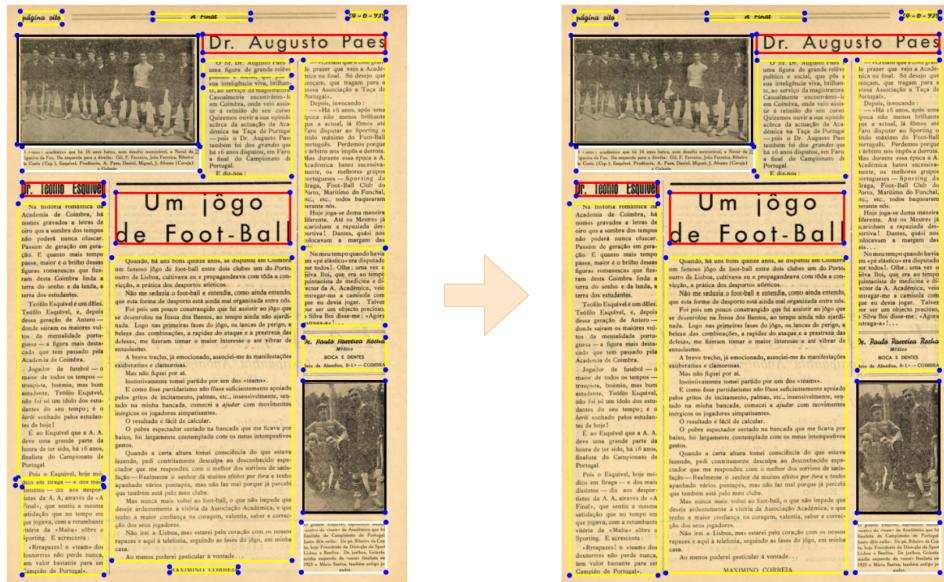


Figura 20: Exemplo de uso de `unite_blocks` (10 de confiança de texto mínima).

### `split_whitespaces`

**Descrição:** Separa blocos com espaçamento entre palavras suficientemente grande. Apenas realiza divisão perpendicular ao eixo x. Tem em conta uma análise do texto para calcular o espaçamento médio de palavras e um valor de ratio de espaço vazio para realizar divisão, analisa a presença de espaços brancos em blocos com texto e divide-os quando as condições se verificam. No caso de blocos com múltiplas linhas, tem de verificar se existe um espaço vazio comum válido.

#### Argumentos:

- `ocr_results` : OCR Tree a limpar
- `conf` : confiança de texto utilizada para verificação de blocos vazios e análise de texto
- `dif_ratio` : ratio de espaço entre palavras para realizar divisão

#### Algoritmo:

##### Algorithm 7 Divisão por espaços vazios

```

1: text_analysis ← analise do texto
2: avg_word_dist = text_analysis['average_word_distance']
3: blocks ← obtém blocos com texto
4: // Percorre blocos com texto para ajustar as suas BBs
5: for block in blocks do
6:   lines ← obtém linhas do bloco
7:   line_seq_positions = []

```

```

8:    valid_split = True
9:    // Para cada linha procura espaços vazios para divisão e guarda as coordenadas destes
10:   for line in lines do
11:     line_words ← obtém palavras da linha
12:     line_seq_position = [None,None]
13:     line_word_dists ← guarda as distancias entre palavras na linha
14:     line_word_pairs ← guarda pares de palavras
15:     if line_word_dists then
16:       average ← media entre a media do tamanho das palavras e o avg_word_dist
17:       line_seq_position ← procura primeiro espaço que valide a regra  $dist \geq dif\_ratio * average$ 
18:       if not line_seq_positio then
19:         // todas as linhas têm de ter um espaço válido
20:         valid_split = False
21:       else
22:         line_seq_positions.append(line_seq_position)
23:       end if
24:     end if
25:   end for
26:   if valid_split and line_seq_positions then
27:     // verifica se todos os intervalos guardados intersetam
28:     // em caso positivo, verifica o máximo tamanho da divisão a fazer
29:     widest_interval ← espaço entre palavras mais longo
30:     interception ← verifica se todas as linhas têm um espaço que está dentro de widest_interval
31:     if interception then
32:       left ← ponto mais à esquerda entre todos os espaços
33:       right ← ponto mais à direita entre todos os espaços
34:       division_line ← Box vertical representante da linha de divisão
35:       // realizar divisão
36:       blocks = split_block(block,delimiter,orientation='vertical',keep_all=True,conf=conf)
37:       ocr_results ← atualiza OCR Tree, adicionando novo bloco resultante da divisão
38:     end if
39:   end if
40: end for

```

---

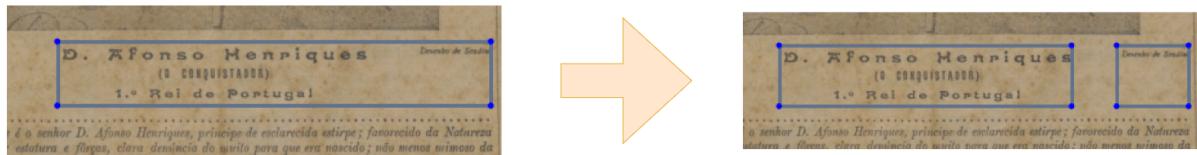


Figura 21: Exemplo de uso de split\_whitespaces.

## delimiters\_fix

**Descrição:** Correções de blocos de tipo 'delimiter'. Especialmente útil para limpezas finais de blocos resultantes do método 'get\_document\_delimiters'. Remove delimitadores dentro de potenciais imagens; Divide texto com delimitador dentro ou quase totalmente dentro deste, se divisão for válida. Ajusta bounding boxes de delimitadores para remover interseções com blocos de texto.

### Argumentos:

- ocr\_results : OCR Tree a limpar
- conf : confiança de texto utilizada para verificação de blocos vazios e análise de texto

### Algoritmo:

---

#### Algorithm 8 Limpeza de um delimitador

---

```

1: delimitador ← delimitador a ser verificado
2: block ← bloco a ser comparado
3: if delimitador dentro ou quase completamente (70%) dentro do bloco then
4:   if bloco não tem texto then
5:     // remove delimitador
6:   else
7:     if divisão do bloco válida pelo delimitador, i.e.: delimitador não intersetava, de acordo com a sua orientação, nenhum
       texto do bloco no seu eixo de divisão, e áreas resultantes do corte têm texto then
8:       block ← divide o bloco em 2 utilizando o delimitador como linha de corte
9:     else
10:      // remove delimitador
11:    end if
12:  end if
13: else if delimitador intersetava bloco then
14:   if se divisão válida possível then
15:     block ← divide o bloco em 2 utilizando o delimitador como linha de corte
16:   else
17:     delimitador ← ajusta bb do delimitador
18:   end if

```

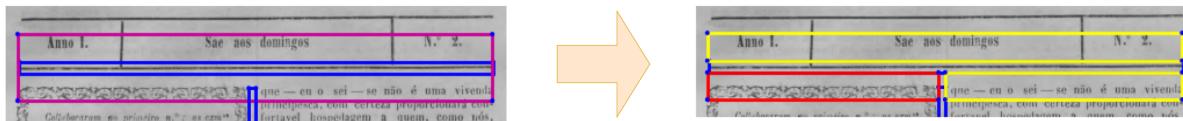


Figura 22: Exemplo de uso de delimiters\_fix.

### 5.2.4 Categorização de blocos

De forma a poder segmentar com maior precisão documentos, informação adicional sobre os blocos reconhecidos é necessária. Por exemplo, para ser possível identificar um artigo de um jornal, seria expectável agrupar um título e pelo menos uma caixa de texto. Desta forma uma tokenização dos blocos de nível 2 seguindo heurísticas foi desenvolvida.

O método que realiza esta tokenização é **categorize\_box**. O método foi desenvolvido com foco na utilização para jornais, sendo que os tipos de elementos identificados é portanto limitada.

A tokenização segue as seguintes regras:

- **text** : bloco com texto, de tamanho dentro do tamanho normal de texto (margem de 10%), ou que não verificou nenhuma das outras regras para blocos de texto
- **title** : bloco com texto, não vertical, com menos de 10 palavras e tamanho, no mínimo, duas vezes maior do que o tamanho normal de texto, ou largura de letra no mínimo 50% superior à largura média
- **highlight** : bloco com texto, semelhante a "title", mas com maior número de palavras
- **caption** : bloco com texto, de tamanho menor do que tamanho normal de texto e está diretamente abaixo de uma imagem ou de um bloco do tipo "caption"
- **delimiter** : bloco sem texto e que verifica a regra:  $box.width \geq 4 * box.height \vee box.height \geq 4 * box.width$
- **other** : bloco sem texto e que não verificou nenhuma das outras regras para blocos vazios

Blocos de tipo "delimiter" ou "image" são também possíveis de identificar utilizando métodos disponibilizados para processamento de imagem.

No caso de categorização de blocos de texto, são também verificadas certas características:

- **start\_text** : texto foi iniciado, i.e. começa com uma letra maiúscula ou com início de diálogo (sinais: ',', '-') seguido de uma letra maiúscula; e tem pelo menos uma letra minúscula no texto.

- **ends\_text** : texto foi terminado, termina com um sinal de pontuação que finaliza uma frase, ou fim de diálogo (".", "?", "!", ","); e tem pelo menos uma letra minúscula no texto.

Estas características de texto guardam informação relevante utilizada em métodos de cálculos de atração entre blocos. Nota-se que estas características são calculadas utilizando um valor mínimo de confiança de texto no valor de 40, dado a facilidade de OCR assumir ruído como pontuação.



Figura 23: Exemplo de uso de categorize\_blocks.

## 5.2.5 Divisão de blocos

Uma funcionalidade de extrema utilidade para a manipulação dos produtos de OCR é a capacidade de dividir um bloco em dois, cada um destes novos ficando com parte dos conteúdos do bloco original. Tal operação facilita a remoção de partes ruidosas de um bloco, ou a segmentação de bloco quando este foi incorretamente unido pelo software de OCR. Exemplo: bloco de texto cuja primeira linha tem potencial de ser título, permite facilmente dividir o bloco em o título e restante texto.

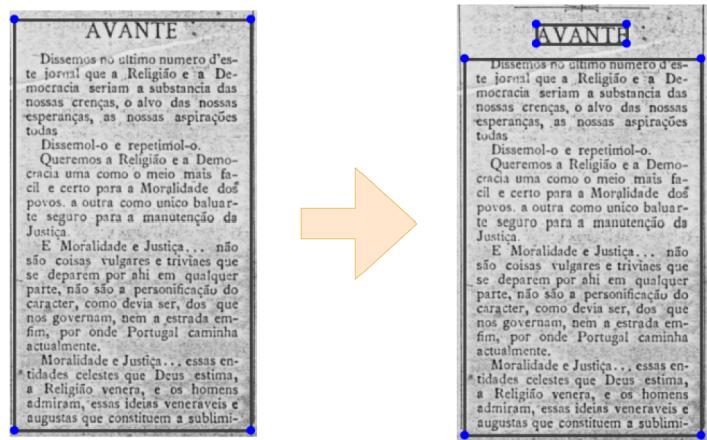


Figura 24: Divisão de bloco para separar título do texto (aplicou-se também um ajuste de bounding boxes para acentuar a diferença).

Esta é no entanto uma operação de relativa complexidade, visto que é necessário ir descendo na árvore e fazer múltiplas verificações para calcular o melhor destino final (entre os novos blocos) para cada um dos filhos.

A divisão implementada admite dois tipos de corte: horizontal, utilizando um corte perpendicular ao eixo y, tem uma divisão do conteúdo relativamente simples, visto apenas ser necessário verificar quais linhas se mantêm em cada novo bloco, refazendo posteriormente os parágrafos conforme necessário; vertical, utilizando um corte perpendicular ao eixo x, tem a divisão de conteúdo mais complexo, visto ser necessário verificar para cada palavra a que bloco pertence e, posteriormente, refazer as linhas e os parágrafos.

Uma aplicação direta desta funcionalidade será discutida sobre o editor OCR, ferramenta que permite uma visualização mais fácil desta transformação.

### **split\_block**

**Descrição:** Dado um bloco, um delimitador representante do corte a realizar e a orientação do corte pretendida, divide o bloco em 2. Reparte o conteúdo original entre os dois novos blocos de acordo com a área em que melhor se incluem. Dos blocos criados, devolve apenas aqueles que ainda tiverem conteúdo (texto).

#### **Argumentos:**

- block : OCR Tree a dividir
- delimiter : Box representante do corte a ser feito
- orientation : orientação do corte a realizar. Opções: "horizontal", "vertical".

- keep\_all : flag que indica se todo o conteúdo deve ser restaurado. Em caso negativo, conteúdo que não se inclui totalmente em nenhum dos novos blocos não será incluído. Se positivo, conteúdo em posição de conflito irá para o bloco que possui a maior parte da sua área.

### **Algoritmo:**

---

#### **Algorithm 9** Divisão de bloco em 2

---

```

1: new_blocks = [block]
2: if delimitador não interceta com bloco then return new_blocks
3: end if
4: area_1, area_2 ← cria 2 Box para os novos blocos, criadas de acordo com a orientação do corte e a posição do delimitador
5: // listas para parágrafos de cada área
6: blocks_1 = []
7: blocks_2 = []
8: // obtém todas as linhas e para cada uma verifica que palavras pertencem a qual área
9: block ← da id às palavras,linhas e parágrafos para as conseguir remover diretamente
10: blocks_1 ← copia parágrafos pertence ao bloco
11: blocks_2 ← copia parágrafos pertence ao bloco
12: for paragrafo in block do
13:     par_words ← obtém palavras do parágrafo
14:     for word in par_words do
15:         if word dentro de area_1 then
16:             blocks_2 ← remove palavra da area 2
17:         else if word dentro de area_2 then
18:             blocks_1 ← remove palavra da area 1
19:         else if keep_all then
20:             if word está mais incluída dentro da área 1 then
21:                 blocks_2 ← remove palavra da area 2
22:             else
23:                 blocks_1 ← remove palavra da area 1
24:             end if
25:         end if
26:     end for
27:     // remove linhas sem palavras dos parágrafos
28:     // atualiza BB dos parágrafos para representar as modificações
29: end for
30: if blocks_1 then
31:     block.children = blocks_1
32:     block ← atualiza BB para representar modificações do conteúdo
33: else

```

```

34: // se a área 1 não tiver blocos, bloco original é atualizado com a área 2 e só é devolvido um bloco
35: if blocks_1 vazio then
36:     block.children = blocks_2
37:     block ← atualiza BB para representar modificações do conteúdo
38: else
39:     // cria novo bloco
40:     new_block ← nova OCR Tree
41:     new_block.children = blocks_2
42:     new_block ← atualiza BB para representar modificações do conteúdo
43:     new_blocks = [block,new_block]
44: end if
45: end if

return new_blocks

```

---

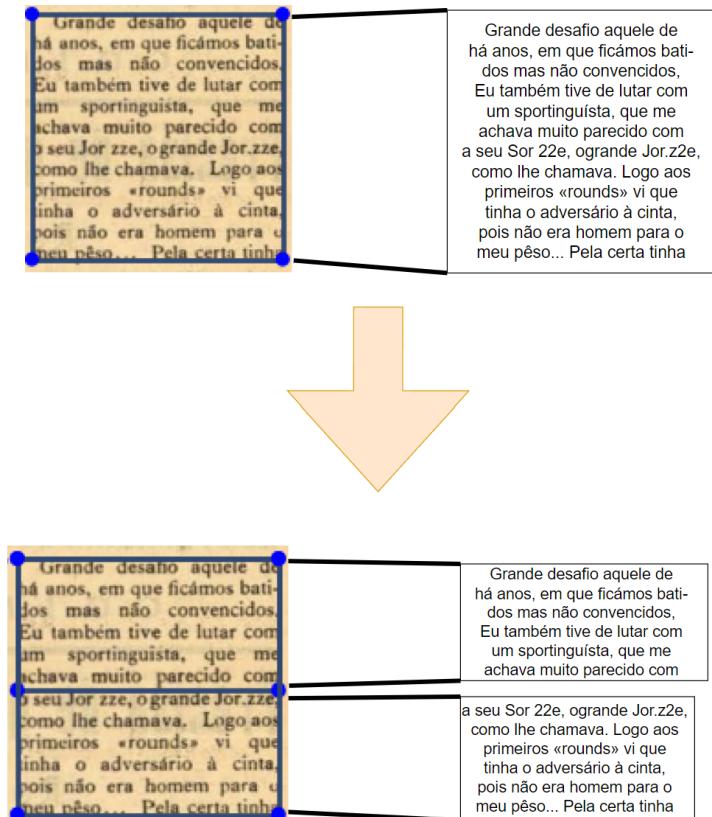


Figura 25: Exemplo de uso de `split_block` com corte horizontal.

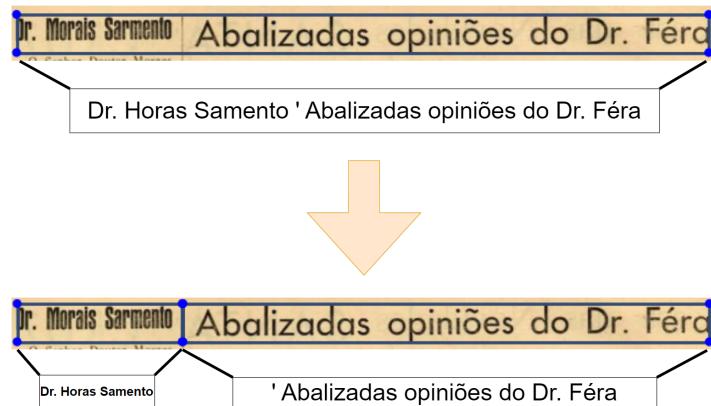


Figura 26: Exemplo de uso de `split_block` com corte vertical.

### **find\_text\_titles**

**Descrição:** Procura analisar blocos da OCR Tree por potenciais blocos de tipo título que podem ser separados de blocos de texto. Quando potenciais títulos são encontrados, divide o bloco original de modo a separar em bloco(s) texto e bloco título.

#### **Argumentos:**

- `ocr_results` : OCR Tree a analisar
- `conf` : confiança de texto mínima.

**Algoritmo:** Para os blocos de tipo 'text', analisa os seus filhos de nível linha, verificando se poderiam ser categorizados como 'title'. Em caso positivo, utiliza a função de divisão de bloco para separar o bloco 'title' do 'text'. A operação de divisão poderá resultar em 2 blocos, caso o 'title' se encontre no início ou fim do bloco; ou 3, no caso de estar inserido no meio do bloco.

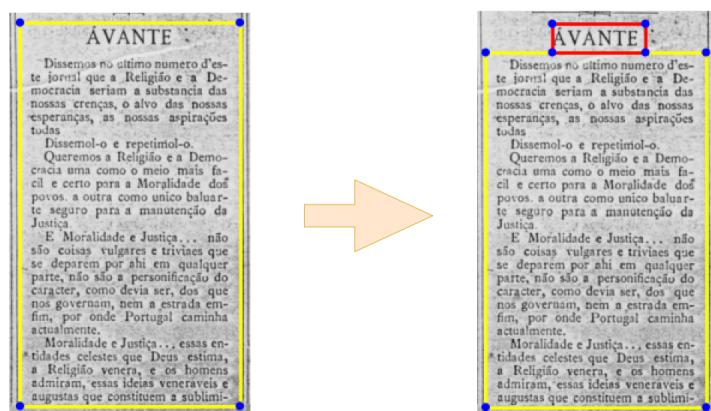


Figura 27: Exemplo de uso de `find_text_titles` com corte vertical.

### 5.2.6 Cálculo de ordem de leitura

No caso de documentos de leitura não linear, como é o caso de jornais, a ordem de leitura proposta pelos motores OCR pode muitas vezes ser incorreto.

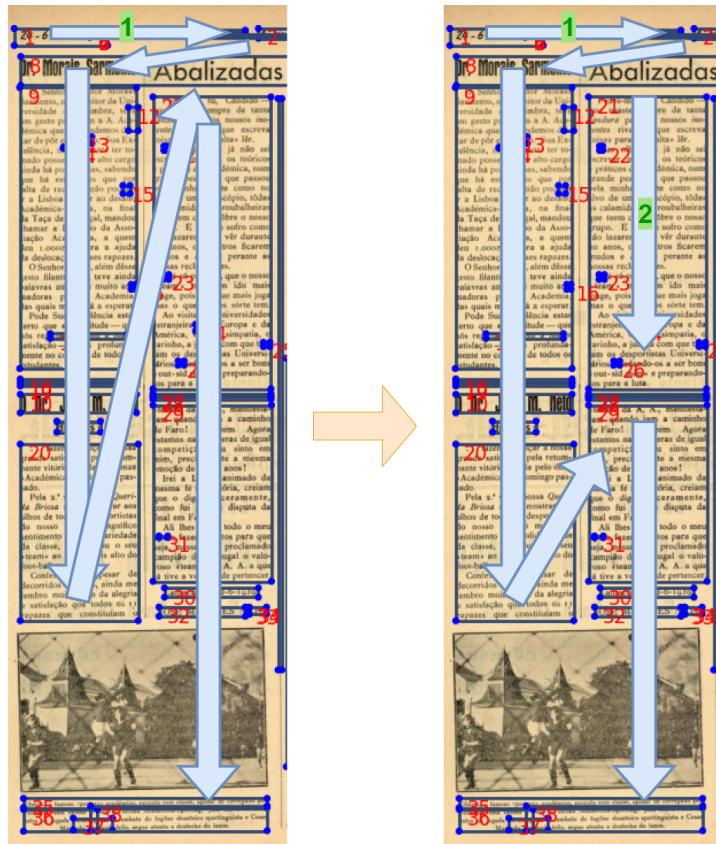


Figura 28: Exemplo de jornal incorretamente ordenado.

(a) Do lado esquerdo está a ordenação (marcado pelas setas azuis, por sua vez ordenadas pela numeração verde) resultante de OCR (tesseract). Do lado direito está a ordenação de leitura correta.

Foi então desenvolvido um algoritmo geral, baseado em grafos pesados, para calcular a ordem de leitura do corpo de um jornal. Este algoritmo procura, através de um conjunto de regras sobre as características dos blocos dos resultados OCR, como por exemplo: texto incompleto, texto seguido de título; calcular a atração entre blocos de acordo com as suas características e posição relativa (similar a ordenação topológica); e posteriormente ordenar os blocos, procurando agrupar artigos. Um artigo considera-se como um grupo de blocos, em que o primeiro é um título e não pode ter mais títulos dentro do grupo, a não ser que seja diretamente a seguir ao primeiro título (complementares).

Para conseguir este produto final foi então necessário criar uma estrutura de dados do tipo grafo, neste caso pesado; um método para calcular a atração entre nodos do grafo; e o algoritmo que calcula o

caminho do grafo.

## Estrutura de dados Graph

A estrutura de dados representante do grafo é composta pelas classes:

- Graph : representante do grafo geral

Possui as funcionalidades principais:

- verificações de existência de um nodo
- verificação de existência de caminho entre nodos
- limpeza de caminhos transitivos no grafo
- limpeza de caminhos de baixo peso : em nodos com múltiplos pais, se a atração para um pai é menos de metade do que a atração para outro, remove aresta

- Node : representante de um nodo do grafo Possui as funcionalidades principais:

- conjunto de pais
- conjunto de filhos
- atualizar pais e filhos
- cálculo de conexão de um nodo a outro
- limpeza de caminhos de baixo peso : em nodos com múltiplos pais, se a atração para um pai é menos de metade do que a atração para outro, remove aresta

- Edge : representante de um aresta do grafo

## Cálculo de atração entre blocos

O cálculo da atração entre dois blocos tem em conta as características dos dois blocos, assim como o contexto da sua vizinhança. O resultado final é o somatório de pontuações adquiridas pela verificação de um conjunto de regras, com diferentes níveis de importância.

Sendo o conjunto de verificações relativamente extenso, segue-se um compilado das condições verificadas, por ordem de relevância:

### Atração

- bloco 'image' tem muita atração a bloco 'caption' (vice-versa)
- bloco 'text' inacabado é muito atraído por bloco 'text' não começado, se este tiver mais de uma linha
- bloco é bastante atraído para blocos não em baixo se por baixo tiver um delimitador (relativo a quanto o delimitador engloba o bloco)

- bloco é bastante atraído para blocos não à direita se à direita tiver um delimitador (relativo a quanto o delimitador engloba o bloco)
- bloco é atraído por blocos abaixo dele
- bloco é atraído por bloco se bloco acima engloba os dois blocos
- bloco é atraído por bloco se bloco acima tem os dois blocos diretamente abaixo dele
- bloco é atraído por bloco se bloco abaixo engloba os dois blocos
- bloco é atraído por bloco se bloco abaixo tem os dois blocos diretamente acima dele
- bloco é atraído por bloco que o englobe (relativo ao rácio de área englobada)
- bloco 'title' é atraído por bloco não 'title'
- bloco 'title' é atraído por bloco abaixo
- bloco 'text' é atraído por bloco abaixo
- bloco 'title' é um pouco atraído por bloco 'text' com texto começado por baixo dele
- bloco é um pouco atraído pelo bloco por baixo mais à esquerda
- bloco 'text' é um pouco atraído pelo bloco à direita se não tiver blocos de texto diretamente por baixo

## **Repulsão**

- bloco não é bastante atraído por bloco à direita se tiver um delimitador diretamente à direita (relativo a quanto o delimitador engloba o bloco)
- bloco não é bastante atraído por bloco abaixo se tiver um delimitador diretamente abaixo (relativo a quanto o delimitador engloba o bloco)
- bloco 'text' não acabado não é atraído por bloco 'text' começado, se este tiver mais de uma linha

O método responsável por este cálculo é **calculate\_block\_attraction**.

## **Argumentos:**

- block : OCR Tree cuja atração por target\_block será calculada
- target\_block : OCR Tree a comparar
- blocks : lista de OCR Tree para servirem de contexto de vizinhança
- direction : direção entre block e target\_block, se nenhuma for dada, direção é calculada. Utilizado para cálculo de distância e verificação de condições. Opções : 'above', 'left','right','below'
- child : flag que indica se cálculo é de pai para filho ou vice-versa

## **Cálculo da ordem de leitura**

O cálculo da ordem de leitura é realizado pelo método **sort\_topologic\_order** que utiliza um algoritmo de ordenação topológica que utiliza os pesos das arestas para resolver conflitos.

Para escolher o primeiro bloco, ou quando não há mais blocos abaixo do bloco atual, utiliza o método `next_top_block` que procura dentro de uma lista de potências próximas blocos, o mais apropriado - normalmente o com menor distância ao canto superior esquerdo. Um filtro diferente pode ser fornecido a este método para a escolha deste próximo bloco, como é exemplo através da opção da pipeline de dar prioridade a títulos.

### Argumentos:

- `topologic_graph` : Graph com as ligações dos blocos da OCR Tree a ordenar
- `sort_weight` : flag que indica se ordenação é apenas topológica ou usa os pesos do grafo para resolver situações de desempate
- `next_node_filter` : filtro que pode ser fornecido para alterar o método de escolha do próximo quando não se possui uma âncora

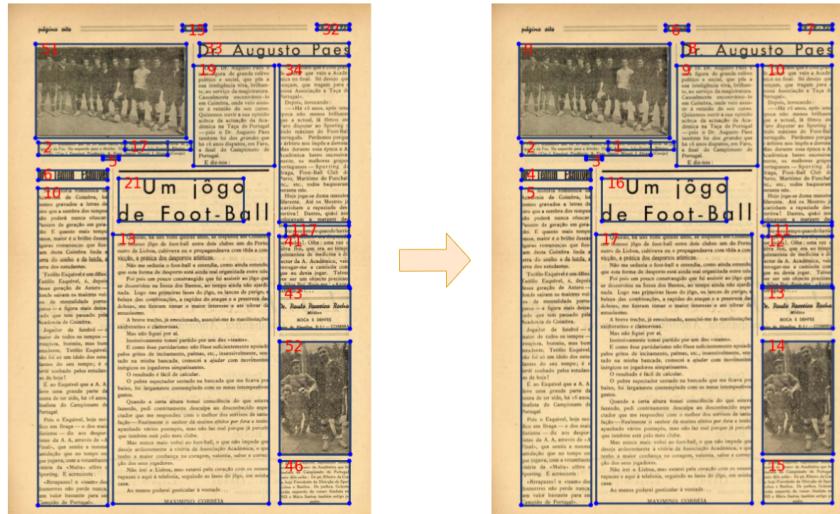


Figura 29: Exemplo de cálculo da ordem de leitura de um jornal (exemplo não linear).

(a) Figura esquerda é a ordem original; figura direita é a ordem calculada. Realça-se que do lado esquerdo os ids são indicativos da sequência, não correspondendo ao número total de blocos devido a transformações e limpezas realizadas para um melhor resultado do cálculo da ordem.

Os resultados desta ordenação é bastante dependente da qualidade do conteúdo da OCR Tree, sendo que uma OCR Tree com texto muito disperso ou ruidoso irá ter mais chance de gerar uma ordenação errada.

Diferentes jornais podem não ter métodos de leitura mais complexos, ou que não acomodem estas regras, sendo que esta metodologia não será ubíqua.

### 5.2.7 Segmentação de resultados

No caso de jornais, como já discutido anteriormente, o elemento base da informação nestes contido são os artigos.

Artigos podem ser descritos como um título seguido por um excerto de elementos de outro tipo, por exemplo texto ou imagem, quebrado pelo final de página ou por um novo título que iniciaria outro artigo. Segundo esta caracterização foi desenvolvida um método que, utilizando os resultados do cálculo da ordem de leitura i.e., o grafo e a ordem de nodos calculada, procura formar uma lista de artigos.

#### graph\_isolate\_articles

**Descrição:** A partir de um grafo de nodos de blocos e uma ordem de leitura destes, procura percorrer esta lista para gerar artigos utilizando todos os blocos. Um artigo é começado por um bloco título e, se um novo bloco título for iterado na lista e a lista tiver blocos não título, inicia um novo artigo.

#### Argumentos:

- graph : Graph com as ligações dos blocos da OCR Tree
- order\_list : list de ordem de leitura dos blocos da OCR Tree



Figura 30: Exemplo de isolamento de artigos num jornal. Foram encontrados 3 artigos pelo algoritmo, cada um identificado com uma cor única.

Métodos para conversão da lista de artigos para texto, markdown e visualização sob uma imagem foram também desenvolvidos.

## 5.3 Processamento de imagem

Um outro ponto de abordagem natural quando se trabalha com OCR, é o tratamento do input, que na maior parte dos casos se trata de uma imagem. O estado deste input inicial terá a maior influência na qualidade dos resultados ao longo de todo o processo.

Os principais problemas que se procurou abordar neste trabalho em relação ao tratamento de imagens de documento foram: correção automática de rotações do documento em relação ao texto; remoção de sombras de bainha nas bordas do documento; métodos de binarização para documentos antigos e em mau estado; segmentação de documentos, focando em jornais (divisão entre header e body, divisão de colunas); identificação de imagens/ilustrações no documento; identificação de delimitadores/linhas divisórias no documento.

Além disso, soluções já existentes para a resolução de problemas como: upscaling, denoising e ajustes na iluminação de imagem; foi reutilizado, embora maioritariamente para o uso na pipeline desenvolvida (6).

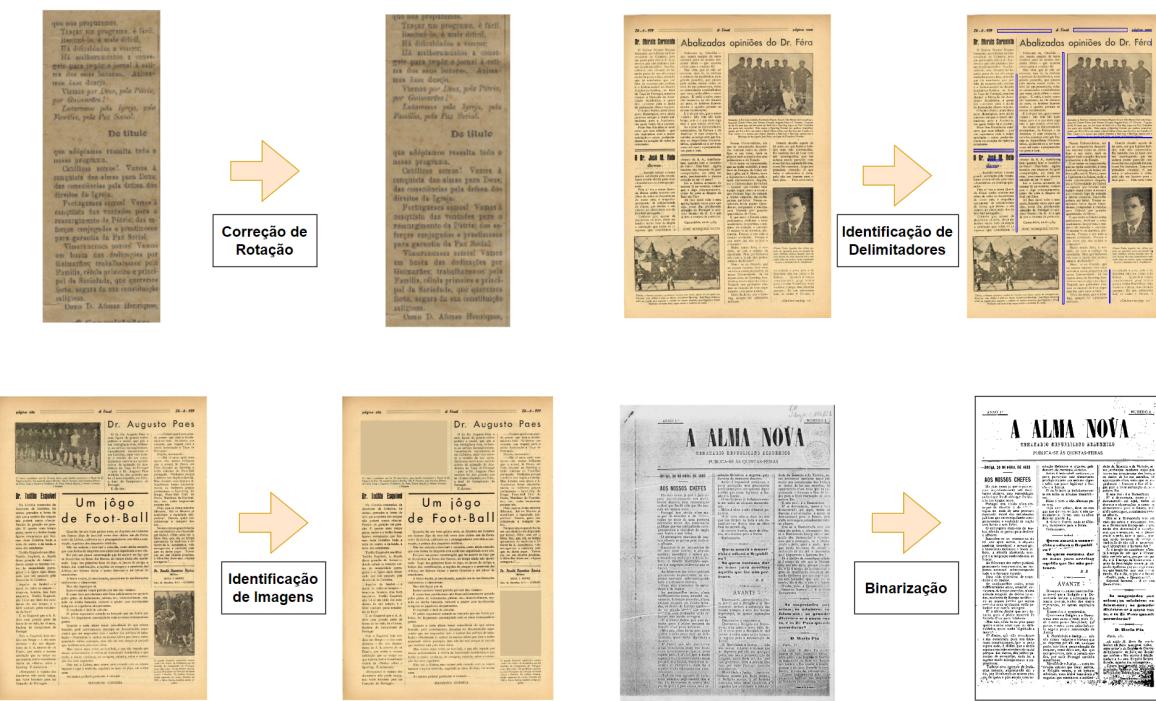


Figura 31: Exemplo de aplicação de métodos de processamento de imagem do toolkit.

Esta secção irá abordar os métodos desenvolvidos neste sentido.

### 5.3.1 Correção de ângulo de rotação

Para imagens de alta definição, i.e. normalmente com dpi elevado e, portanto, mais favorável para realização de OCR, notou-se que mesmo pequenas inclinações no texto podem resultar em resultados de OCR extremamente ruidosos e desorganizados.

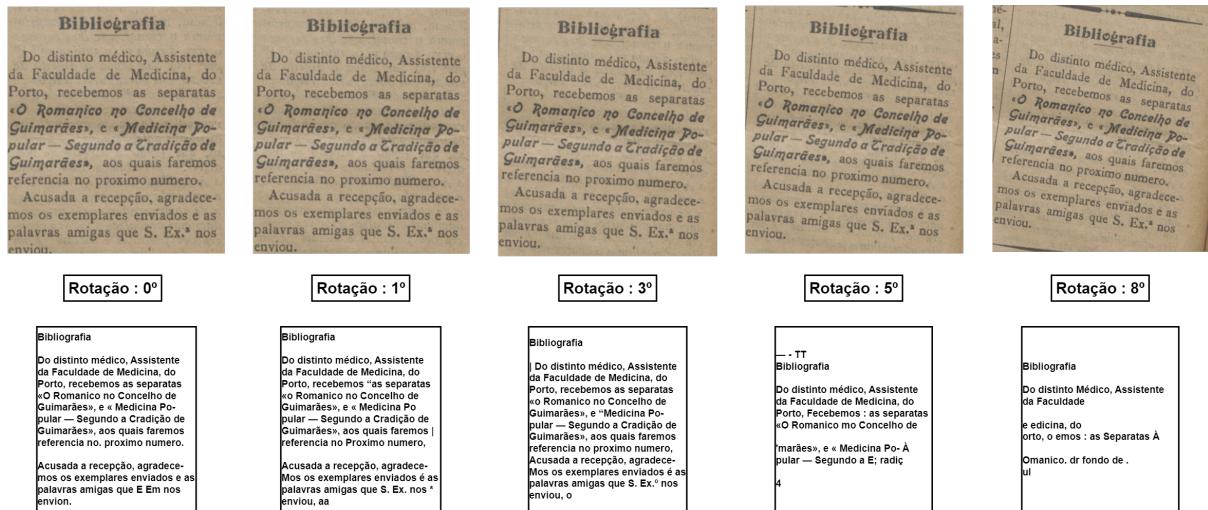


Figura 32: Variação dos resultados de OCR em diferentes níveis de rotação de texto.

Foi então desenvolvido um algoritmo com o propósito de automaticamente detetar a existência de uma rotação do texto e, calculando o seu ângulo, corrigi-la.

Este algoritmo foi baseado maioritariamente na solução proposta por [Bieniecki et al. \[2007\]](#) (discutida na secção de estado da arte) com adaptações adicionadas para complementar pontos não explicados da solução e para ser mais viável para documentos com algum ruído.

Na sua essência o algoritmo procura, num documento com texto, analisar as posições do primeiro pixel preto (expectável a primeira instância de texto estar do lado esquerdo) à esquerda em diferentes linhas, para tentar desenhar uma linha, na qual a inclinação será calculada. As adaptações realizadas serviram para fazer uma decisão inicial sobre a direção da rotação, e para, assumindo que se trabalha com imagens com ruído, procurar a linha mais confiável para o cálculo da inclinação.

Para ajustes mais minuciosos, também é aplicado o método de ajuste de rotação da biblioteca leptotica. Este método por si só não é capaz de corrigir ângulos mais consideráveis (+- acima de 6 graus).

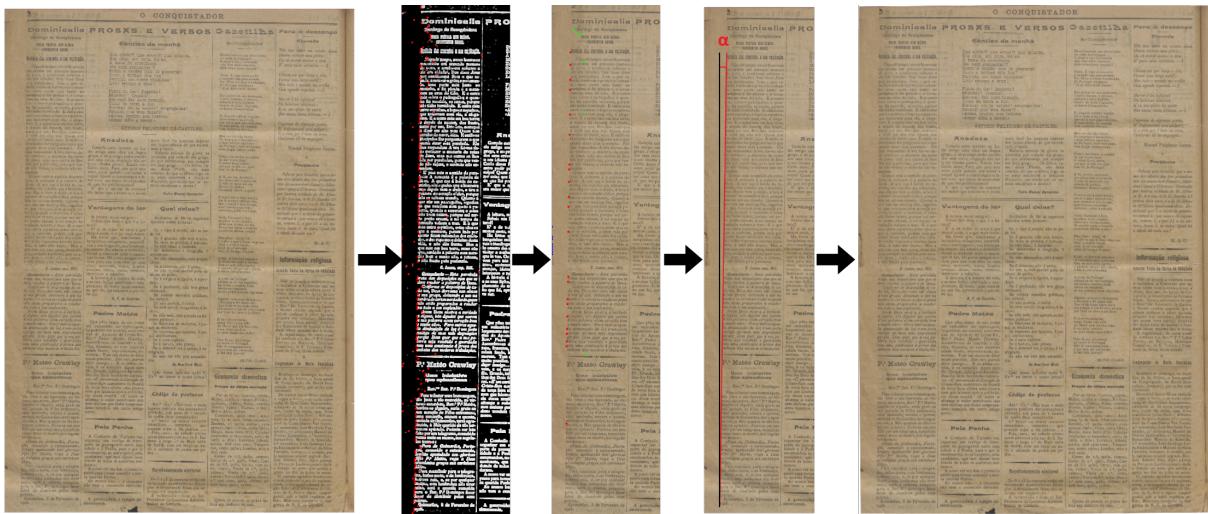


Figura 33: Exemplo de aplicação do algoritmo.

(a) **1<sup>a</sup>** imagem representa a imagem do documento original. **2<sup>a</sup>** imagem representa a identificação dos pixels à esquerda no documento. **3<sup>a</sup>** apresenta os 3 maiores sets de cada tipo: set de pixels com mesmo x (azul); set de pixels counter-clockwise (verde); set de pixels clockwise (vermelho). **4<sup>a</sup>** mostra o ângulo de rotação calculado a partir do maior set. **5<sup>a</sup>** tem a imagem corrigida.

### **calculate\_rotation\_direction**

**Descrição:** Calcula a direção de rotação num documento de texto. Retorna um dos valores: 'counter-clockwise'; 'clockwise'; 'none'. Cria conjuntos de pontos alinhados (que possibilitem traçar uma reta) para cada uma das direções e, de acordo com qual das direções tem o maior conjunto retorna o identificador dessa direção.

#### **Argumentos:**

- image : imagem do documento a analisar
- line\_quantetization : valor de quantetização do eixo y da imagem a usar (para a análise dos pixels)

### **rotate\_image**

**Descrição:** Corrige rotação numa imagem de um documento, se houver. Retorna a imagem com a rotação aplicada. Dada uma direção de rotação, calcula os conjuntos de pixels de texto que formam linhas e com essa dada inclinação no documento. Escolhe o maior destes conjuntos para formar uma linha e calcular a sua inclinação. Aplica uma rotação na imagem para corrigir a inclinação detetada. Se possível, aplica o método *pixFindSkewAndDeskew* de leptonica para correções menores.

#### **Argumentos:**

- image : imagem do documento a analisar
- line\_quantetization : valor de quantetização do eixo y da imagem a usar (para a análise dos pixels)
- direction : direção da rotação. 'counter-clockwise', 'clockwise' ou 'auto'. Se auto, aplica o método acima para calcular a direção.
- auto\_crop : flag para aplicação do método de cropping de possível bainha na imagem. Estas quando presentes afetam consideravelmente o algoritmo pois não costumam estar alinhadas com o texto.

### 5.3.2 Corte de sombra nas margens

No caso de documentos com múltiplas folhas, como livros ou jornais, é comum a presença de uma sombra nas margens das páginas. Esta pode, como mencionado anteriormente, impactar a solução para correção de rotação de documento, mas também, quando aplicado OCR, esta sombra poderá ser detetada como um elemento (um tipo de delimitador ou figura) que na realidade gera ruído extra. Pode ser então útil a remoção desta secção da imagem.



Figura 34: Exemplo de documento com inclinação de texto e sombra de bainha.

O algoritmo de remoção segue uma lógica semelhante aos métodos de análise de tamanhos de texto ou de deteção de colunas anteriormente explicados. Na imagem binarizada, é analisada a frequência de pixels pretos na imagem e, se nas suas extremidades esquerda ou direta (10% de cada uma das pontas) for detetado um pico seguido de uma descida muito acentuada (igual ou menor a 10% da frequência máxima de pontos pretos) é recortado a zona desse pico da imagem.

## **cut\_document\_margins**

**Descrição:** Procura cortar sombras de bainha de uma imagem de um documento. Se estas não existirem, recortam margem da imagem para se aproximar do texto, se este for adjacente a uma zona vazia. Retorna uma instância do tipo Box com as dimensões da imagem recortada.

### **Argumentos:**

- image : imagem do documento a analisar
- method : método de smoothing. Opções : WhittakerSmoothen (por defeito), savgol\_filter

### **5.3.3 Binarização de imagem**

Como foi discutido no estado da arte sobre o processamento de imagens, não existe um método de binarização que funcione de forma geral para qualquer tipo de documento, visto que muitas vezes estes são adaptativos e diferentes distribuições dentro da imagem vão gerar resultados diferentes.

Assim sendo, o trabalho realizado neste sentido não foi com o objetivo de propor um método generalista, mas sim de decidir métodos úteis no contexto do trabalho.

Com o trabalho realizado, conclui-se que para documentos com pouco ruído, ou onde este se apresente distribuído pela imagem sem grandes focos particulares, um denoising de médias não locais seguido de uma binarização com algoritmo de Otsu, obtém resultados satisfatórios. Esta binarização é similar à realizada pelo Tesseract. No caso de o ruído estar focado num local específico da imagem, verificou-se que binarizações mais extremas no thresholding, como estilo fax, conseguem obter melhor resultado nas zonas mais afetadas.

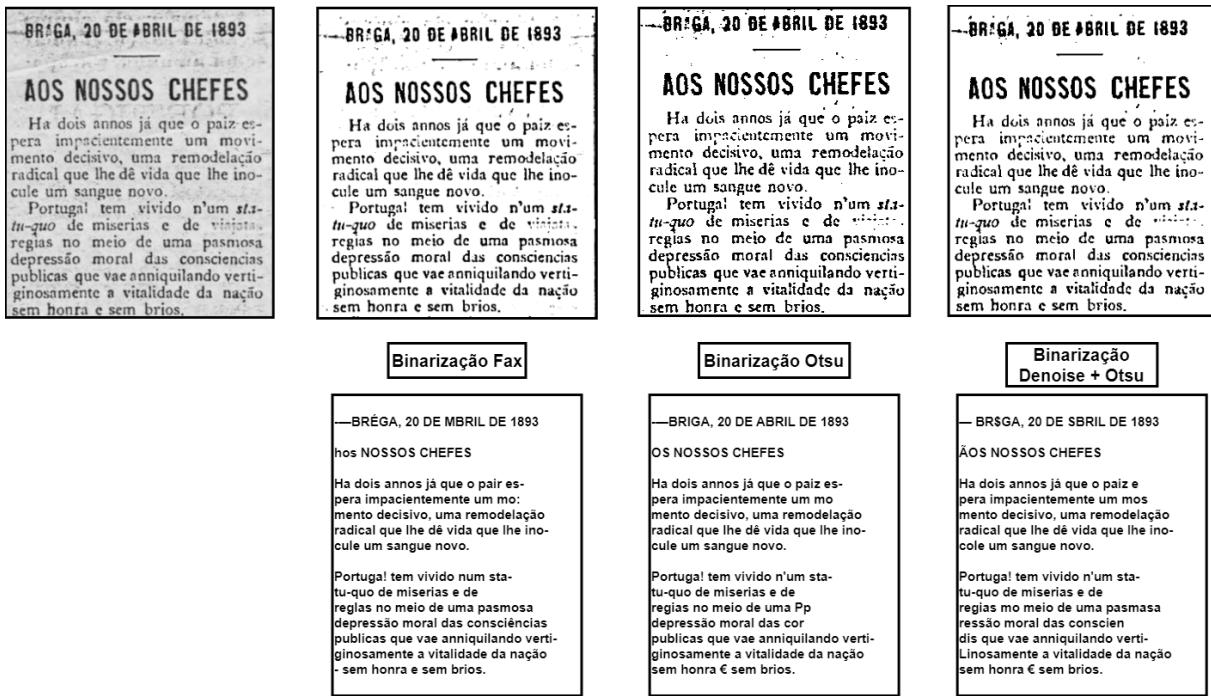


Figura 35: Exemplo de resultado de OCR com diferentes binarizações em local sem ruído muito acentuado.

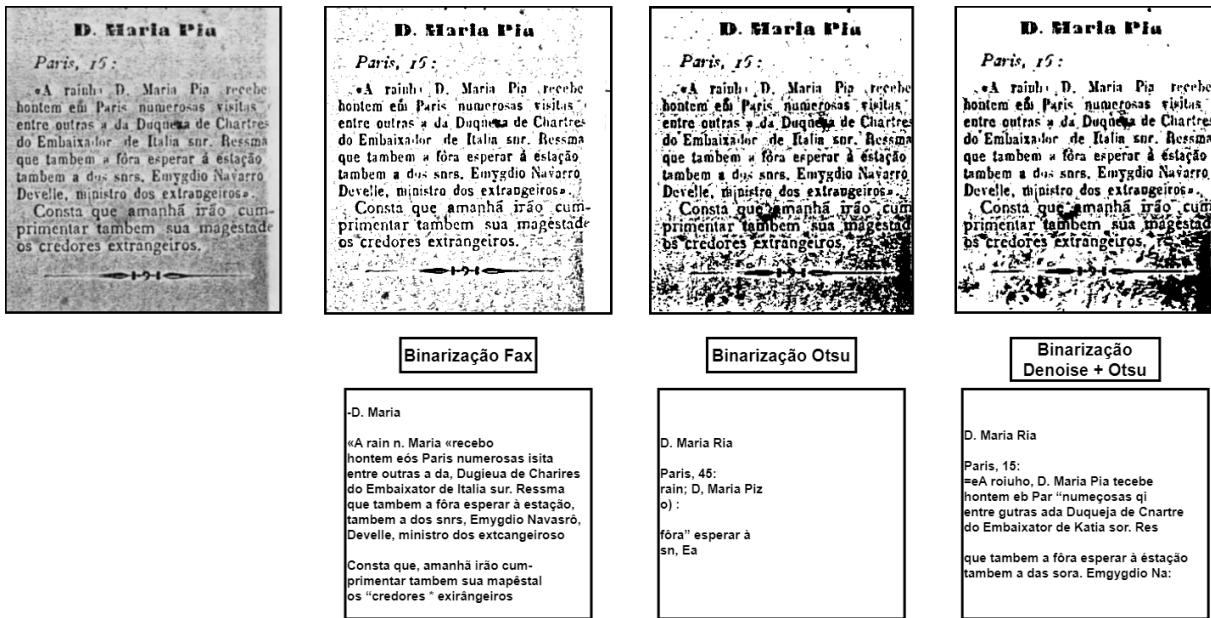


Figura 36: Exemplo de resultado de OCR com diferentes binarizações em local com muito ruído.

Um método para cada uma destas binarizações foi desenvolvido.

## binarize

**Descrição:** Binarização utilizando tresholding de Otsu. Realiza limpeza de ruído utilizando algoritmo de médias não locais.

### **Argumentos:**

- image : imagem a binarizar
- denoise\_strength : valor de intensidade a utilizar para denoising
- invert : flag para inverter cor de binarização

### **binarize\_fax**

**Descrição:** Binarização utilizando algoritmo de emulação de fax. Algoritmo de emulação proposto pelo orientador Prof. José João. Utilizando métodos do image magick pode ser conseguido como : convert <image> -colorspace Gray ( +clone -blur 15,15 ) -compose Divide\_Src -composite -level 10%,90%,0.2.

### **Argumentos:**

- image : imagem a binarizar
- g\_kernel\_size : tamanho do kernel gaussiano utilizado para o cálculo da imagem desfocada
- g\_sigma : valor de sigma do kernel gaussiano utilizado para o cálculo da imagem desfocada
- black\_point : valor de thresh de ponto preto, utilizado para ajustar os níveis da imagem
- white\_point : valor de thresh de ponto branco, utilizado para ajustar os níveis da imagem
- gamma : valor usado para ajuste do gamma após ajustar níveis da imagem
- is\_percentage : flag que indica se valores de black e white point são para serem considerados como percentagem
- invert : flag para inverter cor de binarização

### **Algoritmo:**

Calcula-se primeiramente uma versão desfocada da imagem. Esta será usada para se criar uma distinção entre o plano de fundo e o texto da imagem, ao se dividir a imagem original pela desfocada. Utiliza-se 2 kernels gaussianos por defeito, um aplicado a cada eixo, de dimensão 15 e sigma 15.

Em seguida ajusta-se os níveis dos valores pretos e brancos da imagem de forma a estarem contidos dentro dos valores definidos, 10% e 90% da escala disponível respetivamente.

Por fim, ajusta-se o gamma da imagem para a escurecer.

### **5.3.4 Identificação de delimitadores**

Para documentos estruturados como jornais, certos elementos são utilizados de forma geral para realçar a dada estrutura, como é o caso de delimitadores.



Figura 37: Página de jornal com delimitadores identificados (a vermelho).

Pode então ser útil conseguir identificar estes elementos para facilitar o seu uso, por exemplo, no cálculo de ordem de leitura de um jornal e cálculo de atração entre blocos.

Estes delimitadores são, por norma, linhas verticais ou horizontais, podendo ser filtradas utilizando um algoritmo similar ao desenvolvido pelo OpenCV [[opencv extract lines](#)]. Este algoritmo é utilizado para separar linhas horizontais do resto da imagem. Adaptando para o propósito de identificar as linhas horizontais, é necessário, após separar as linhas do resto da imagem, utilizar o algoritmo de Hough para identificar as posições e dimensões das linhas existentes.

### **get\_document\_delimiters**

**Descrição:** Identifica e devolve uma lista de delimitadores em uma imagem. Delimitadores são linhas horizontais ou verticais. Lista devolvida são objetos do tipo Box.

#### **Argumentos:**

- image : imagem a analisar
- tmp\_dir : diretório usado para armazenar ficheiros temporários
- min\_length\_h : valor de comprimento mínimo para um delimitador horizontal válido
- min\_length\_v : valor de comprimento mínimo para um delimitador vertical válido

- max\_line\_gap\_h : valor de descontinuidade mais comprida aceite para um delimitador horizontal válido
- max\_line\_gap\_v : valor de descontinuidade mais comprida aceite para um delimitador vertical válido
- reduce\_delimiters : flag para reduzir/limpar lista de delimitadores, procurando remover ruído

### **Algoritmo:**

---

#### **Algorithm 10** Identificação de delimitadores (apenas horizontais)

---

```

1: image ← imagem binarizada
2: morph ← aplica dilatação geral, seguido de linhas horizontais, aplica-se filtro de estruturas para filtrar linhas horizontais
   e, por fim, aplica-se nova dilatação para acentuar as linhas
3: edges ← método Canny para detecção de contornos, seguido de acentuação destes por dilatação
4: horizontal_lines = cv2.HoughLinesP(edges,1,np.pi/100,50,None,minLineLength=min_length_h,maxLineGap=max_line_gap_h)
5: lines = horizontal_lines
6: delimiters = []
7: for line in lines do
8:   // verifica se linhas são verticais ou horizontais
9:   // compara razão entre variação de x ou de y (declives da reta) com tan(60) e tan(5) respetivamente
10:  is_vertical = dy != 0 and abs(dx/dy) < t5
11:  is_horizontal = dx != 0 and abs(dy/dx) < t60
12:  if is_vertical or is_horizontal then
13:    delimiters ← adiciona Box para linha
14:  end if
15: end for
16: delimiters ← remove delimitadores na borda (provavelmente sombras)
17: if reduce_delimiters then
18:   delimiters ← aplica limpezas sobre lista: remove delimitadores com muitas interseções, une delimitadores próximos
      e com a mesma direção, remove delimitadores com muitas componentes ligadas (provavelmente texto)
19: end if
20: delimiters ← remove delimitadores demasiado curtos (menos de 5% do tamanho da imagem)

```

---

Na generalidade dos casos, a quantidade de linhas detetadas será muito superior aos delimitadores esperados, devido ao o ruído, texto do documento e morfologias aplicadas. Diferentes valores de parâmetros para o algoritmo de Hough variam estes resultados.



Figura 38: Resultados de identificação de linhas verticais e horizontais.

É então recomendado uma aplicação de métodos para reduzir o número de delimitadores encontrados. Os que foram desenvolvidos foram:

- remover interseções**: se 2 delimitadores com diferente orientação intersetam, remove o menor
- une delimitadores**: une delimitadores de igual orientação e suficientemente próximos
- remover linhas com demasiadas componentes ligadas**: linhas com muitas componentes ligadas, são provavelmente texto, ruído ou outro tipo de elemento, sendo portanto removidas



Figura 39: Resultados de identificação de delimitadores, aplicados métodos de redução.

Como se pode observar na figura 39, embora os métodos de redução filtrem de forma considerável a quantidade de delimitadores identificados, o resultado ainda contém elementos que não devia. Vários destes são resultados de texto de maior grossura, sendo portanto possível ainda realizar uma limpeza posterior onde, tendo em conta os resultados de OCR, se resolvem conflitos de delimitadores inseridos em blocos de texto. Um método deste tipo está disponível no Toolkit (delimiters\_fix).

### 5.3.5 Identificação de imagens no documento

A possibilidade de identificar e isolar elementos de imagem/ilustração em documentos é, por si só, uma ferramenta extremamente útil. Infelizmente nem todos os motores OCR identificam diretamente estes elementos como sendo imagens, por exemplo, no caso do Tesseract imagens são rodeadas com boa precisão porém, no ficheiro resultante de OCR, não é atribuído um identificador para isolar o elemento como uma imagem (de igual modo para os delimitadores).

Assim sendo, explorou-se abordagens para se obter esta informação de forma mais focada. Entre estas optou-se por soluções apoiadas por Deep Learning, como é o caso de Detectron desenvolvido pela Meta, e o Layout Parser que surge a partir do Detectron. Esta, embora apresente bons resultados de forma consistente, apresenta uma instalação complexa.

Uma solução mais simples em termos de dependências/installação e computação é o uso de um mé-

todo de segmentação de página disponibilizado pela biblioteca Leptonica ([método de segmentação](#)). Este método identifica, através de múltiplas aplicações de morfologias, bounding boxes para zonas previstas de texto e imagem. O código original foi apenas levemente modificado para alterar o destino do output.

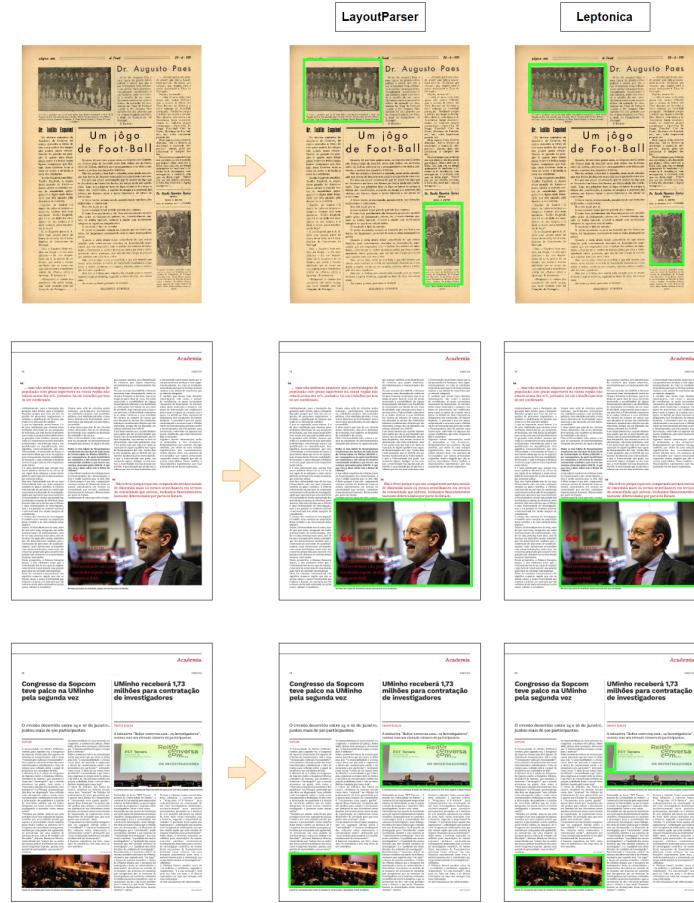


Figura 40: Exemplo de identificação de ilustrações em jornais com diferentes características.

### **identify\_document\_images**

**Descrição:** Método que invoca algoritmo de leptonica para segmentação de página, interpreta o output deste, e retorna uma lista de objetos do tipo Box que representa a posição das imagens identificadas no documento.

#### **Argumentos:**

- image : imagem a binarizar
- tmp\_dir : diretório para armazenar ficheiros temporários

### 5.3.6 Segmentação de documento

Como discutido na secção de métodos aplicados sob os resultados de OCR, a segmentação de um documento tem diversas aplicações. No entanto, como também foi mencionado, a segmentação utilizando os resultados de OCR é muito dependente da qualidade destes, sendo preferível procurar realizar esta diretamente através da imagem.

Para este propósito, foram desenvolvidos métodos para identificação de colunas e para segmentação de documento (header, body, footer).

#### **divide\_columns**

**Descrição:** Método que, através de análise da distribuição de pixels pretos numa imagem de um documento, divide este em colunas. Retorna uma lista de objetos do tipo Box. É recomendado remover ilustrações/imagens do documento pois estes são aglomerados que são outliers na análise de frequências.

#### **Argumentos:**

- image : imagem a binarizar
- method : método de smoothing. Opções : WhittakerSmoothen (por defeito).

#### **Algoritmo:**

Similar ao utilizado para a deteção de colunas através dos blocos de resultado de OCR, porém a análise de frequências é sobre a distribuição de pixels invés das margens dos blocos.

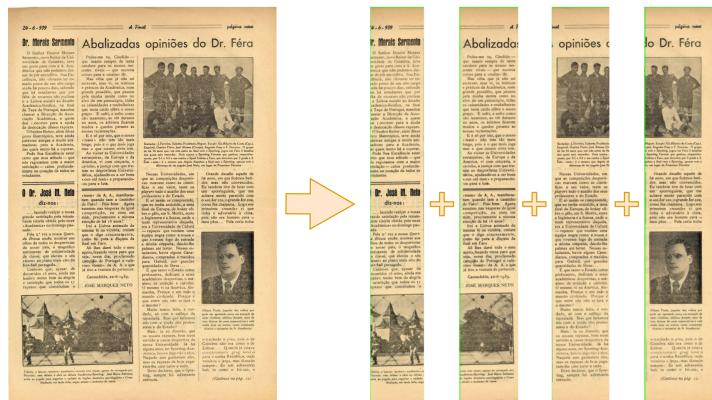


Figura 41: Exemplo de uso de divide\_columns numa página de jornal.

#### **segment\_document**

**Descrição:** Método que, obtendo uma lista de delimitadores detetados no documento, divide procura dividir o documento em header, footer e body.

## Argumentos:

- image : imagem a binarizar
- remove\_images : flag para aplicar método de remoção de imagens
- tmp\_dir : diretoria de ficheiros temporários
- target\_segments : lista segmentos de interesse. Opções possíveis: 'header', 'body','footer'

## Algoritmo:

Utilizando uma lista de delimitadores horizontais, procura ajustar o body, inicialmente considerado como a imagem inteira, em header e footer (se procurados).

O header é assinalado por um delimitador horizontal que segue a seguinte regra:

$$\text{delimiter.bottom} \leq 0.3 * \text{image.shape}[0] \wedge \text{delimiter.width} \geq 0.4 * \text{image.shape}[1]$$

O footer é assinalado por um delimitador horizontal que segue a seguinte regra:

$$\text{delimiter.bottom} \geq 0.7 * \text{image.shape}[0] \wedge \text{delimiter.width} \geq 0.4 * \text{image.shape}[1]$$



Figura 42: Exemplo de uso de `segment_document` em um jornal.

(a) Header encontrado está desenhado a vermelho; o Body foi desenhado a azul.

## 5.4 Processamento de texto

O último conjunto de ferramentas do toolkit dizem respeito à criação de output e, consequentemente, tratamento de texto.

### 5.4.1 Correção de hifenização

Especialmente no caso de documentos multi-colunas, é comum a presença de palavras separadas entre duas linhas através da hifenização. Além disso, é também observável, devido a má formatações de texto ou erros de OCR, a ocorrência de espaços vazios em palavras compostas. Ambos estes cenários podem ser facilmente resolvidos com heurísticos, no toolkit sendo aplicadas pelo método fix\_hifenization.

#### **fix\_hifenization**

**Descrição:** Une palavras separadas entre linhas através de hifenização. Remove espaços vazios extra em palavras compostas.

#### **Argumentos:**

- text : texto a corrigir

**Algoritmo:** Aplicação de substituição através das expressões regulares:

- $'(\backslash w)[\backslash r\backslash t\backslash f\backslash v] * - (\backslash s * \backslash n\backslash s * -*)([a - z0 - 9_])'$  : procura de palavras separadas entre linhas através de hifenização
- $'(\backslash w)[\backslash r\backslash t\backslash f\backslash v] * - [\backslash r\backslash t\backslash f\backslash v] * ([a - z0 - 9_])'$  : procura de palavras compostas com espaços vazios.

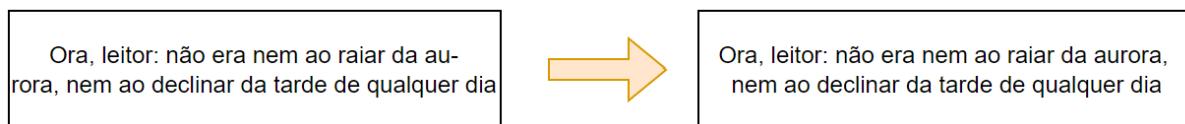


Figura 43: Exemplo de uso de fix\_hifenization em um excerto de texto.

### 5.4.2 Geração de output

A criação de outputs finais de fácil visualização é uma função essencial do trabalho produzido, permitindo uma digestão dos resultados de OCR mais simples.

Fundamentalmente, existem dois métodos principais de geração de output:

- A partir de uma OCR Tree, chamar o método da classe "to\_text" para transformar a árvore numa única string de texto. A ordem do texto seguirá a ordem dos filhos de cada nodo. Delimitadores personalizados para cada um dos níveis podem ser distribuídos através do fornecimento de um dicionário.

- A partir da classe Article, invocar o método "to\_text" para transformar a árvore interna processada numa string simples do título seguido do corpo do artigo.

Invocar o método "to\_md" para gerar uma string markdown que utiliza uma sintaxe mais apropriada para markdown, adicionando também separador entre o título e o corpo. Neste caso, se o artigo tiver elementos de imagem, é possível que estes sejam incluídos no output final.

## **Capítulo 6**

# **OSDOCR Pipeline - Implementação**

Neste capítulo, será explorada a implementação da componente de pipeline.

De forma a criar um caso de uso para as ferramentas criadas e outras exploradas e disponibilizadas, foi criado um comando de aplicação de uma pipeline de aplicação de OCR, que permite, a partir de uma imagem: aplicar tratamentos para melhorar a sua qualidade; aplicar OCR; tratar os resultados; e gerar um output simples, ou específico para jornais.

Através dos argumentos passados à ferramenta, a pipeline consegue comportar-se de forma diferente, por exemplo: aceitar como output inicial uma instância de Hocr ou de OCR Tree em formato JSON; ignorar pré-processamento ou passos específicos deste; etc.

As funcionalidades disponíveis na pipeline são uma culminação do trabalho descrito anteriormente sobre o OSDOCR Toolkit, assim como ferramentas exploradas para resolver questões que este não aborda como, por exemplo, realização de upscaling de uma imagem.

Como discutido no estudo da solução, os blocos que compõem a pipeline são, sempre que possível, opcionais de modo a aumentar a sua versatilidade.

Além disso, dependendo do modo de utilização da pipeline, output extra pode ser requisitado. Utilizando a flag de debug ('-d' ou '--debug') cada um dos blocos da pipeline irá guardar o seu resultado na forma de um ficheiro situado na diretoria de resultados do input respetivo.

## **6.1 Configuração**

O comando de invocação da pipeline é 'osdocr'. Este apresenta as seguintes opções de utilização:

- **target**
  - **alternativo** : t
  - **descrição** : ficheiro de imagem a ser processado pela pipeline.
- **file**

- **alternativo** : f
  - **Descrição** : ficheiro de texto que lista um target por linha.
- **target\_ocr\_results**
  - **alternativo** : tocr
  - **Descrição** : ficheiro de resultados OCR. Pode ser de tipo hocr ou json. Se fornecido, não será realizado pré processamento de imagem ou OCR.
- **output\_folder**
  - **alternativo** : of
  - **Descrição** : caminho para guardar os resultados
- **segmented\_ocr**
  - **alternativo** : sgocr
  - **Descrição** : flag para aplicação de OCR no target será realizada em cada um dos segmentos, invés de na imagem inteira. Cada uma das partes é posteriormente unida para criar uma única OCR Tree de resultados.
  - **valor default** : False
- **target\_segments**
  - **alternativo** : ts
  - **Descrição** : segmentos a calcular no target. Segmento body é sempre obtido. O body é ainda repartido nas suas colunas.
  - **opções** : 'header', 'body', 'footer'
  - **valor default** : 'header','body'
- **output\_segments**
  - **alternativo** : os
  - **Descrição** : indica os output a criar, de acordo com os segmentos fornecidos. Cria um output para cada segmento, se a área para este calculado tiver resultados (nodos da OCR Tree).
  - **opções** : 'all','header', 'body', 'footer'
  - **valor default** : 'all'
- **force\_ocr**
  - **alternativo** : focr
  - **Descrição** : flag que ignora possíveis ficheiros em cache de resultados OCR, consequentes de iterações anteriores do target. Força aplicação de na imagem.
  - **valor default** : False
- **tesseract\_config**

- **descrição** : flags para serem passadas ao Tesseract no momento de aplicação de OCR.  
Estas estão disponíveis na documentação do Tesseract. Cada argumento tem de ter como prefixo ‘\_\_’ para permitir o seu processamento.
  - **valor default** : \_\_l por
- **text\_confidence**
  - **alternativo** : tc
  - **descrição** : valor de confiança de texto que pipeline irá usar.
  - **valor default** : 10
- **split\_whitespace**
  - **alternativo** : sw
  - **descrição** : valor utilizado como razão entre um espaço branco num bloco de texto e a média pesada dos espaçamentos entre palavras, para este ser considerado válido como ponto de divisão de um bloco.
  - **valor default** : 3
- **fix\_rotation**
  - **alternativo** : fr
  - **descrição** : opções usadas para a correção de rotação de documento.
  - **opções** : ‘auto’, ‘clockwise’, ‘counter-clockwise’
  - **valor default** : ‘auto’
- **upscale\_image**
  - **alternativo** : upi
  - **descrição** : opções usadas para o upscaleing de imagem. Se opção ‘autoscale’ do módulo ‘waifu2x’ for escolhido, aplica upscaleing da imagem até esta chegar ao dpi alvo (opção target\_dpi).
  - **opções** : ‘waifu2x’ -> ‘scale2x’, ‘scale4x’, ‘autoscale’
  - **valor default** : ‘waifu2x’
- **target\_dpi**
  - **alternativo** : tdpi
  - **descrição** : valor do dpi alvo
  - **valor default** : 300
- **target\_dimensions**
  - **alternativo** : tdim
  - **descrição** : valor das dimensões físicas a usar para cálculo do dpi da imagem. Opções estão disponíveis no ficheiro ‘dimensions.json’ no projeto, podendo ser atualizado.
  - **opções** : ‘A5’, ‘A4’, ‘A3’, ‘A2’, ‘A1’, ‘A0’, ‘2A0’
  - **valor default** : A3

- **denoise\_image**
  - **alternativo** : di
  - **Descrição** : opções usadas para o denoising de imagem.
  - **opções** : 'waifu2x' -> '-1' - '3'
  - **valor default** : 'waifu2x'
- **light\_correction**
  - **alternativo** : lc
  - **Descrição** : opções usadas para a correção de iluminação de uma imagem.
  - **opções** : 'best\_SSIM', 'best\_PSNR', 'LOL-Blur', 'SICE', 'SID', 'w\_perc'
  - **valor default** : 'best\_SSIM'
- **light\_correction\_split\_image**
  - **alternativo** : lcs
  - **Descrição** : flag para aplicação de correção de iluminação da imagem em patches, invés de na totalidade, de modo a melhorar tempo de processamento. Para certos modelos, resulta em contrastes consideráveis entre os patches.
  - **valor default** : True
- **binarize\_image**
  - **alternativo** : bi
  - **Descrição** : opções usadas para a binarização de imagem para aplicação de OCR.
  - **opções** : 'fax', 'otsu'
  - **valor default** : 'fax'
- **remove\_document\_images**
  - **alternativo** : bi
  - **Descrição** : método utilizado para remoção de imagens.
  - **opções** : 'leptonica', 'layoutparser'
  - **valor default** : 'leptonica'
- **target\_old\_document**
  - **alternativo** : tod
  - **Descrição** : flag para indicar que target é um documento antigo. Utilizado quando método 'layoutparser' é escolhido, de forma a escolher o modelo mais apropriado
  - **valor default** : True
- **ignore\_delimiters**
  - **alternativo** : igd
  - **Descrição** : flag para ignorar delimitadores. Se ativada, estes não serão tidos em conta como indicadores do layout do documento no cálculo da ordem de leitura.
  - **valor default** : False

- **`title_priority`**
  - **alternativo** : tp
  - **descrição** : flag dar prioridade a títulos na escolha de blocos (quando não há âncora) durante cálculo de ordem de leitura.
  - **valor default** : False
- **`skip_method`**
  - **descrição** : métodos/passos da pipeline a ignorar.
  - **opções** : 'leptonica', 'layoutparser'
  - **valor default** : 'all', 'auto\_rotate', 'noise\_removal', 'blur\_removal', 'light\_correction', 'image\_preprocess', 'remove\_document\_margins', 'remove\_document\_images', 'image\_upscaling', 'identify\_document\_delimiter', 'binarize\_image', 'clean\_ocr', 'split\_whitespace', 'unite\_blocks', 'calculate\_reading\_order', 'extract\_articles', 'posprocessing'
- **`calibrate`**
  - **descrição** : aplicar modo de calibração de pipeline. Usado para encontrar melhor configuração de pipeline para um dado target, ou testar uma configuração. Espera ser fornecido um diretório com configurações da pipeline, ou um caminho para a configuração.
- **`calibrate_no_reuse`**
  - **descrição** : flag para não utilizar cache existente nos resultados de calibração.
  - **valor default** : False
- **`pipeline_config`**
  - **descrição** : ficheiro do tipo JSON com configuração de pipeline a usar. Pode ser usado como alternativa a aplicar argumentos no terminal de comandos.

## 6.2 Pré-processamento de imagem

Como primeiro procedimento da pipeline, no caso de uso de inputs de imagem, tem-se o tratamento desta, para procurar obter, a partir da aplicação OCR, uma melhor transcrição do conteúdo textual, assim como melhor identificação de outros elementos, como figuras.

Para isso, procurou-se abordar os seguintes problemas: imagens rodadas; remoção de margens/- sombras na margem de documentos; imagens de baixa resolução; remoção de figuras de documentos; imagens com ruído; imagens com distribuição de iluminação inconsistente;.

As solução destes problemas envolveram o uso de métodos desenvolvidos no toolkit, assim como soluções já existentes.

A ordem de aplicação destas soluções mostrou ser relevante pois estas podem interferir umas com as outras, por exemplo: aplicação de denoising antes de remover as figuras do documento pode afetar a identificação destas. Assim sendo, a ordem ótima de execução segue a listagem seguinte.

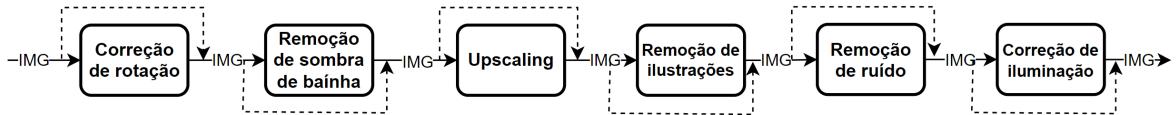


Figura 44: Pipeline - secção pré-processamento

## Correção de rotação

A correção de possíveis rotações de imagens de documentos é resolvida utilizando os métodos de correção de rotação descritos na secção de métodos de processamento de imagem.

## Remoção de sombras das margens do documento

De igual modo, a remoção de possíveis sombras na margem de um documento aproveita os métodos desenvolvidos e descritos para este efeito na mesma secção.

A aplicação desta funcionalidade deve vir após a correção de possíveis rotações pois, como esta solução envolve a análise da distribuição dos valores de pixels, se estas sombras se apresentarem inclinadas a possibilidade que elas não sejam resolvidas aumenta.

## Upscaling de imagem

Para a resolução de problemas resultantes de imagens com baixos valores de dpi, modelos de Deep Learning obtém resultados mais interessantes em relação a algoritmos passados, como interpolação. Dentro destes modelos, o [waifu2x](#) é bastante respeitado no que toca a aumento de resolução de imagem e limpeza de imperfeições.

Embora não especificamente focado para o processamento de imagens de documentos, o seu fácil uso e instalação, assim como a verificação de bons resultados mesmo nesta área relativamente externa ao treino, levaram a que este modelo seja uma boa adição às ferramentas de pré-processamento de imagem disponíveis na pipeline.

Esta funcionalidade, quando escolhida para realizar upscaling automático, é dependente das configurações utilizadas para cálculo de dpi de imagem (dimensões reais proporcionadas).

A aplicação de OCR é recomendada para dpi no valor de 300 ou superior, sendo que para imagens antigas, esta funcionalidade é essencial.

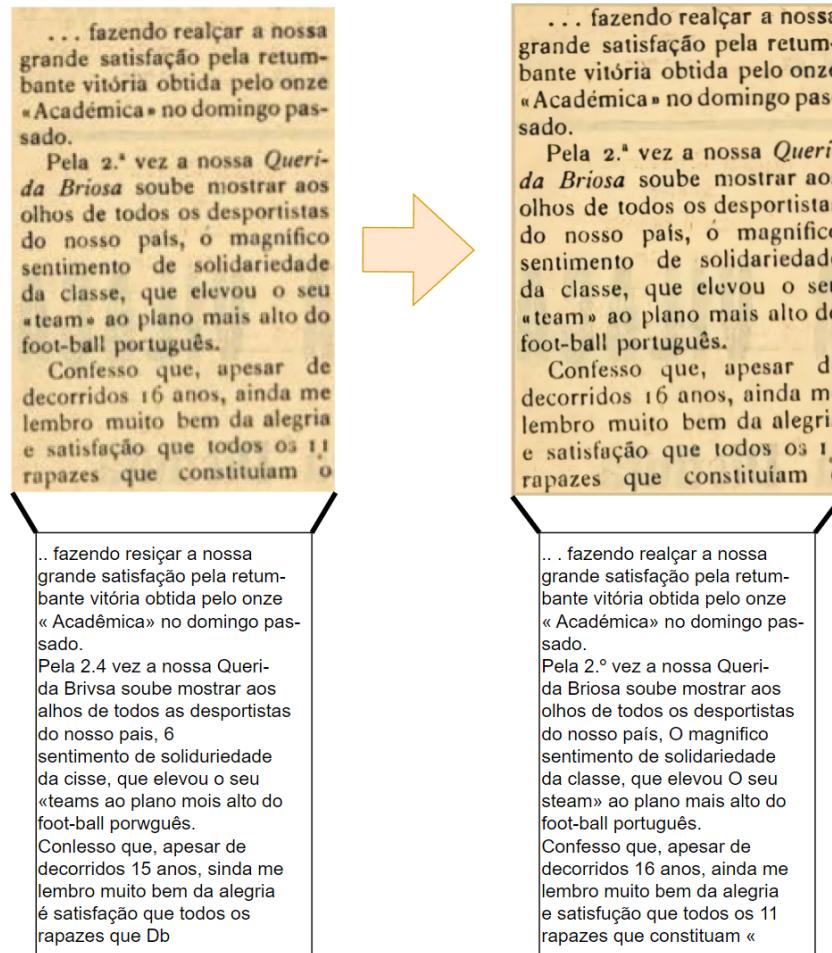


Figura 45: Exemplo de diferença de OCR numa secção de texto.

(a) Sem (esquerda) e com (direita) upscaling. Upscaling realizado foi apenas de 2x, utilizando o modelo waifu2x.

## Remoção de ilustrações de documento

De forma a diminuir a presença de elementos não identificados nos resultados OCR, assim como permitir identifica-los corretamente, esta funcionalidade procura remover as ilustrações de um documento, recortando-as para uma pasta temporário, permitindo a sua reposição na imagem final.

Entre as soluções exploradas para esta questão, a que se enquadrava mais na questão de identificação de ilustrações de documentos, e até com atenção a jornais, foi o modelo [Layout Parser](#), mais especificamente, aproveitando o modelo Detectron desenvolvido pela Meta. Este, no entanto, apresenta uma instalação complexa devido a dependências de bibliotecas da própria Meta que demonstram incom-

patibilidades e que necessitam ser modificadas manualmente.

Deste modo, aproveitou-se uma alternativa com resultados também satisfatórios e de menor custo computacional, com métodos de segmentação de documento propostos pela Leptonica. Estes foram descritos também na secção anterior.

Comparando os dois, a opção de usar Leptonica é satisfatória na generalidade dos casos, até identificando com maior precisão as ilustrações (delimitando menos fundo do documento) em vários casos de estudo. Os métodos de Leptonica tendem no entanto a errar para ilustrações que não apresentem uma borda notável e tenham um fundo similar ao fundo do documento. Nestes casos, o Layout Parser é mais propício a acertar.

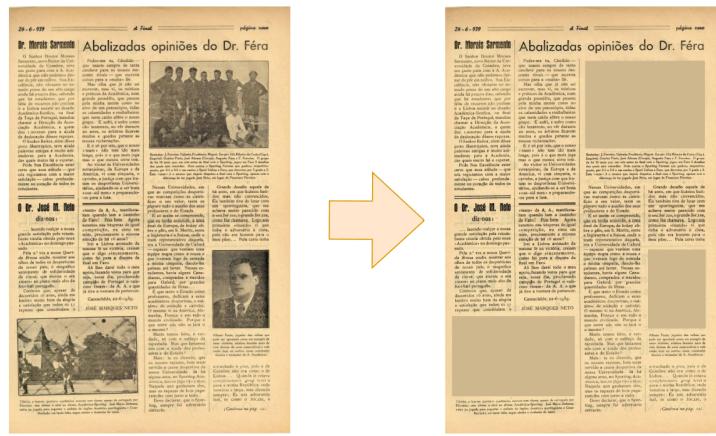


Figura 46: Exemplo de resultado de passo de remoção das imagens identificadas.

É também importante realçar que a pipeline, no processo de recortar as ilustrações, preenche o espaço vazio com uma média das cores de fundo da imagem. Este passo é importante pois caso contrário a binarização posterior da imagem, dependendo da distribuição das cores desta, pode resultar numa imagem inutilizável.

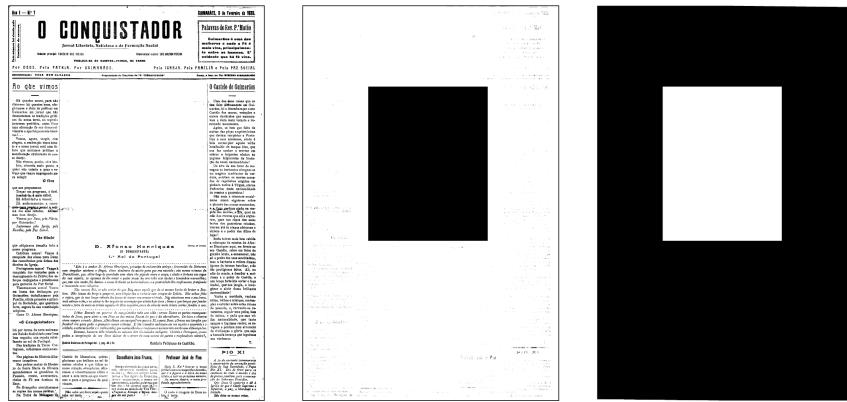


Figura 47: Resultado de binarização correspondente a diferentes preenchimentos da zona das ilustrações removidas.

(a) Esquerda: preenchido com a cor média da imagem; Meio: preenchido com branco; Direita: preenchido com preto.

## Denoising de imagem

Para a realização de denoising, foi também aplicado o modelo de **waifu2x**, visto este proporcionar opções para este efeito. Este denoising é realizado em imagens a cor, sendo portanto subtil.

O denoising mais relevante é realizado na binarização realizada antes da aplicação de OCR.

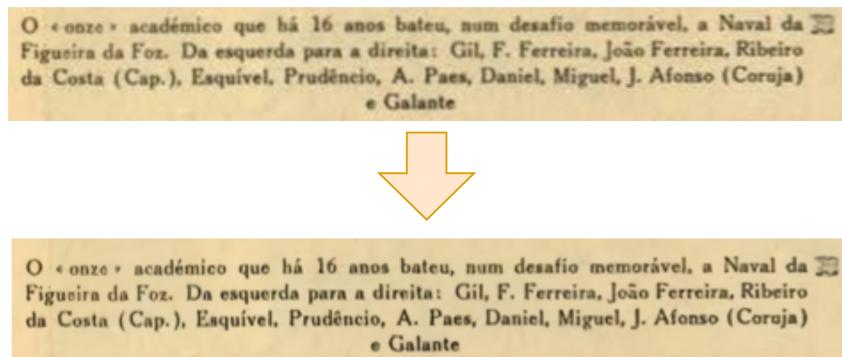


Figura 48: Resultado de modelo de denoise do waifu2x.

## Correção de iluminação

Para a correção de iluminação de documentos, também se conclui que modelos de Deep Learning são a melhor opção. A família de modelos que mostraram resultados mais interessantes e com instalação simples foi [HVI-CIDNet](#). Os pesos para estes estão disponibilizados nesse mesmo repositório.

Para imagens de maior resolução, o tempo de processamento destes modelos é considerável, sendo

que como solução a pipeline apresenta uma opção para dividir a imagem em patches e correr o modelo em cada um destes, unindo-os no final. Para alguns modelos, esta divisão pode resultar em contrastes notáveis na imagem, particularmente para imagens de maior resolução (por ser repartido em mais patches).



Figura 49: Resultado de uso de diferentes modelos de correção de iluminação.

(a) Do lado esquerdo de cada modelo tem o resultado de uso sem divisão da imagem, e do lado direito o uso com divisão. Esta é uma imagem de resolução 1103x1612, já demonstrando para alguns modelos irregularidades no patch direito.

### 6.3 OCR

Seguindo na pipeline, temos a aplicação de OCR para extração do conteúdo textual de uma imagem. No caso de input do tipo OCR Tree - ou seu convertível -, este passo é ignorado, caso contrário é obrigatório.

Ao longo da implementação e estudo dos resultados desta, foi utilizado **Tesseract** para a realização de OCR, devido a ser uma ferramenta open-source no topo do estado da arte.

O processo de OCR da pipeline pode ser dividido em blocos menores, sendo estes:

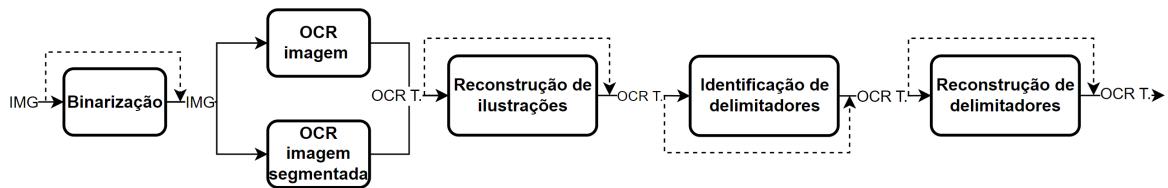


Figura 50: Pipeline - secção OCR

## Binarização

A binarização de imagem é um processo que permite a realização de redução de ruido através da aplicação de thresholds, assim como a acentuação do texto de documentos. Este passo é também recomendado na documentação do Tesseract..

Este passo pode ser ignorado na pipeline, até pois o Tesseract internamente irá aplicar uma binarização própria.

A pipeline apresenta a opção de realizar binarização utilizando **Otsu** ou estilo **Fax**.

## OCR em imagem

A aplicação de OCR é realizada utilizando **Tesseract** podendo, através dos argumentos da pipeline, modificar a configuração deste, por exemplo: linguagem do texto a detetar; dpi da imagem; modo de segmentação.

Os resultados de OCR serão transformados numa OCR Tree.

## OCR em imagem segmentada

Em muitos casos, os documentos antigos apresentam variações do seu estado ao longo do documento. Por este motivo, a pipeline permite a habilidade de realizar um reconhecimento de texto aos diferentes segmentos da imagem, invés de numa única passagem.

Para isto, podem ser passados os segmentos esperados na consola: 'header' , 'body' e 'footer'. Utilizando métodos de processamento de imagem do Toolkit, o documento será dividido nestes segmentos, particularmente o body será possivelmente ainda dividido em colunas, que serão separadamente binarizados (se intedido) e analisados com OCR.

Em seguida as diferentes OCR Tree resultantes serão unidas numa única.

## Reconstrução de delimitadores e imagens nos resultados

Com a OCR Tree obtida, o último passo desta secção passa pela reconstrução de elementos não textuais do documento na OCR Tree.

Estes são: ilustrações reconhecidas durante o pré-processamento; delimitadores identificados utilizando o Toolkit (se intedido).

Os elementos serão adicionados na OCR Tree, removendo blocos vazios que se situem dentro ou a intersetar com estes novos elementos.

## 6.4 Pós-processamento de OCR

Esta secção trata maioritariamente de manipulação da estrutura OCR Tree. A manipulação desta tem, naturalmente, consequências no texto reconhecido, e.x.: processos de eliminação de blocos vazios, estes são reconhecidos através do threshold de confiança definido, sendo que na realidade poderão conter texto.

Esta secção é também o segundo ponto de entrada da pipeline, sendo possível aceder diretamente a este através de um input do tipo Json ou HOCR, convertível para OCR Tree.

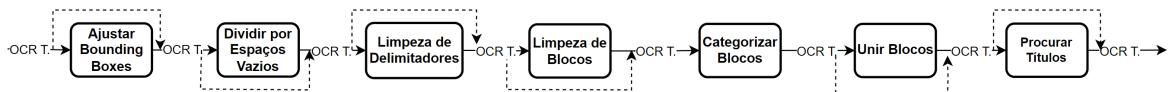


Figura 51: Pipeline - secção pós-processamento

### Remover blocos vazios

Assumindo um valor de confiança de texto para ser vir de threshold, são removidos os blocos sem texto dos resultados. Estes são usualmente ruído que foi erradamente detetado como texto.

São exceções imagens e delimitadores, que no caso do Tesseract, são detetados como blocos vazios. Dependendo da configuração da pipeline estes são detetados de diferentes formas, como já foi descrito.

### Ajustar Bounding Boxes

O primeiro bloco de ação tem como objetivo reduzir a dimensão das bounding boxes dos blocos. Isto consegue-se através do valor dado para threshold de confiança de texto. Fazendo uso deste e do método 'text\_bound\_box\_fix', faz-se uma análise dos nodos de texto dentro dos blocos e, quando ignorados os de confiança abaixo da mínima, é possível reduzir a dimensão do bloco.

Esta ação é importante para diminuir possíveis intersecções entre os blocos, normalmente causadas

por deteções de ruído como texto. A redução destas interseções auxilia outras ações, como união de blocos ou cálculo da ordem de leitura.



Figura 52: Resultado de passo de ajuste de bounding boxes utilizando diferentes valores de *conf*.

## Dividir por espaços vazios

A ação seguinte tem o intuito de dividir texto separado por um considerável espaço vazio em mais do que um bloco. Tal tendência a ocorrer na área de menção dos autores de artigos ou ilustrações.

Este bloco é conseguido usando o método 'split\_whitespaces'. Os parâmetros de confiança de texto e a razão entre o valor normal de espaço vazio e um válido de separação, são as principais influências no desempenho deste passo.

## Limpeza de delimitadores

Tanto no caso de delimitadores detetados através de métodos de análise de imagem, ou por heurísticas de dimensão dos blocos vazios, é expectável a presença de ruído, e.x.: delimitadores que foram detetados através do texto de título do jornal por este ser muito acentuado.

Deste modo, a limpeza destes é um passo recomendado. Esta consegue-se através de um método criado para este propósito da pipeline: 'delimiters\_fix'. Este método analisa os delimitadores e, validadas certas condições, aplica limpezas, ex.: delimitador inserido completamente dentro de texto -> remove delimitador; delimitador a intersetar com texto -> ajusta bounding box do delimitador.

## Limpeza de blocos

Segue-se o passo de limpeza geral de blocos, tratando casos de interseções, blocos vazios e blocos inseridos em outros. Para isto é usado o método 'block\_bound\_box\_fix'.

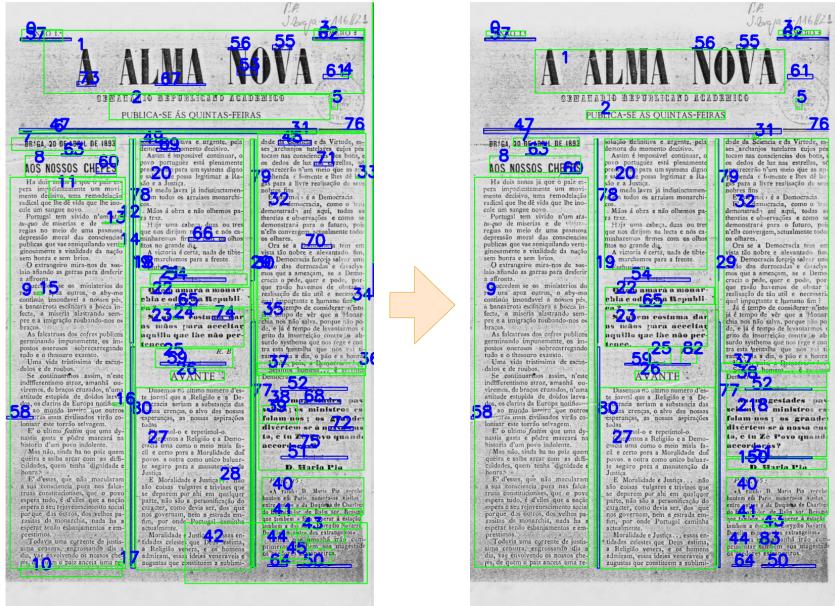


Figura 53: Resultado de aplicação dos passos de limpeza (passos até agora listados) dos resultados OCR.

## Categorização de blocos

O passo de categorização de blocos é essencial para as ações posteriores de união de blocos, procura por títulos e cálculos de atração entre blocos. Não sendo uma ação impactante da OCR Tree, apenas atualizando o atributo 'type'. Este é um passo obrigatório na pipeline.



Figura 54: Resultado de aplicação do passo de categorização de blocos.

## União de blocos

A união de blocos auxilia o passo de cálculo da ordem de leitura, e realiza uma simplificação da OCR Tree. O objetivo é a redução da quantidade de blocos existente ao unir aqueles que são do mesmo tipo e aceitam determinadas condições geométricas.

A união de blocos pode realçar posição relativa entre estes e outros que anteriormente não eram expressas, potencializando o cálculo da atração entre blocos.

## Procura por títulos

O bloco de procura por títulos pretende corrigir casos em que a segmentação em blocos do motor OCR uniu texto com potencial de título (de artigo sendo esse o foco do projeto) do texto seguinte.

Tal permitirá em passos seguintes, de isolamento dos artigos, a identificação de mais artigos do que na ausência deste passo, dado que os artigos são distinguidos pela deteção de títulos.

No caso de serem detetados novos títulos, o método 'find\_text\_titles' irá dividir o bloco de texto, gerando o novo bloco título e 1 a 2 (dependendo da posição relativa do título) blocos texto.

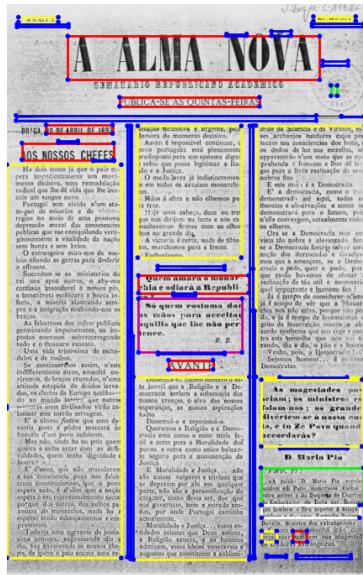


Figura 55: Resultado final de pós processamento dos resultados de OCR.

## 6.5 Geração de output

A última secção da pipeline passa pela geração do output final em formato textual.

Nesta, a pipeline da opção de escolher: a geração dedicada a jornais, onde a flag 'extract\_articles' está ativa e portanto os artigos serão isolados para o output final; ou output simples. Em ambos os casos é ainda possível aplicar o bloco de cálculo da ordem de leitura e escolher o formato do output textual.

Os formatos para output final disponíveis são: texto simples; markdown.

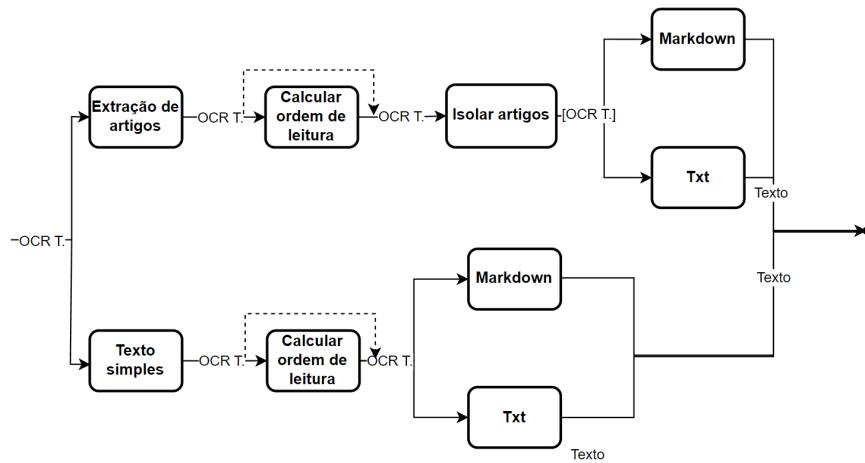


Figura 56: Pipeline - secção output

## Cálculo da ordem de leitura

O bloco de cálculo da ordem de leitura pretende, através do cálculo de atração entre blocos - que dependem das características dos blocos assim como posições relativas - do documento, e de uma ordenação topológica pesada, gerar uma ordenação mais próxima da intendida do que originalmente decidida pelo motor OCR. Como explicitado anteriormente, os algoritmos desenhados têm um foco na ordenação de jornais.

Pode ser utilizada a opção 'title\_priority' para escolher a utilização de um filtro por títulos no momento de escolha do próximo bloco da ordenação (quando não existe uma âncora prévia).



Figura 57: Resultado de cálculo da ordem de leitura.

## Isolamento de artigos

O passo de isolamento de artigos procura, dentro da lista de blocos ordenados, segmentá-los em diferentes artigos. Os artigos são caracterizados por serem iniciados por um título e possuírem texto além deste.

O resultado final será uma lista de listas OCR Tree do nível bloco, i.e. uma lista de artigos.



Figura 58: Resultado de isolamento de artigos.

## Output final

O output final será sempre em formato textual. Este pode ser do tipo markdown ou texto simples, e

do tipo documento jornal ou outro.

No caso de documento jornal, o algoritmo irá utilizar os artigos isolados e, por ordem, transcrevê-los no output. Para outro tipo, apenas transcreve os blocos da forma como foram ordenados.

Em particular para o formato markdown, tendo em conta a possibilidade de incluir imagens neste formato, blocos do tipo imagem identificados durante a pipeline, serão incluídos no output final na posição indicada pela ordem final de blocos/artigos.

Opções de segmentação podem ser dadas na pipeline de forma a ser gerada mais do que um output, nomeadamente fornecendo os segmentos desejados de output, como 'header' e 'body' os quais serão calculadas as áreas e, se válidas, serão processados para conceberem apenas os blocos que as incluem.

## 6.6 Validação de resultados

O comando que aciona a pipeline apresenta ainda uma aplicação alternativa. Esta outra aplicação é dedicada à verificação/validation dos resultados da utilização da pipeline e, consequentemente, calibrar a mesma.

Esta utilização, ao contrário da pipeline, não é de independente do utilizador, necessitando que este provisione recursos extra além da imagem de input inicial. Estes recursos extra são respetivos a resultados esperados do output: ground truth total; ground truth parcial; ficheiro de resultados esperadas. Destes, pelo menos uma das ground truth deve ser fornecida. Além disso, pelo menos uma pipeline tem de ser fornecida para serem analisados os seus resultados no input.

O caso de uso esperado é: dada para uma coleção de um dado jornal, em que as diferentes edições têm características semelhantes, o utilizador forneça ,para uma destas, estes recursos para comparação, assim como uma lista comprehensiva de diferentes pipelines com diferentes configurações que serão comparadas e, entre estas, serão escolhidas as opções que melhores resultados obtiveram. Será resultado da execução um ficheiro de configuração de pipeline que englobe as opções que melhor resultado obtiveram.

Além disso, este módulo permite uma apreciação objetiva do uso da pipeline, sendo que será utilizado para a obtenção de resultados desta mesma componente da solução global.

Os métodos utilizados nesta funcionalidade são, na generalidade, dedicados à mesma, não tendo sido implementados com a mesma filosofia de independência do toolkit.

### Ground truth total

A ground truth total trata-se de uma transcrição completa do texto do documento. Utilizada para uma

comparação direta do output textual final, no formato simples, da pipeline com o real.

Deste é obtido os dados:

- número total de palavras
- conjunto de palavras únicas
- número de aparições de cada palavra única

## **Ground truth parcial**

A ground truth parcial trata-se de uma transcrição de apenas alguns trechos do documento, que devem estar ordenados corretamente, ex.: uma frase de cada uma dos artigos do jornal. Tal permite simultaneamente verificar a precisão de acerto de palavras do OCR, assim como do algoritmo de ordenação utilizado.

Deste é obtido os dados:

- número de linhas
- ordem das linhas

## **Ficheiro de resultados esperados**

O ficheiro de resultados contém informação além do output textual final. Este pode conter: o número de artigos que o output final deve ter; número de ilustrações que o documento contém; número de colunas do jornal.

Deste é obtido os dados:

- número de colunas esperadas
- número de imagens esperadas
- número de artigos esperados

## **Comparação dos resultados e calibração da pipeline**

Dada um conjunto de pipelines para serem comparadas, para cada uma seguirá o seguinte processo:

**Aplicação da pipeline no input**, guardando os seus outputs numa pasta própria.

**Apuramento dos resultados** da pipeline, nomeadamente:

- nível médio da confiança do texto
- número de palavras

- número de blocos com texto
- similaridade de cosseno entre o texto e a ground truth total
- número de palavras únicas detetadas
- rácio de número de aparição de cada palavra relativamente à ground truth
- número e rácio de linhas semelhantes<sup>1</sup> entre ground truth parcial e o texto
- número e rácio de linhas iguais entre ground truth parcial e o texto na ordem certa
- rácio geral de ordem da ground truth parcial
- número de colunas detetadas
- número de ilustrações detetadas
- número de artigos detetados

**Atribuição de uma classificação** numérica a cada secção da pipeline - pré e pós processamento

- ao comparar os resultados obtidos com os recursos dos resultados esperados.

**Calibração da pipeline** ao comparar os diferentes classificações das pipelines e, escolhendo para cada secção as configurações da pipeline (respetivas à dada secção) que nela teve melhor classificação.

É também possível apenas realizar a classificação de uma pipeline única.

---

<sup>1</sup> Utilizado similaridade de Jaro-Winkler com threshold de 90%

## **Capítulo 7**

# **OSDOCR Editor - Implementação**

Neste capítulo será analisada a implementação do último módulo da solução OSDOCR, um editor de OCR Tree gráfico. Este vem com o propósito de servir como uma outra aplicação do Toolkit criado, maioritariamente no que toca às ferramentas que rondam a estrutura OCR Tree, assim como um complemento para a OSDOCR Pipeline que, em casos mais complexos, ou para outros tipos de documentos que não foram o foco da solução, requeira uma manipulação dos resultados mais manual. O editor gráfico facilita esta manipulação.

Embora todos os níveis da OCR Tree possam ser transformados com o editor, o objetivo principal é a de trabalhar com os de nível 2, sendo este nível o que será desenhado na tela de edição.

Esta interface gráfica foi criada utilizando para a parte visual e controlo das Views as bibliotecas **PySimpleGui** e **Matplotlib**; e para módulo de dados e controlador destes o Toolkit, com a opção de utilizar a Pipeline para a realização de OCR localmente no input.

### **7.1 Sumário**

Segue-se uma listagem das funcionalidades que serão realçadas:

- Inputs
- Manipulação manual de OCR Tree
- Aplicação local de OCR
- Ferramentas disponíveis
  - Divisão de blocos
  - Junção de blocos
  - Categorização de blocos
  - Ordenação de blocos
  - Segmentação de blocos em artigos
  - Divisão de imagem
- Outputs
- Operações adicionais

## 7.2 Funcionalidades

### 7.2.1 Inputs

Naturalmente, a primeira funcionalidade a ser discutida são os tipos de ficheiros de entrada necessários e admitidos pelo editor.

Sendo este um editor de OCR Tree, poderia se esperar que o único input necessário seria um ficheiro compatível com OCR Tree, no entanto, a utilidade da ferramenta seria consideravelmente limitada se o utilizador não tivesse a imagem par da OCR Tree para servir como fundo. Tomou-se então a decisão que são necessários 2 inputs para iniciar a tela de edição do GUI, uma imagem de input e um ficheiro convertível para OCR Tree - seja este um Json ou HOCR.

Realça-se que a imagem e a OCR Tree podem não corresponder, sendo responsabilidade do utilizador escolher os inputs corretamente pareados.

A tela resultará na OCR Tree sobreposta na imagem de input. Se estes estiverem propriamente pareados, os blocos desenhados estarão corretamente alinhados com o texto respetivo na imagem.



Figura 59: OCR Editor: escolha de inputs. 1 - Input de imagem. 2 - Input de ficheiro OCR Tree

Numa nota extra sobre os inputs: o editor possui uma opção nas configurações que permite que ao ser escolhida uma imagem de input, seja procurada automaticamente por outputs desta resultantes do uso da pipeline, acelerando o processo de seleção da OCR Tree.

## 7.2.2 Manipulação manual de OCR Tree

Segue-se o propósito principal da criação do editor gráfico, a facilitação da manipulação manual de OCR Tree.

Utilizando o OSDOCR Editor é possível:

**Selecionar blocos** clicando nestes usando o rato. Múltiplos blocos podem ser selecionados - mantendo premido a tecla 'ctrl' ou clicando e arrastando por cima de uma área - e, consequentemente, manipulados em simultâneo. Para desselecionar um bloco, basta clicar neste novamente ou selecionar uma zona sem blocos.

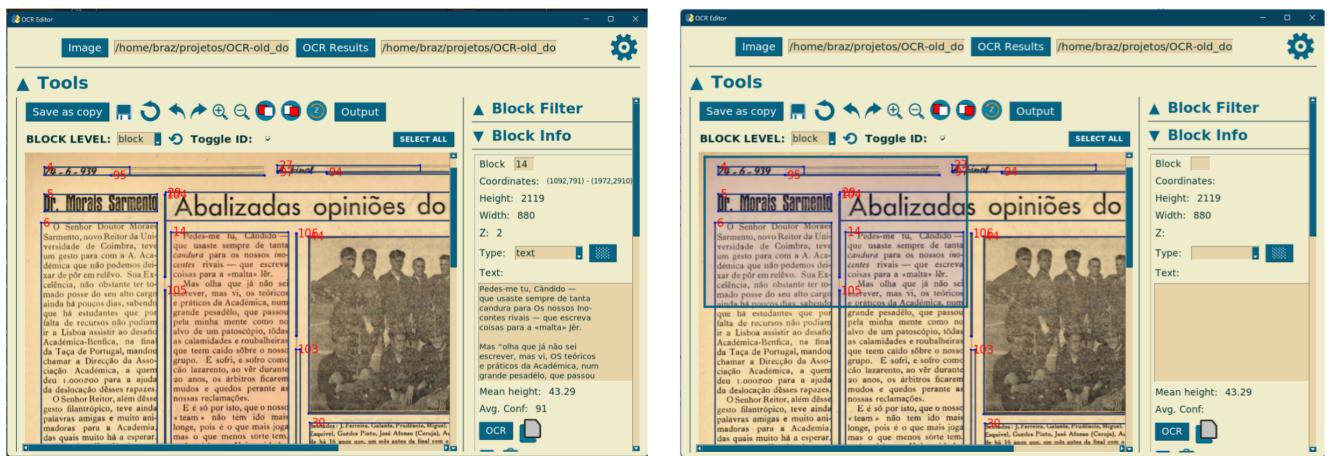


Figura 60: OCR Editor: seleção de blocos

**Mover blocos** selecionando um ou múltiplos blocos e, com o botão do rato pressionado, mover o cursor para o local desejado.

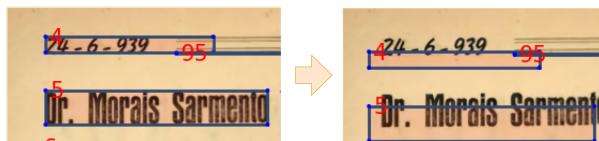


Figura 61: OCR Editor: mover blocos

**Redimensionar blocos** selecionando um ou múltiplos blocos e, clicando no vértice para pivô de redimensionamento, mover conforme desejado.

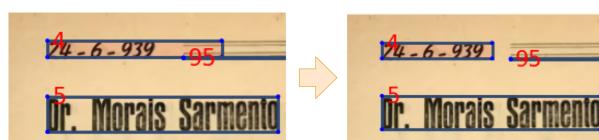


Figura 62: OCR Editor: redimensionar bloco

**Atualizar texto do bloco** manualmente, ao selecionar um bloco e, na aba da direita com a informação do bloco, atualizar o texto de acordo e clicar em salvar. No caso de múltiplos blocos serem selecionados, apenas o último selecionado é modificável.

Realça-se que a edição manual do texto leva a perda de alguma informação, nomeadamente as dimensões dos níveis mais profundos dentro do bloco, visto que requerer ao utilizador que delimitar as coordenadas e nível de confiança de texto das linhas, palavras ou parágrafos editados iria contra o propósito da edição de OCR Tree facilitada. Estes níveis terão as suas dimensões repartidas de forma uniforme de forma a encaixarem dentro do bloco pai e a confiança de texto será considerada como total (100).



Figura 63: OCR Editor: atualizar bloco. 1 - Selecionar bloco. 2 - Modificar dados do bloco. 3 - Guardar alterações.

**Adicionar um novo bloco** clicando no botão do meio do rato no local desejado, ou utilizando o botão com o mesmo efeito.

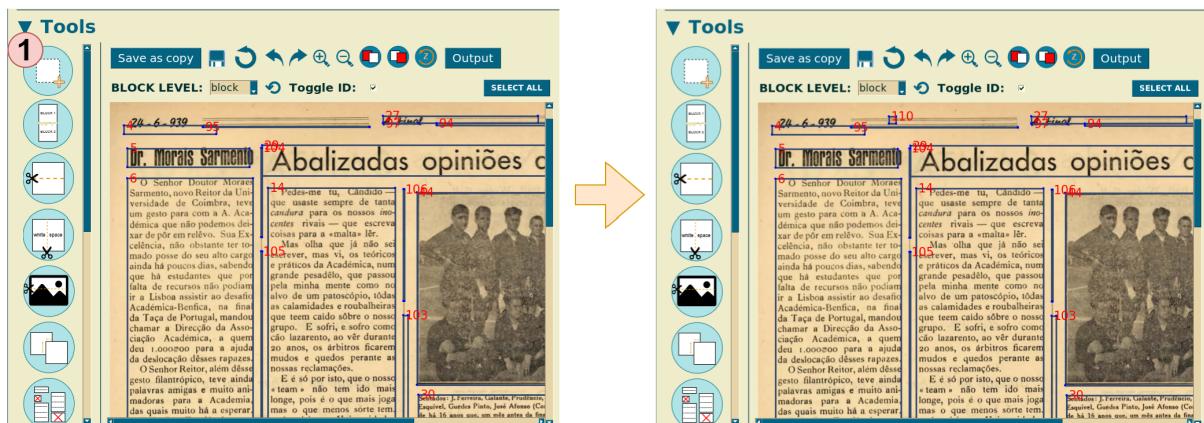


Figura 64: OCR Editor: criar novo bloco. 1 - Ferramenta para criar novo bloco (ID 110).

**Remover um bloco** selecionando o bloco a remover e clicando no botão (ou utilizando a tecla 'delete') para o efeito. No caso de múltiplos blocos selecionados, todos serão removidos.



Figura 65: OCR Editor: eliminar bloco. 1 - Selecionar bloco. 2 - Eliminar bloco.

### 7.2.3 Aplicação local de OCR

Em instâncias em que apenas porções da OCR Tree apresentem resultados não satisfatórios, ou se pretenda criar novos blocos de texto, a habilidade de aplicar OCR no local afetado é mais prática do que transcrever o texto manualmente por completo.

Isto consegue-se aplicando a OSDOCR Pipeline. Nas configurações do editor é possível costumizar os parâmetros da pipeline a utilizar, por exemplo: não aplicar pós processamento e no pré processamento não realizar upscaling.

A utilização da implementação da pipeline implica que é necessário transformar os outputs desta, transformando a OCR Tree resultante numa de nível 2 e, atualizar as suas coordenadas para fazerem sentido à posição do bloco no input do editor, assim como a escala das dimensões caso necessário.

Uma configuração própria desta funcionalidade, é a escolha entre os resultados serem comprimidos em um único bloco, ou serem inseridos novos blocos de acordo com os resultados da pipeline. Deste modo, é também possível realizar OCR numa imagem inteira, ampliando esta ferramenta.

Para utilizar esta funcionalidade própria do editor, basta selecionar o bloco alvo e clicar no botão de OCR na aba lateral direita.

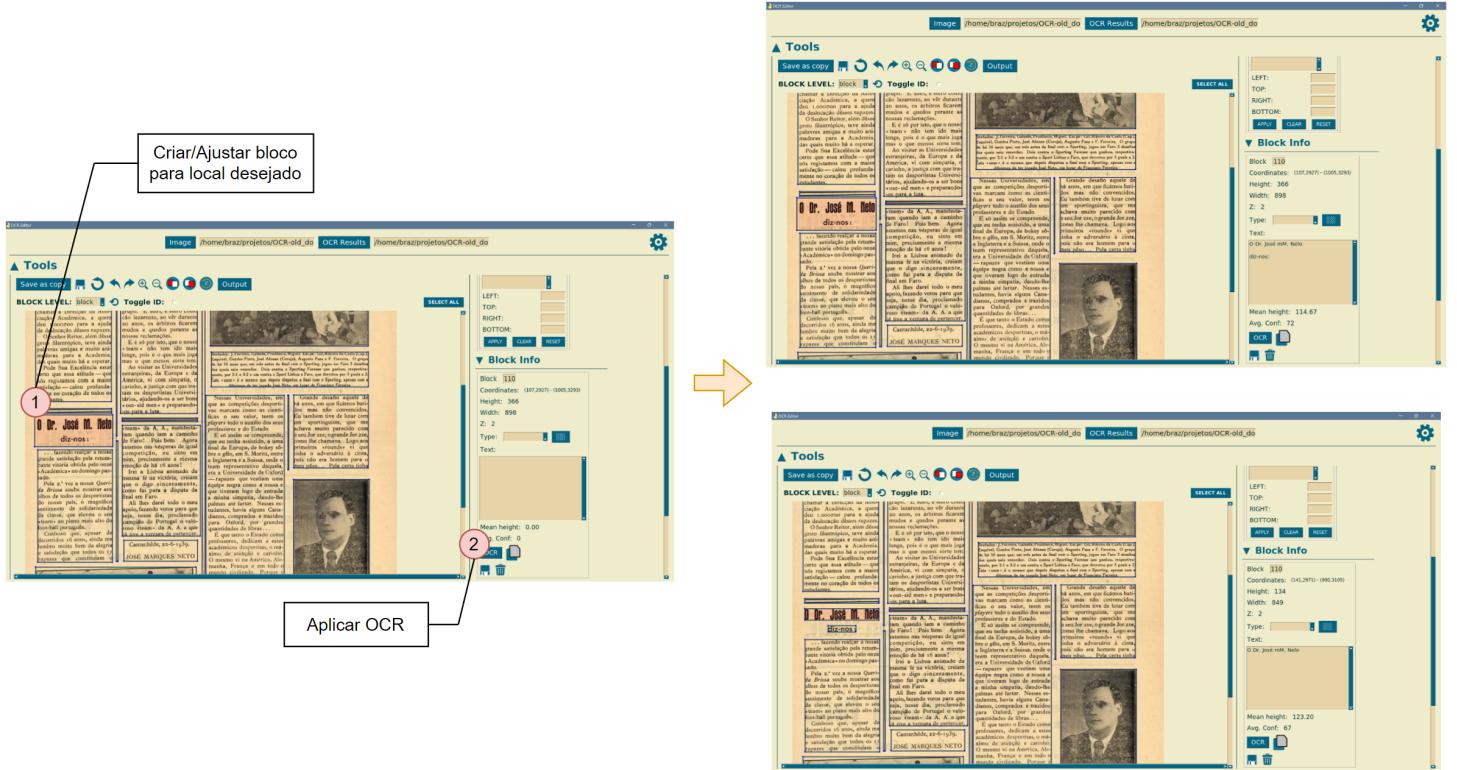


Figura 66: OCR Editor: aplicar OCR. 1 - Selecionar bloco. 2 - Aplicar OCR. Resultado em cima - bloco único. Resultado em baixo - blocos resultantes.

## 7.2.4 Ferramentas disponíveis

Nesta secção serão descritas as funcionalidades conseguidas através do uso do OSDOCR Toolkit. Várias destas já estão presentes no OSDOCR pipeline porém, o editor permite uma utilização mais controlada destas.

### Divisão de blocos

A divisão de blocos tem como principal propósito corrigir a segmentação de blocos realizada pelo motor OCR.

Nesta categoria temos 2 tipos de divisão:

#### Corte de bloco

Esta funcionalidade não está diretamente disponível na pipeline - embora o toolkit a ofereça e outras funcionalidades na pipeline façam uso deste método - devido a necessitar de um input humano.

Tendo selecionado um bloco para realizar um corte e, em seguida, clicado na funcionalidade de corte,

o editor irá entrar no estado especial de recorte. Este é caracterizado por uma linha de corte aparecer sob o bloco selecionado, servindo como auxílio para a divisão do bloco em 2.

Clicando no botão esquerdo do rato neste estado realizará o corte de acordo com a linha visível. O texto será também dividido entre os dois blocos de acordo com a linha de corte escolhida. Naturalmente, a divisão do texto segue na realidade as coordenadas da linha de corte e da OCR Tree respetiva ao bloco, sendo que se a bounding box não estiver propriamente alinhada com o texto respetivo, esta divisão poderá não fazer sentido visualmente.

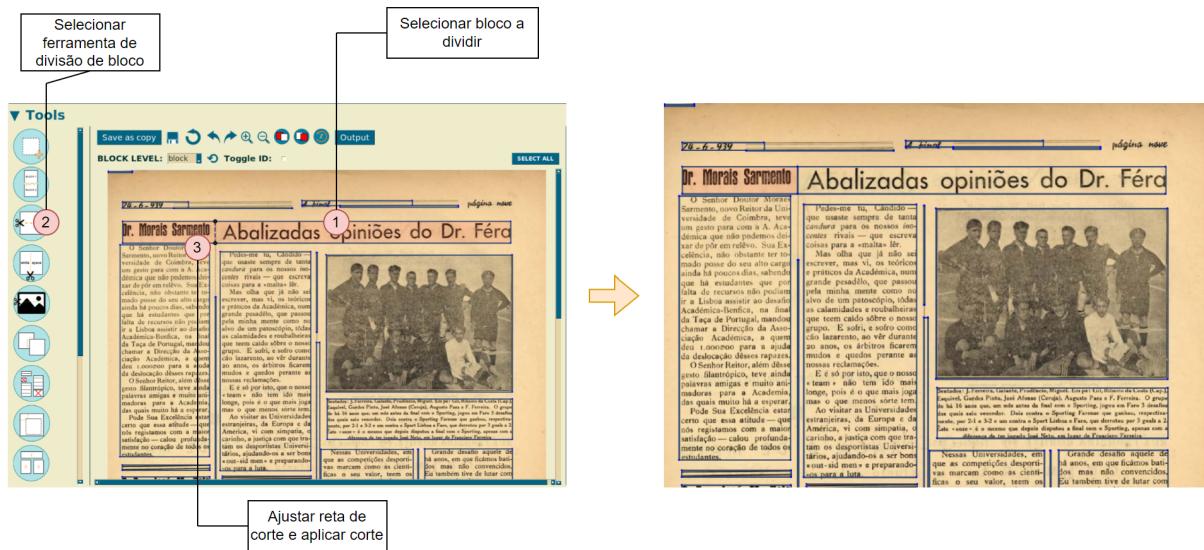


Figura 67: OCR Editor: divisão de bloco.

## Divisão por espaços vazios

Esta funcionalidade é semelhante à disponibilizada na pipeline com o mesmo nome, que por sua vez provém do toolkit.

A distinção principal neste cenário é que pode ser utilizada num ambiente mais controlado, apenas afetando, caso desejado, os blocos selecionados.

Caso nenhum bloco esteja selecionado, todos os blocos serão afetados.

Devido à característica de análise estatística do algoritmo para verificação das razões dos espaços vazios, mesmo no caso de haverem blocos selecionados, será introduzido no método a OCR Tree geral.

## Junção de blocos

Utilizando métodos de junção de múltiplas OCR Tree, tal como na pipeline no bloco de união de blocos, esta é uma utilização manual de junção de blocos.

Existem, como discutido no Toolkit, duas formas principais de junção de blocos, horizontal ou vertical.

No caso do Editor, basta selecionar dois ou mais blocos para junção e clicar na ferramenta. O algoritmo de junção irá automaticamente unir os dois textos tendo em conta as suas coordenadas, podendo ficar com linhas intercaladas.

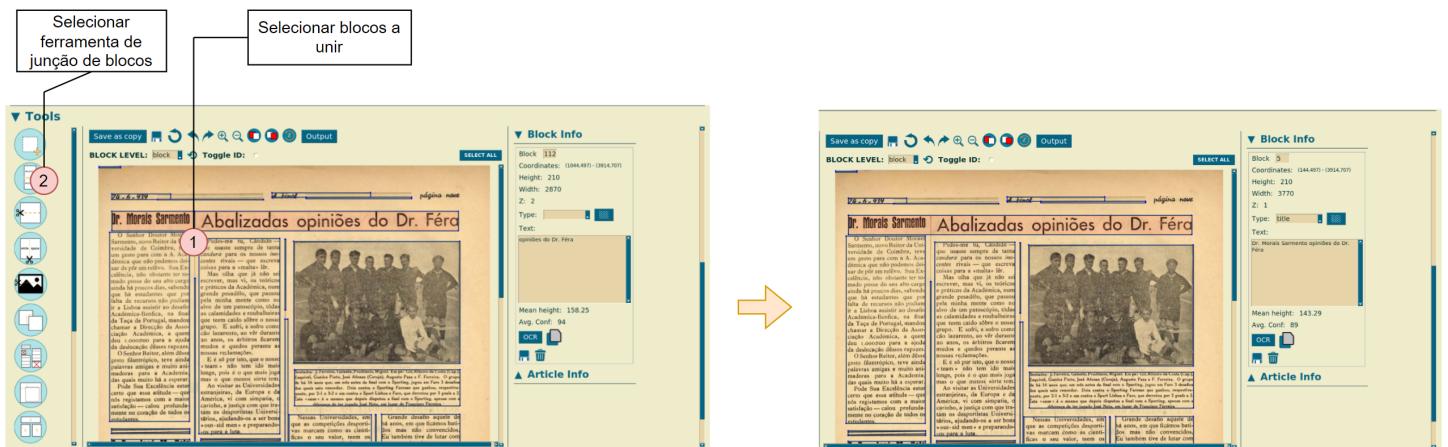


Figura 68: OCR Editor: junção de bloco.

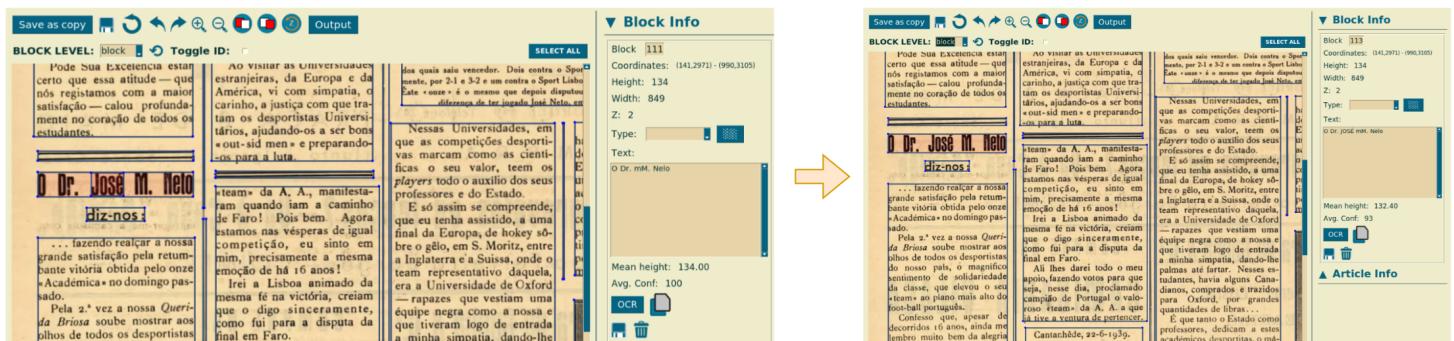


Figura 69: OCR Editor: junção de bloco, texto intercalado.

(a) Neste exemplo foi removida a palavra "José" do bloco, e criado um novo bloco com essa mesma palavra; posicionou-se o novo bloco na posição pretendida; juntou-se os blocos.

## Categorização de blocos

A tokenização dos blocos é especialmente importante para os algoritmos de cálculo de atração entre blocos e consequentemente cálculo de ordem de leitura; e de divisão dos blocos de texto em artigos.

Desta forma, a possibilidade de realizar uma categorização manual, para maior confiança ou correções em relação ao algoritmo automático, é evidentemente benéfico.

Tal pode ser conseguido selecionando um bloco e, na aba de edição do bloco, escolher o tipo deste. Por fim, é necessário clicar para guardar as modificações no bloco.

Por defeito, o editor não ilustra os tipos dos blocos, sendo necessário escolher a opção para os diferentes tipos de blocos serem coloridos.



Figura 70: OCR Editor: categorização de blocos.

## Ordenação de blocos

O algoritmo que mais beneficia da possibilidade de correções manuais é o de ordenação de blocos. Tal deve-se à notável mutabilidade dos templates de jornais.

Deste modo, o editor permite correções, ou ordenações completas dos blocos, de forma simples. Existem 2 formas de realizar a reordenação de um bloco selecionado: simplesmente escrevendo o lugar do bloco na ordem, embora tal tenha restrições de tempo entre cada número do id premido; na aba de edição do bloco, alterar o id para o desejado e guardar as alterações.

O lugar na ordem dos blocos, serve também como id do bloco, sendo que, para ambos os métodos, no caso de haver conflito entre o id dos blocos, estes serão imediatamente ajustados.

Ao mesmo tempo, também é possível usar o método do Toolkit - também usado na pipeline -, para cálculo automático da ordem de leitura. Caso estejam blocos selecionados no uso desta ferramenta, o seu comportamento será mais simples, apenas reordenando os seus ids de acordo com a ordem de seleção e o id do primeiro bloco selecionado.



Figura 71: OCR Editor: ordenação de blocos.

## Segmentação de blocos em artigos

Na pipeline, a segmentação da OCR Tree em artigos é uma das últimas operações realizadas, sendo o seu output extremamente dependente do sucesso das operações antecedentes. Novamente, a manipulação manual destes aparenta-se útil.

Uma aba específica para edição destes está presente no editor.

Várias operações estão relacionados com a manipulação de artigos:

**Segmentação da OCR Tree em artigos** utilizando o método semelhante ao discutido para a pipeline.

**Adicionar ou remover blocos de um artigo**, selecionando o artigo, o que irá atualizar a lista de blocos selecionados com os pertencentes ao artigo. Selecionar blocos não pertencentes ao artigo, no caso de se querer adicionar blocos; ou clicar em blocos dentro dos selecionados, para remover blocos do artigo. Por fim, atualiza-se o artigo para guardar as alterações.

**Adicionar um artigo**, ao selecionar bloco(s) e carregar para adicionar um artigo.

**Remover um artigo**, selecionando o artigo e clicando para o remover.

**Alterar ordem dos artigos**, ao selecionar um artigo e no botão para subir ou descer o seu lugar na ordem de artigos. Esta ordem tem influência na ordem em que os artigos aparecerão no output.

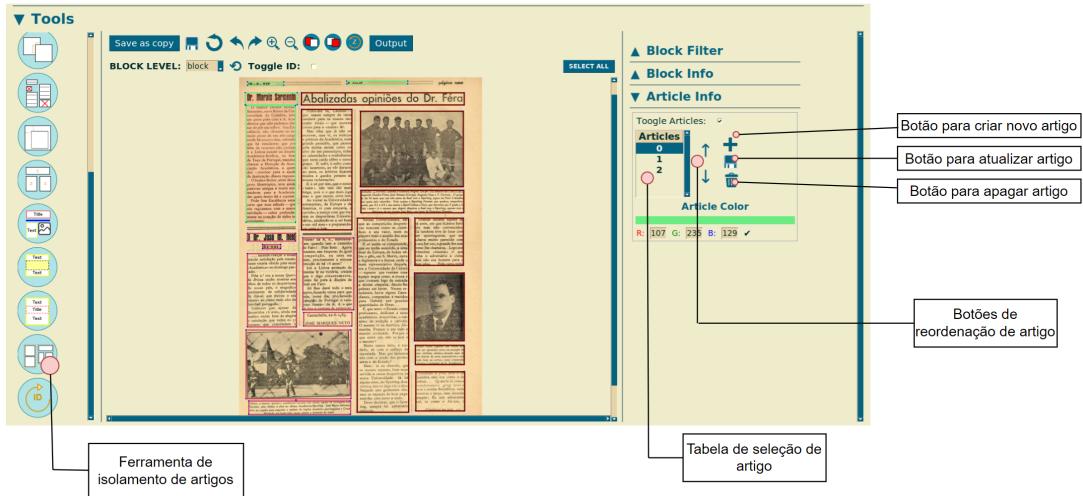


Figura 72: OCR Editor: manipulação de artigos.

## Divisão de imagem

De forma a facilitar a criação de resultados OCR em locais específicos de uma imagem, foi implementada uma funcionalidade única para o OSDOCR Editor que possibilita o corte de uma imagem.

Esta funciona de forma semelhante ao corte de um bloco, onde será realizado um corte que será visualmente representado por uma linha.

Antes de o corte ser realizado, será questionado qual das partes da imagem se pretende manter.

Realizado o corte, uma nova imagem (recortada da original) será criada, e os nodos da OCR Tree inseridos na área recortada serão adicionados na OCR Tree correspondente a esta imagem. Realça-se que apenas os nodos que apresentem uma bounding box completamente dentro da área serão incluídos, sendo que será recorrente pedaços significativos de texto percam o seu respetivo bloco, no caso de este texto ter continuação numa área fora do recorte.

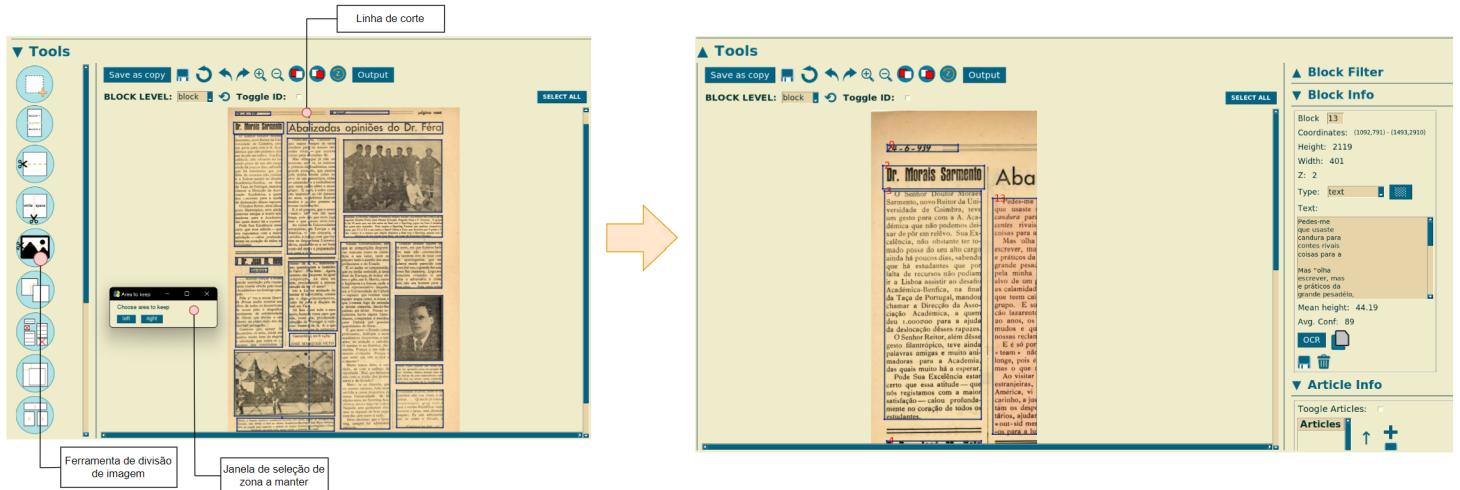


Figura 73: OCR Editor: divisão de imagem.

## 7.2.5 Outputs

O editor permite ainda a geração de outputs textuais semelhantes aos da pipeline.

É possível a geração de ficheiros do tipo *txt* ou *markdown*, para documentos do tipo jornal - criando artigos caso não existam - ou simples. O tipo de documento pode ser modificado através das configurações do editor.

Este output textual é opcional, ao contrário da pipeline.

A OCR Tree utilizada como input será modificada - caso operações sob ela sejam aplicadas -, sendo portanto um output resultante da utilização do editor.

Utilizando o método de divisão de imagem, o recorte da imagem também será também guardado como uma nova imagem, pareado com o respetivo ficheiro OCR Tree.



Figura 74: OCR Editor: output.

## 7.2.6 Operações adicionais

Serve esta última secção para descrever outras funcionalidades do editor, que se agrupam mais dentro de *quality of life* ao invés de pura manipulação dos dados.

### Cache de operações

Como na generalidade dos editores, este não é exceção na habilidade de retroceder e reconstruir operações. Tal permite desfazer ou refazer ações rapidamente, tornando o editor uma ferramenta ainda mais útil para a manipulação destes tipos de dados que podem ser consideravelmente densos.

Esta funcionalidade é possível através do uso de uma cache que vai sendo atualizada após a realização de cada operação.

O tamanho desta cache está por defeito definido para 10, mas pode ser alterado nas configurações do editor.

### Filtro de blocos

A capacidade de filtragem em ferramentas de edição traz proveito para a sua utilização, através da permissão de maior controlo e focagem de conteúdos interessantes.

Exemplo disto é a existência de tipos essenciais para a criação de outputs especiais, como é o caso do "title" para a geração de artigos. A possibilidade para facilmente distinguir quais os blocos deste tipo existentes é essencial.

De forma sucinta, os filtros disponíveis são:

- Nível de bloco
- Tipo de bloco
- Id de bloco
- Texto de bloco
- Coordenadas de bloco

## **Altura de blocos**

A disponibilização de um referencial de altura para os blocos permite a resolução de conflito entre blocos que se intersetam ou estão dentro de outros.

Nestes casos, se não houver distinção da altura entre estes, não será possível selecionar o pretendido entre os dois.

No entanto, selecionando um dos blocos e, alterando a sua altura, torna-se possível criar distinção entre os dois para mais facilmente selecionar aquele que anteriormente não era diretamente selecionável.

## **Redimensionamento do input**

O editor permite ainda a realização de redimensionamento da imagem de input para facilitar a visualização e edição local da respetiva parte da OCR Tree nesta. A OCR Tree é automaticamente deslocada sempre que um redimensionamento ocorre.

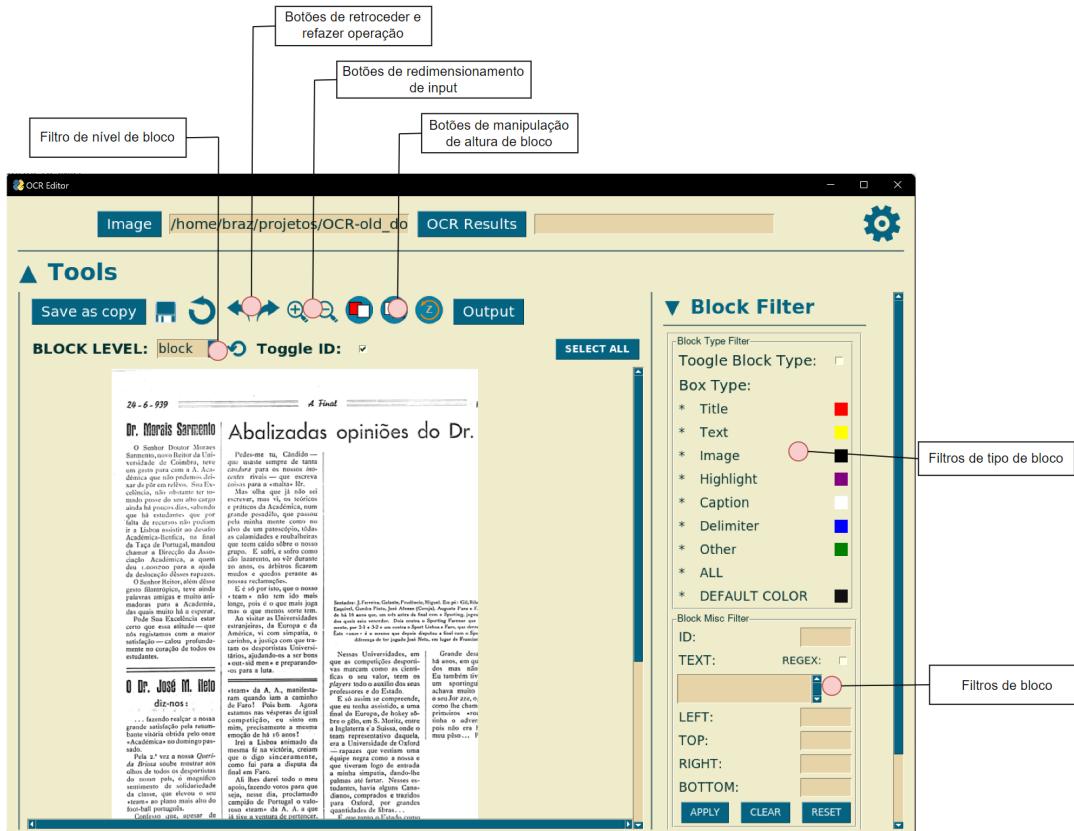


Figura 75: OCR Editor: operações adicionais.

## Configurações do editor

De forma a poder alterar a utilização do editor, uma janela de configurações foi disponibilizada. Esta é composta por 3 abas principais: configurações gerais do editor; configurações das ferramentas; configurações da pipeline.

As **configurações gerais**, abrangem definições visuais do editor, como zoom inicial, cor e grossura das bounding boxes; e definições do output, como o formato, tipo e tratamento de texto.

As **configurações das ferramentas**, permitem modificar o comportamento de algumas das ferramentas, por exemplo: definir se no split de imagem, se mantém todas as caixas que intersetam ou estão dentro da zona escolhida, ou apenas as que estão completamente dentro.

As **configurações da pipeline**, são uma versão reduzida das configurações da OSDOCR Pipeline. Estas são na realidade também configurações de uma ferramenta mas, sendo a quantidade de opções considerável, torna-se mais simples para o utilizador ter uma janela dedicada a esta ferramenta.

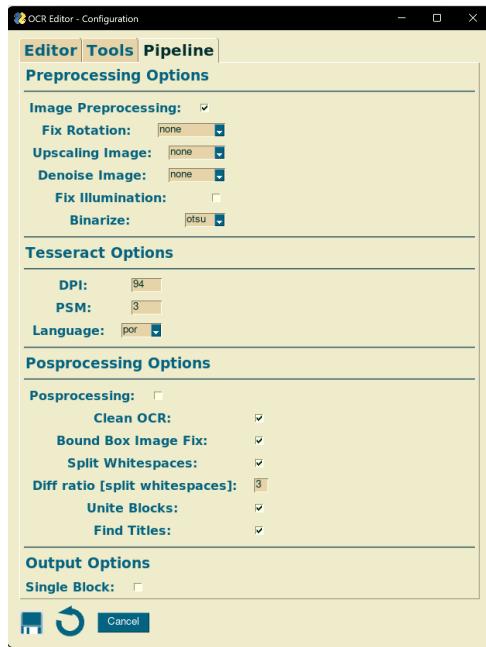


Figura 76: OCR Editor: janela de configurações.

## **Capítulo 8**

# **Resultados**

Nesta secção será realizado um estudo sobre os resultados e competências da solução implementada.

Como discutido previamente, a solução desta tese é composta por 3 módulos distintos, cada um deles tendo sido explorado em capítulos dedicados, mas dos quais 2 - OSDOCR Pipeline e OSDOCR Editor - são dependentes e produtos do 1º, sendo este, OSDOCR Toolkit, a base do projeto como um todo.

Desta forma, será dedicada uma secção para cada uma destas componentes, com especial tratamento para a primeira pois, sendo um caso particular de implementação que dificulta uma apreciação sucinta, pode ser tomada como seus resultados, os próprios resultados das outras duas componentes.

As duas restantes componentes terão também por sua vez métricas de apreciação distintas, que serão nas respetivas secções explicadas.

### **8.1 OSDOCR Toolkit**

O toolkit para melhoria do processo de OCR serviu como objetivo base do atual trabalho. Este, através do estudo do estado da arte, reflexão sobre a filosofia da solução, assim como pelo cílico processo de implementação e teste, evoluiu para englobar diferentes aspectos da aplicação de OCR: desde o processamento de imagem; tratamento, análise e manipulação dos resultados de OCR; e tratamento de texto. Dentro destes foi evidente o foco na segunda componente do processo, sendo a que menor atenção tem nas soluções correntes, que tendem a focar no processamento de imagem ou tratamento de texto fazendo uso de deep learning.

Existem diferentes formas de avaliar um toolkit, como: a apreciação da capacidade das suas ferramentas de forma individual; a sua utilidade em contextos além dos designados para a sua utilização; e a capacidade de integração em soluções mais complexas, i.e. soluções que façam uso do toolkit.

Ao longo do capítulo 5, à medida que eram expostas as diferentes ferramentas desenvolvidas, exemplos do uso destas foram sido fornecidos, estando uma demonstração individual destas já fornecido.

A solução engloba ainda 2 outras componentes que, como base, fazem uso deste mesmo toolkit. Estas 2 componentes, ambas também com o foco concreto de uso em jornais antigos, podem ser utilizadas em contextos extra, como revistas ou documentos atuais. Assim, através da avaliação destas 2 soluções, podemos simultaneamente, embora indiretamente, avaliar o Toolkit.

## 8.2 OSDOCR Pipeline

### 8.2.1 Metodologia

A pipeline de aplicação de OCR é, das 3 componentes da solução, a que mais facilmente pode ser objetivamente discutida. Esta é das 3 a que se assemelha mais aos trabalhos envolvendo aplicação de OCR, onde se tem um resultado direto e comparável com um valor inicial, i.e. texto real de um documento (ground truth), do qual se podem obter diferentes métricas, comparado com o texto resultante da aplicação de OCR numa imagem.

Seguindo um processo semelhante, obter-se-ão então resultados da aplicação desta componente sob diferentes casos de teste. Para isto, será feito uso do módulo de validação de resultados da pipeline (6.6).

Relativamente aos casos de teste escolhidos, foram escolhidos 18 diferentes jornais, totalizando 35 páginas a serem testadas (de cada jornal foram escolhidas em média 2 páginas). O número amostras é inferior ao ideal sendo limitado pelo escopo do projeto - com uma solução envolvendo 3 componentes, todas de considerável dimensão -; à necessidade de, na generalidade dos casos, ser necessária uma cuidada e manual transcrição das amostras escolhidas; e múltiplas utilizações de cada amostra para diferentes resultados.

As amostras forma obtidas das hemerotecas públicas da [casa de sarmento](#) e da [câmara municipal de Lisboa](#).

As amostras escolhidas apresentam, de modo a testar diferentes capacidades da pipeline, características particulares que as permitem caracterizar em :

- **Template compacto (4)** : templates com muito texto, compactado, e normalmente resolução abaixo da desejada;
- **Ordem de leitura complexa (2)** : templates que apresentam uma ordem de leitura não linear (cima para baixo, esquerda para a direita), fazendo uso do contexto do texto ou de guias (ex.: delimitadores) para guiar o leitor;
- **Moderno (1)** : jornais modernos, com resolução e estado ideal;
- **Inclinados (2)** : jornais que foram digitalizados com má orientação;

- **Template simples (4)** : Jornais que apresentam um template e, consequentemente, uma ordem de leitura simples. Texto/colunas bem espaçados;
- **Outros (5)** : outros tipos de documentos como: revistas, jornais infantis com muitas ilustrações, banda desenhada, etc.;

Para cada um destes documentos, uma transcrição foi preparada manualmente, assim como ficheiros adicionais com transcrição parcial, e com características do documento como o número de artigos, imagens e colunas. Estes serão usados como input do módulo de validação da pipeline. Este, como descrito na secção correspondente, procura comparar, através de diferentes métricas, a verdade de um documento (ground truth) com os resultados obtidos de diferentes configurações da pipeline.

As métricas que nos focaremos para a tabulação de resultados serão:

- Confiança média de texto
- Similaridade do texto (similaridade de cosseno)
- Rácio de aparições de palavras na GT e no resultado
- Rácio de palavras únicas na GT e no resultado
- Rácio de acerto da GT parcial no resultado
- Ordem geral da GT parcial no resultado
- Rácio de número de blocos de texto no resultado



Figura 77: Amostras do conjunto de casos de teste.

Sendo a pipeline mutável através da configuração a ela passada, diferentes configurações foram desenhadas para possibilitar a extração de mais conclusões. Note-se que esta lista não é extensiva, sendo que estas são multiplicativas ao número de execuções total.

Como base, será usada uma configuração de pipeline que não fará uso de nenhum módulo desta para além de OCR.

As outras configurações são, de forma sucinta:

1. **Pipeline completa**: utilizada para testar uma sequência de todos os módulos implementados.
2. **Pipeline completa sem ordenação**: possibilitando a comparação da ordenação original com a aplicação dos módulos.
3. **Apenas pré-processamento**: permitindo a avaliação dos módulos de imagem na transcrição
4. **Apenas pós-processamento**: afetando principalmente a capacidade de melhorar a leitura da transcrição geral
5. **Pipeline geral**: configuração que, ao longo do trabalho, se avaliou como sendo mais ubíquo (relativamente)
6. **Segmentação de imagem (e configuração geral)**: útil principalmente para casos de muito ruído ou texto compactado, procurando auxiliar a segmentação dos blocos.

Como características comuns entre estas configurações, todas utilizam o tesseract como motor de OCR, utilizando a configuração de linguagem 'por' para português visto ser esta a linguagem dos casos de teste. No caso de ser realizado upscaling da imagem, procura-se aumentar os dpi para 150, assumindo uma dimensão de folha A3, ocorrendo para a generalidade dos casos o upscaling (exceção a categoria moderna). Por último, todas utilizam confiança de texto mínima 10.

No total, serão analisadas 245 (7 configurações x 35 páginas) execuções da pipeline.

Todos os preparativos para esta análise (jornais, ground truths e configurações de pipeline), assim como os resultados pré tabulados, estão disponíveis [aqui](#).

### 8.2.2 Resultados

Segue-se uma redução dos resultados das métricas escolhidas para análise da componente. Estes serão apresentados na forma de gráficos de barras, sendo que, como mencionado, a totalidade dos dados está também disponibilizada com métricas extra não aqui focadas. Nestes gráficos o sujeito principal de comparação será a configuração simples, barra amarela, sempre a primeira para cada um dos documentos.

Versões ampliadas dos gráficos estão disponíveis na secção de apêndices.

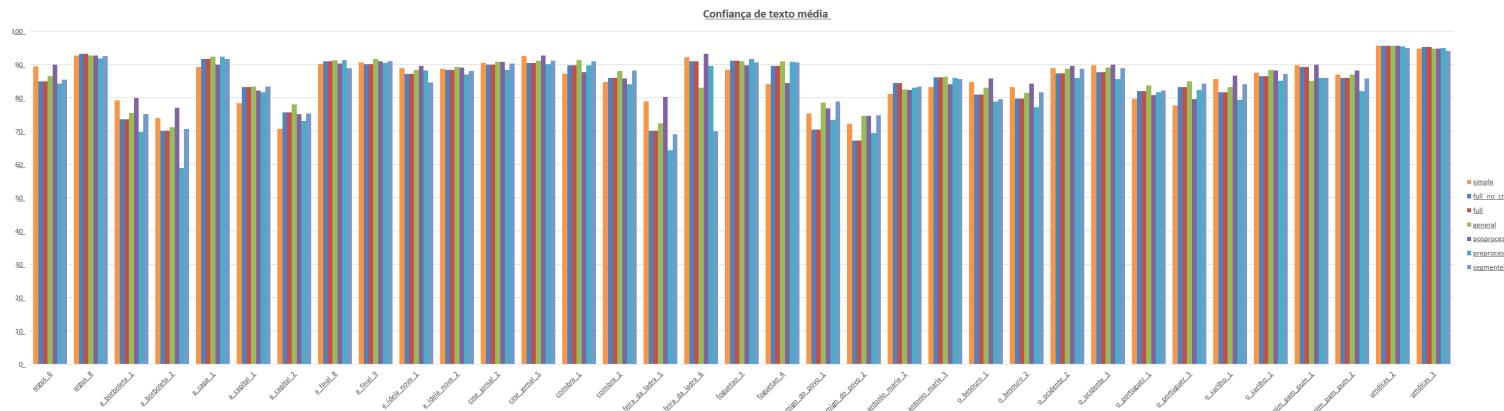


Figura 78: Valores de confiança média de texto das diferentes pipelines.

Pipeline	Média
simples	85.296
completa	84.838
completa (sem ordenação)	84.838
apenas pré-proc.	83.513
apenas pós-proc.	86.339
geral	85.854
segmentação	84.769

Tabela 1: Média geral de confiança de texto das pipelines.

Analizando os dados desta métrica, verifica-se que, na generalidade, não houve grandes oscilações na confiança média de texto, sendo que a pipeline geral e a de pós processamento obtiveram o melhor resultado médio, seguidos pela simples. Estas duas pipelines fazem uso controlado, ou nenhum de processamento de imagem, aproveitando os processos de limpeza do toolkit para remover texto sem confiança, acoplado com melhorias de imagem.

Por outro lado, tendo a pior média e maior oscilação, tem-se a pipeline de pré processamento, realçase a questão que já se tinha anteriormente estudado neste trabalho, que a utilização indiscriminada de ferramentas de processamento de imagem não é benéfica, sendo preferível estas serem adaptadas para cada documento.

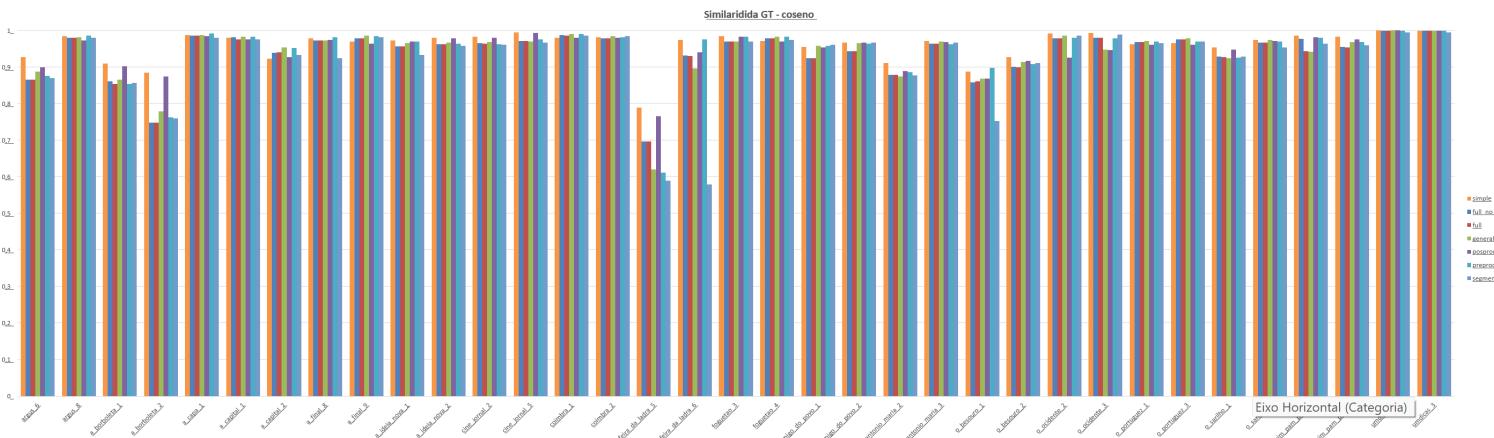


Figura 79: Valores de similaridade do texto (similaridade por cosseno) com GT das diferentes pipelines.

Pipeline	Média
simples	0.959
completa	0.939
completa (sem ordenação)	0.940
apenas pré-proc.	0.945
apenas pós-proc.	0.949
geral	0.941
segmentação	0.923

Tabela 2: Média geral de similaridade do texto com GT.

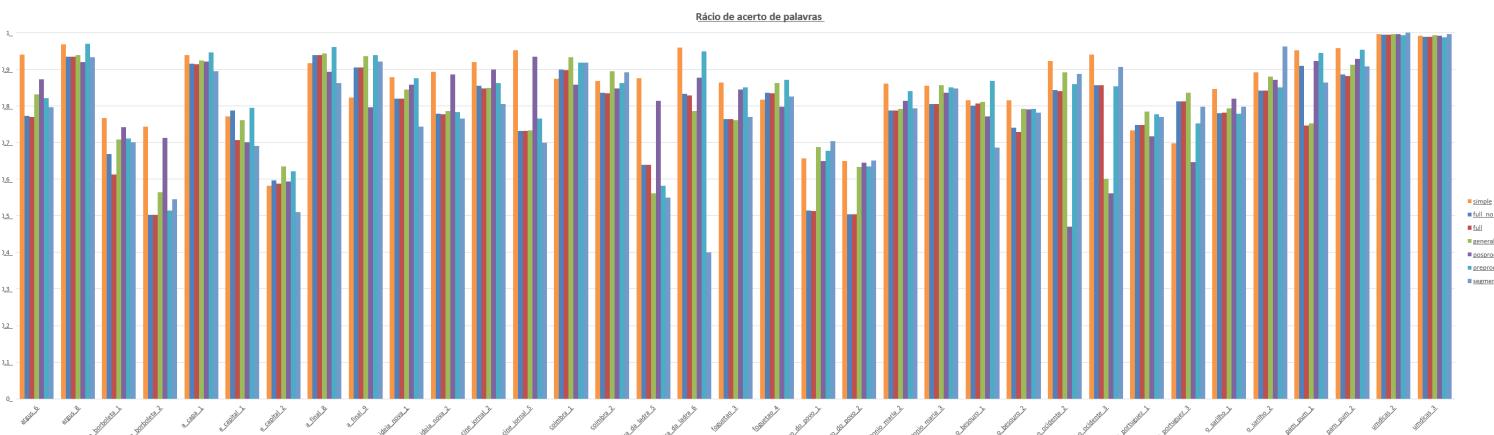


Figura 80: Ráios de aparição total de palavras da GT das diferentes pipelines.

Pipeline	Média
simples	0.855
completa	0.786
completa (sem ordenação)	0.796
apenas pré-proc.	0.829
apenas pós-proc.	0.806
geral	0.808
segmentação	0.788

Tabela 3: Média geral de rácios de aparição total de palavras da GT.

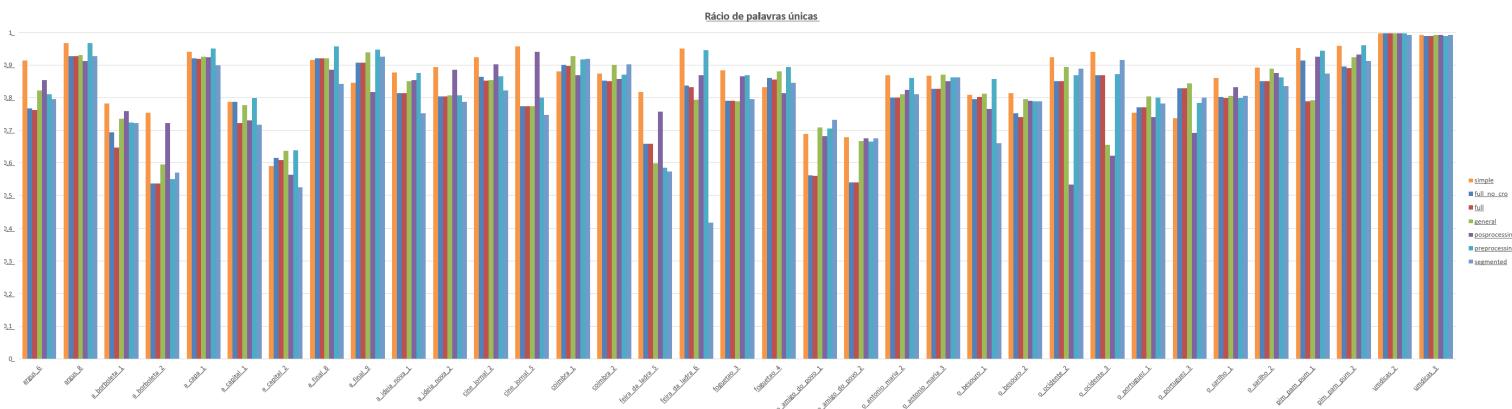


Figura 81: Rácios de aparição de palavras distintas da GT das diferentes pipelines.

Pipeline	Média
simples	0.860
completa	0.799
completa (sem ordenação)	0.808
apenas pré-proc.	0.839
apenas pós-proc.	0.814
geral	0.820
segmentação	0.794

Tabela 4: Média geral de rácios de aparição de palavras distintas da GT.

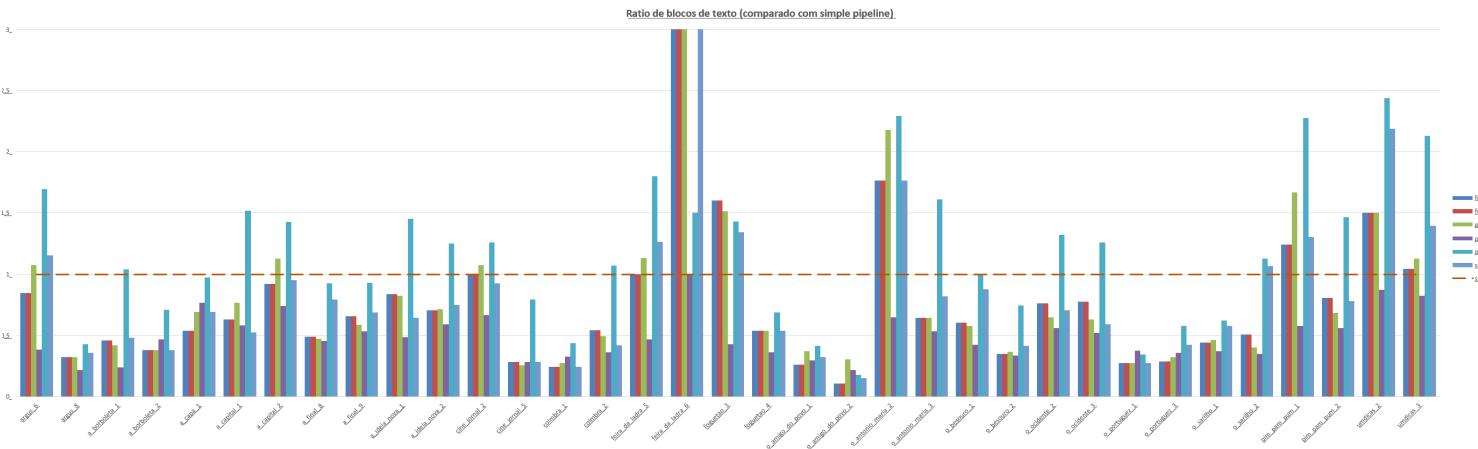


Figura 82: Rácios de número de blocos de texto relativo à pipeline simples.

Pipeline	Média
completa	0.767
completa (sem ordenação)	0.767
apenas pré-proc.	1.175
apenas pós-proc.	0.490
geral	0.795
segmentação	0.974

Tabela 5: Média de rácio de número de blocos de texto comparado com pipeline simples.

Verificando as métricas respetivas á ground truth, nota-se que em geral os resultados são mais favoráveis para o uso da template simples no que toca a métricas de similaridade simples de texto. Localmente, no entanto, podemos verificar que em certas instâncias, por exemplo, no documento 'o\_portuguez\_3', o rácio de palavras únicas detetadas melhorou cerca de 11% com a pipeline geral relativamente ao OCR simples, e 14% de acerto geral das palavras, com a generalidade das pipelines sendo melhor do que a base.

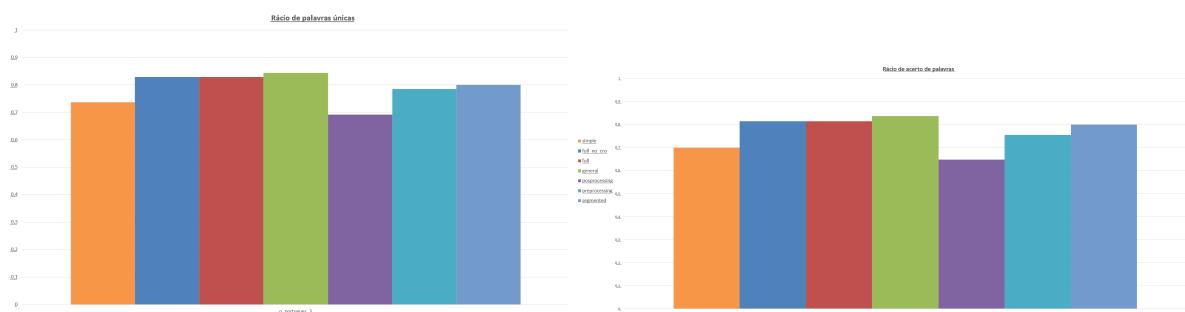


Figura 83: Resultados locais de 'o\_portuguez\_3' para rácio de palavras totais e distintas.

Por outro lado, assumindo que, embora na generalidade inferior, as diferentes configurações se assemelhem à base, nota-se a diferença na capacidade de reduzir a quantidade de dados da OCR Tree, i.e. quantidade de blocos nos resultados finais. Isto apenas se nota, naturalmente, nas pipelines que apresentam pós processamento. A diminuição da quantidade de blocos finais constitui uma redução da complexidade do resultado, permitindo uma análise mais fácil deste.

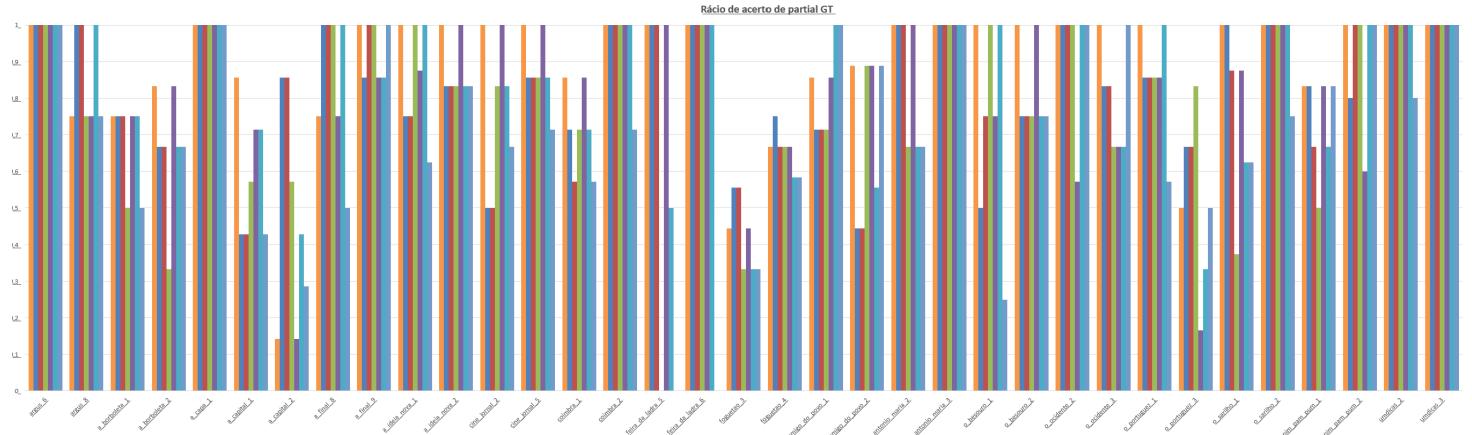


Figura 84: Ráios de aparição de linhas da Partial GT das diferentes pipelines.

Pipeline	Média
simples	0.889
completa	0.828
completa (sem ordenação)	0.826
apenas pré-proc.	0.809
apenas pós-proc.	0.820
geral	0.778
segmentação	0.680

Tabela 6: Média de rácio de acerto das linhas da Partial GT.

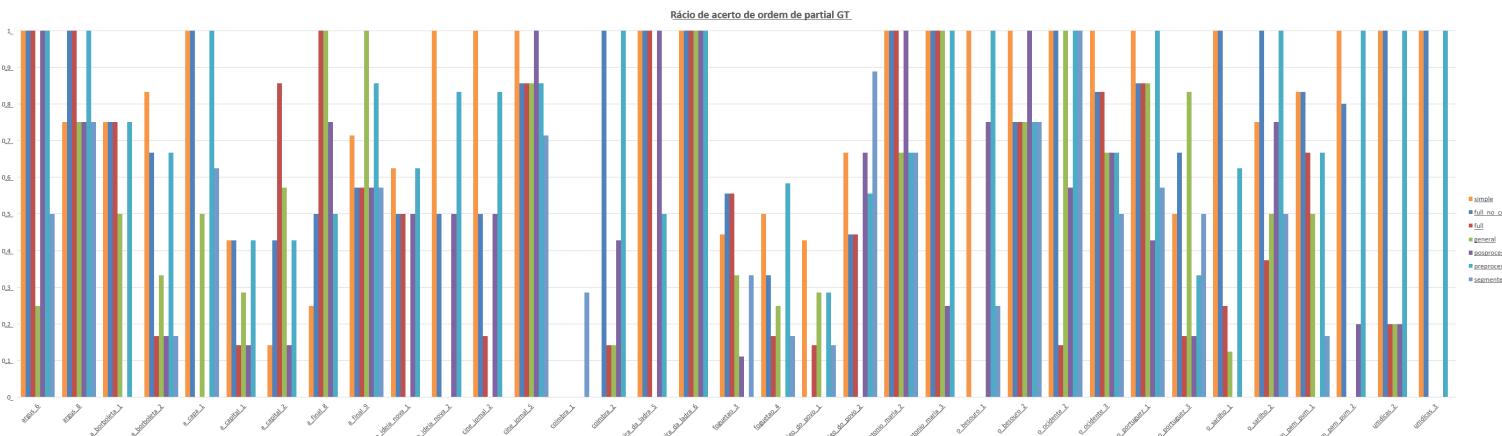


Figura 85: Ráculos de acerto da ordem das linhas da Partial GT das diferentes pipelines.

Pipeline	Média
simples	0.760
completa	0.477
completa (sem ordenação)	0.708
apenas pré-proc.	0.726
apenas pós-proc.	0.435
geral	0.433
segmentação	0.287

Tabela 7: Média de rácio de ordem das linhas da Partial GT.

No caso da GT parcial, maioritariamente dedicada à verificação de localização de algumas frases da GT e da sua ordem no resultado final, pode-se inferir que a aplicação de cálculo da ordem de leitura do Toolkit não é benéfica na generalidade, enfatizado pela diferença entre a média de rácio de ordem correta entre as configurações 'completa' e 'completa (sem ordenação)'.

Porém, observando individualmente, podem-se observar múltiplos casos em que este melhorou notavelmente os resultados em relação ao OCR simples como: 'a\_final\_8' (5x melhor); 'a\_capital\_2' ( 5.6x); 'o\_portuguez\_3' ( 1.6x).

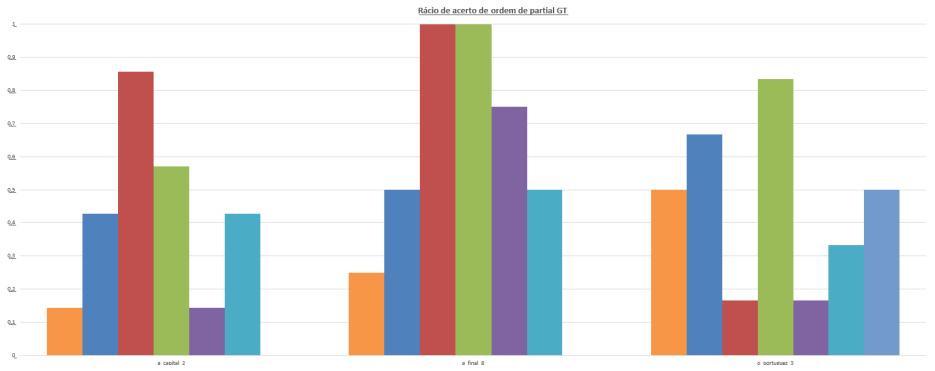


Figura 86: Resultados locais de casos de melhoria na ordenação da GT parcial.

### 8.2.3 Conclusão

Na generalidade, é possível concluir que, pelo menos dentro das configurações testadas, não houve nenhuma que excedesse os resultados de OCR simples.

Em casos particulares, no entanto, podemos observar vários cenários de melhoria dos resultados.

Seguem-se exemplos:

- Melhorias substâncias nas taxas de acerto da GT parcial no jornal 'a\_capital\_2' relativamente a OCR simples.

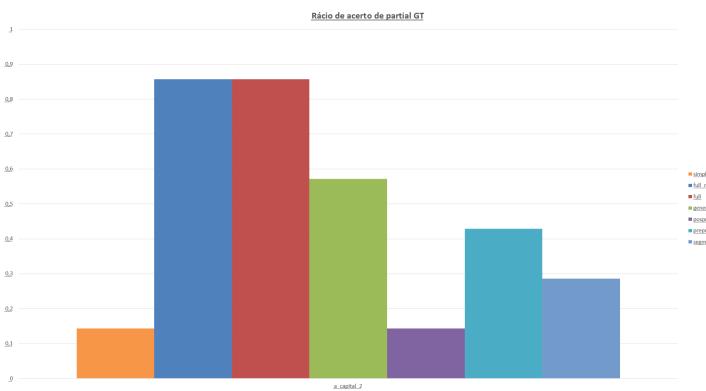


Figura 87: Exemplo de melhoria substancial de melhoria de acerto na GT parcial.

- Para os documentos 'argus\_8','a\_final\_8','a\_final\_9' e 'o\_sarilho\_2' com total acerto na ordenação da GT parcial por uma das configurações, sendo que OCR simples não o conseguiu e até teve resultados muito inferiores em alguns dos casos.

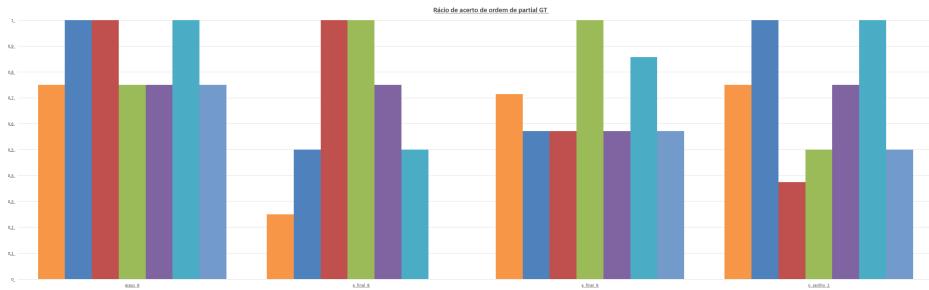


Figura 88: Exemplos de acerto total de ordenação da GT parcial.

Concluindo, os testes realizados permitem entender que a pipeline, no seu estado atual, tem o potencial para, quando devidamente configurada, melhorar os resultados relativamente ao uso base de OCR. Como esperado, também se verificou que a criação de uma configuração universal será difícil de alcançar devido às oscilações que cada configuração, especialmente de pré processamento, provocam na globalidade dos resultados.

Deste modo, uma perspetiva de evolução passaria pela permissão de adaptação interna da pipeline, ou de uma aplicação semelhante do Toolkit, que procure adaptar de forma inteligente as suas configurações de acordo com o input.

## 8.3 OSDOCR Editor

A última componente a analisar é o editor. Este tem como base o Toolkit e, como *modus operandi*, a manipulação da estrutura OCR Tree.

A sua proposta principal, como discutido nas proposições do capítulo 3, é a disponibilização de um ambiente gráfico para fácil manipulação da estrutura de dados universal OCR Tree, consequentemente permitindo fazer reparos minuciosos nos resultados da pipeline e, servir como uma poderosa ferramenta de debugging da pipeline e do toolkit.

Através das suas funcionalidades, além da persistente dependência no Toolkit, também faz uso da pipeline - como é exemplo a aplicação de OCR localmente, ou reutilização do módulo de extração de output -, sendo portanto relevante como parcial resultado destes, especialmente do toolkit.

Como o toolkit, a avaliação desta componente não é direta, a não ser num cenário de extensa listagem das funcionalidades, isoladas e combinadas, analisadas em diferentes casos de utilização. Alguns destes já foram descritos para funcionalidades mais relevantes e únicas desta componente, no seu capítulo de implementação.

Outra forma de avaliar esta ferramenta, mais subjetiva embora não menos relevante, provém do uso

de grupos de utilizadores. A estes poderiam ser dados conjuntos de tarefas a realizar, e registadas as dificuldades (ou falta delas) na sua execução, opiniões sobre as capacidades da ferramenta, e capacidade de as executar sem apoio externo. Esta metodologia não foi seguida, por restrições de tempo, mas é importante realçar a sua relevância no contexto de disponibilização da ferramenta para um público mais amplo, assim como a sua acessibilidade para sujeitos menos envolvidos no contexto técnico.

A avaliação seguida baseia-se então na apresentação das capacidades e adaptabilidade desta interface gráfica, através da listagem de casos de uso distintos e contrastantes.

- Análise e manipulação de ficheiros do tipo OCR Tree, i.e. formato json e hocr : aplicabilidade para resultados de soluções além da OSDOCR pipeline.
- Visualização de diferentes níveis da OCR Tree.
- Filtragem da OCR Tree: através de filtros de tipo, texto, coordenadas e ID.
- Fácil retrocesso e reconstrução de operações complexas sob a OCR Tree: através de uma cache de OCR Tree.
- Limpeza de blocos de ruído de OCR Tree.
- Ajuste de dimensões e posicionamento de blocos da OCR Tree.
- Divisão de blocos através da ferramenta de corte : sem GUI seria necessário manualmente modificar as árvores, respetivamente atualizando o texto dos filhos e as suas coordenadas.
- Junção facilitada de blocos : sem GUI seria necessário modificar as árvores, mantendo apenas uma modificando as suas coordenadas e, especialmente, realocando os filhos de acordo com o seu posicionamento. Especialmente difícil na junção de árvores com texto que se intercala ou sobrepõe.
- (Re)categorização de blocos.
- Modificação direta do texto de blocos : manualmente seria necessário modificar as folhas para atualizar ou adicionar palavras, e criar novos nodos para cada linha e parágrafo.
- Criação de blocos não existentes.
- Realizar OCR num bloco : servindo para melhorar texto e a sua confiança por transcrição máquina.
- Realizar OCR num segmento da imagem: gera, de acordo com a segmentação do motor OCR, múltiplos blocos com uma só ferramenta.
- Aplicar pipeline localmente: permite adaptar o OCR realizado configurando a pipeline para lidar com o segmento da imagem escolhido. Ex.: pipeline com upscaling numa imagem inteira pode não detetar com tanta precisão texto de um título específico, comparado com a aplicação da mesma pipeline localmente (usualmente produto de dpi assumido pelo tesseract).

- (Re)ordenar blocos através da modificação do seu ID: modificação direta ou com ferramentas do Toolkit.
- Gerar OCR Tree numa imagem sem resultados.
- Visualizar resultados da OSDOCR Pipeline: útil para contexto de debugging e desenvolvimento.
- Corrigir resultados da OSDOCR Pipeline: sendo que este durante as vários módulos produz uma OCR Tree, pode-se corrigir um ponto específico.
- Visualizar resultados do OSDOCR Toolkit.
- Conversão de OCR Tree em output textual simples.
- Conversão de OCR Tree em output textual de artigos.
- Segmentação da OCR Tree em artigos : manual ou utilizando métodos do toolkit.
- Manipulação de artigos : reordenação, atualização, inserção e remoção.
- Criação de OCR Tree para segmento de uma imagem com respetiva OCR Tree : com input de imagem e OCR Tree, a ferramenta de divisão de imagem gera uma nova imagem, partição da primeira, com uma OCR Tree constando cópias dos blocos que estavam inseridos e/ou intersestados (dependendo das configurações do editor) na partição. Facilita assim a criação de resultados de OCR para partição de uma imagem que já possui resultados.
- Criação facilitada de resultados de confiança para jornais antigos manualmente: como intedido pela premissa da tese.
- Criação de resultados de confiança para documentos de outro tipo manualmente: não sendo tão focado para documentos que produzem resultados ruidosos, amplia a utilidade da solução; ex. documentos aplicáveis: banda desenhada, livros, revistas, recibos, etc..
- Suavização do processo de criação de ilustrações no contexto de OCR e manipulação da OCR Tree: como prova deste conceito, tem-se que a maioria das imagens desta dissertação fizeram uso do OSDOCR Editor

## **Capítulo 9**

# **Conclusões e trabalho futuro**

Neste capítulo será feito um sumário do trabalho e estudo realizado, tiradas conclusões sobre o projeto concebido, e uma introspeção sobre perspetivas de trabalho futuro.

## **9.1 O Trabalho**

A dissertação propôs como seu objetivo global, a criação de uma solução que pode-se ser acoplada ao processo de reconhecimento ótico de caracteres, tanto antes e depois deste, para a melhoria dos seus resultados.

Através do estudo do problema, focado essencialmente em jornais antigos - devido à sua mutabilidade de estrutura e geral estado deteriorado que afetam a transcrição máquina -, foi possível delimitar a solução em componentes distintas. Estas foram : um Toolkit ou conjunto de métodos que permitam realizar transformações, sobretudo nos resultados de OCR mas também incluindo processamento de imagem e texto, em via de reduzir erros na transcrição e facilitar a sua manipulação e análise dos resultados OCR; uma pipeline, estrutura semelhante à maioria de problemas que abordam este tema, e que permite a aplicação e consequente validação do Toolkit; um editor gráfico de resultados OCR que disponibilize um meio de manipulação de resultados OCR facilitada, assim como depuração facilitada - pelo seu aspeto visual - das duas anteriores componentes.

Essas 3 componentes tiveram em comum a necessidade de analisar e manipular resultados OCR, o que resultou na criação de uma estrutura universal para estes, OCR Tree.

Discutida a implementação deste modelo e componentes, foram também analisados os seus resultados quando aplicados em diferentes contextos, com especial atenção para a pipeline, sendo esta a mais permissiva de métricas objetivas.

O código desenvolvido e ficheiros de resultados pode ser encontrado em:

- [Repositório principal](#)

- [Repositório auxiliar com métodos para tratamento de imagem](#)
- [Ficheiros de resultados](#)

## 9.2 Conclusões

Concluído o trabalho, é possível comparar a sua visão inicial do estado atual. Num primeiro desenho do projeto, apenas se expectava a implementação de um conjunto de métodos independentes e, incontornavelmente, o modelo OCR Tree.

Este conjunto de ferramentas sucedeu na criação de uma estrutura de dados complexa, mas de comprehensiva manipulação e análise, tendo sido esta área o seu ponto forte. Além disso, na área de tratamento e análise de imagem, ferramentas de correção de anomalias de imagem e identificação de elementos de documento como delimitadores e colunas, que foram aplicados nas outras partes da solução. Porém, com o desenvolvimento desta componente entendeu-se a necessidade de um ambiente de teste da mesma.

Daqui nasce a pipeline, nova componente que, embora já desde inícios do trabalho pensada, inevitavelmente obrigou a divisão dos esforços na primeira componente, tendo essa pecado principalmente em aspetos de tratamento de texto. A pipeline, como aplicação do Toolkit, foi também aproveitada para abranger aspetos que este não iria abordar, como o upscaling de imagem. Utilizando a pipeline, é possível a observação direta do efeito do Toolkit na transcrição máquina, permitindo ainda, métricas objetivas através do módulo de validação nesta implementada.

Esta foi testada utilizando um conjunto de casos de teste - utilizando o próprio módulo de validação da pipeline-, os quais foram executados com diferentes configurações de características fundamentalmente diferentes, não adaptadas para nenhum caso em particular. Concluiu-se a partir destes teste que a pipeline é mais útil quando a configuração fornecida é adaptada aos problemas de um documento, sendo que não obteve na globalidade resultados notórios utilizando as configurações generalizadas. Por outro lado, em casos pontuais certas configurações, mesmo generalizadas, obtiveram melhorias consideráveis em relação a OCR base, demonstrando-se o seu potencial. Além disso, notou-se a utilidade da secção de pós processamento para diminuição da complexidade dos resultados OCR, e a oportunidade que uma evolução no módulo de tratamento de texto poderia trazer para o output final.

Na presença de questões mais minuciosas que a pipeline não foi capaz de lidar, como: ordens de leitura muito complexos, trechos de texto demasiado danificados para transcrição automática, segmentação insatisfatória do motor OCR; e também a dificuldade em visualizar o estado da OCR Tree durante as

diferentes etapas da Pipeline ou manipulações pelo Toolkit, gerou-se a oportunidade de incluir um editor visual. Este foi a última e mais tardia componente a ser incluída na solução mas, possivelmente, a mais útil. Esta componente permitiu uma suavização no uso da OCR Tree que, consideravelmente, habilitou a compensação de problemas ignorados ou não totalmente tratados pela Pipeline, assim como a potencialização do Toolkit, fazendo uso de partes dele até então desusadas. Adicionalmente, este meio de visualizar a manipulação permitida pelas anteriores componentes, permitiu mais facilmente detetar defeitos nelas ou possíveis melhorias, recursivamente melhorando-se ao simplificar o refinamento delas.

O compilado de todas as componentes foi essencial para um aprofundamento do conhecimento sobre a área de pré, e particularmente, pós processamento de OCR, modularização de soluções e, no caso do Editor, o trabalho sobre interfaces gráficas e editor de modelos.

### **9.3 Perspetiva de trabalho futuro**

Apesar de satisfeito e enriquecido com a realização do projeto, seria ingênuo ignorar os refinamentos que este poderia ser sujeito.

Como já referido, a área em que o Toolkit mais tem espaço para enriquecer trata-se do tratamento de texto, mais especificamente no ato de criação de output. Tal, como os tempos têm mostrado haver um maior potencial, daria-se provavelmente na exploração de Large Language Models para a realização de correções de texto, que possivelmente também poderiam permitir a restauração de texto não detectado/transcrito. Naturalmente, heurísticas mais determinísticas para tratamento de texto também seriam bem-vindas, como, por exemplo, as técnicas exploradas no estado da arte, de correção de palavras fazendo uso de léxicos conhecidos.

Do mesmo modo, embora neste caso já levemente explorado, o processamento de imagem fazendo uso de inteligência artificial seria inevitável, nomeadamente para: correção de distorções de imagens como curvaturas do texto ou inclinações internas; restauro de texto. De forma mais complexa, e como também foi mencionado no estado da arte, acoplar a visão máquina com os mecanismos criados para cálculo de ordem de leitura, categorização de texto e isolamento de artigos tem o potencial de aumentar a adaptabilidade destes. As restrições temporais e de recursos não permitiram porém divergência neste sentido, podendo por si só este ramo tornar-se num projeto único.

A estrutura OCR Tree poderia ser tornada mais 'universal' para OCR relaxando a responsabilidade dos níveis de blocos, visto pressupor atualmente na maioria dos cenários que o nível 2 é representativo de um bloco como um todo, e o 5 como o local de texto. Por exemplo, na procura de texto, poderia ser possível

incluir texto em blocos superiores, reduzindo a necessidade de sempre iterar às folhas para o obter; ou no caso de análise de texto, utilizar como base o nível fornecido ao invés de assumir os níveis 4 (linha) e 5 (palavra) como essenciais.

O módulo de output teria bem-vinda a conversão para um formato que mantivesse mais fielmente a estrutura original do documento, como pdf ou html. A existência de conversores de hocr para estes terá diminuído a sua prioridade.

Como discutido durante a análise de resultados, o estudo de uma pipeline auto configurável seria interessante para a criação de uma solução adaptável a documentos de diferentes características.

Por último, a exploração de diferentes motores OCR seria relevante para prova da potencialidade da OCR Tree como módulo para resultados OCR, e possibilitando mais uso de todas as componentes da solução.

## Bibliografia

Abdullah Almutairi and Meshal Almashan. Instance segmentation of newspaper elements using mask r-cnn. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 1371–1375, 2019. doi: 10.1109/ICMLA.2019.00223.

ALTO. Alto documentação. URL <https://www.loc.gov/standards/alto/techcenter/elementSet/index.html>.

Anukriti Bansal, Santanu Chaudhury, Sumantra Dutta Roy, and J.B. Srivastava. Newspaper article extraction using hierarchical fixed point model. In *2014 11th IAPR International Workshop on Document Analysis Systems*, pages 257–261, 2014. doi: 10.1109/DAS.2014.42.

Raphaël Barman, Maud Ehrmann, Simon Clematide, Sofia Ares Oliveira, and Frédéric Kaplan. Combining visual and textual features for semantic segmentation of historical newspapers. *Journal of Data Mining & Digital Humanities*, Histolnformatics(Histolnformatics), January 2021. ISSN 2416-5999. doi: 10.46298/jdmdh.6107. URL <http://dx.doi.org/10.46298/jdmdh.6107>.

Deepa Berchmans and S S Kumar. Optical character recognition: An overview and an insight. In *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCI/CCT)*, pages 1361–1365, 2014. doi: 10.1109/ICCI/CCT.2014.6993174.

Wojciech Bieniecki, Szymon Grabowski, and Wojciech Rozenberg. Image preprocessing for improving ocr accuracy. In *2007 International Conference on Perspective Technologies and Methods in MEMS Design*, pages 75–80, 2007. doi: 10.1109/MEMSTECH.2007.4283429.

Thomas Breuel. High performance document layout analysis. 05 2003.

Quang Anh Bui, David Mollard, and Salvatore Tabbone. Selecting automatically pre-processing methods to improve ocr performances. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 01, pages 169–174, 2017. doi: 10.1109/ICDAR.2017.36.

R.M. Samitha Chathuranga and Lochandaka Ranathunga. Procedural approach for content segmentation of old newspaper pages. In *2017 IEEE International Conference on Industrial and Information Systems (ICIIS)*, pages 1–6, 2017. doi: 10.1109/ICIINFS.2017.8300390.

Krishnendu Chaudhury, Ankur Jain, Sriram Thirthala, Vivek Sahasranaman, Shobhit Saxena, and Selvam Mahalingam. Google newspaper search – image processing and analysis pipeline. In *2009 10th International Conference on Document Analysis and Recognition*, pages 621–625, 2009. doi: 10.1109/ICDAR.2009.272.

Mostafa Darwiche, The-Anh Pham, and Mathieu Delalandre. Comparison of jpeg's competitors for document images. In *2015 International Conference on Image Processing Theory, Tools and Applications (IPTA)*, pages 487–493, 2015. doi: 10.1109/IPTA.2015.7367194.

Raghunath Dey, Rakesh Chandra Balabantaray, Surajit Mohanty, Debabrata Singh, Marimuthu Karuppiah, and Debabrata Samanta. Approach for preprocessing in offline optical character recognition (ocr). In *2022 Interdisciplinary Research in Technology and Management (IRTM)*, pages 1–6, 2022. doi: 10.1109/IRTM54583.2022.9791698.

Sébastien Eskenazi, Petra Gomez-Krämer, and Jean-Marc Ogier. A comprehensive survey of mostly textual document segmentation algorithms since 2008. *Pattern Recognition*, 64:1–14, 2017. ISSN 0031-3203. doi: <https://doi.org/10.1016/j.patcog.2016.10.023>. URL <https://www.sciencedirect.com/science/article/pii/S0031320316303399>.

Europeana. Projeto europeana. URL <https://pro.europeana.eu/project/europeana-newspapers>.

Dafang He, Scott Cohen, Brian Price, Daniel Kifer, and C. Lee Giles. Multi-scale multi-task fcn for semantic page segmentation and table detection. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 01, pages 254–261, 2017. doi: 10.1109/ICDAR.2017.50.

HOCR. Hocr documentação. URL <https://kba.github.io/hocr-spec/1.2/>.

KerasOCR. Keras ocr documentação. URL <https://keras-ocr.readthedocs.io/en/latest/examples/index.html>.

Samu Kovanen and Kiyoharu Aizawa. A layered method for determining manga text bubble reading order. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 4283–4287, 2015. doi: 10.1109/ICIP.2015.7351614.

Ankit Lat and C. V. Jawahar. Enhancing ocr accuracy with super resolution. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 3162–3167, 2018. doi: 10.1109/ICPR.2018.8545609.

Laurence Likforman-Sulem, Jérôme Darbon, and Elisa H. Barney Smith. Pre-processing of degraded printed documents by non-local means and total variation. In *2009 10th International Conference on Document Analysis and Recognition*, pages 758–762, 2009. doi: 10.1109/ICDAR.2009.210.

Benjamin Meier, Thilo Stadelmann, Jan Stampfli, Marek Arnold, and Mark Cieliebak. Fully convolutional neural networks for newspaper article segmentation. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 01, pages 414–419, 2017. doi: 10.1109/ICDAR.2017.75.

Rishabh Mittal and Anchal Garg. Text extraction using ocr: A systematic review. In *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*, pages 357–362, 2020. doi: 10.1109/ICIRCA48905.2020.9183326.

Thi Tuyet Hai Nguyen, Adam Jatowt, Mickael Coustaty, and Antoine Doucet. Survey of post-ocr processing approaches. *ACM Comput. Surv.*, 54(6), jul 2021. ISSN 0360-0300. doi: 10.1145/3453476. URL <https://doi.org/10.1145/3453476>.

opencv extract lines. Opencv algoritmo de extração de linhas horizontais e verticais. URL [https://docs.opencv.org/4.x/dd/dd7/tutorial\\_morph\\_lines\\_detection.html](https://docs.opencv.org/4.x/dd/dd7/tutorial_morph_lines_detection.html).

PaddleOCR. Paddleocr documentação. URL <https://github.com/PaddlePaddle/PaddleOCR>.

Lorenzo Quiros and Enrique Vidal. Learning to sort handwritten text lines in reading order through estimated binary order relations. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 7661–7668, 2021. doi: 10.1109/ICPR48806.2021.9413256.

Mohamed Ali Souibgui and Yousri Kessentini. De-gan: A conditional generative adversarial network for document enhancement. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(3):1180–1191, 2022. doi: 10.1109/TPAMI.2020.3022406.

Sargur N. Srihari, Ajay Shekhawat, and Stephen W. Lam. *Optical Character Recognition (OCR)*, page 1326–1333. John Wiley and Sons Ltd., GBR, 2003. ISBN 0470864125.

Tesseract. Tesseract documentação. URL <https://tesseract-ocr.github.io>.

Tan Chiang Wei, U. U. Sheikh, and Ab Al-Hadi Ab Rahman. Improved optical character recognition with deep neural network. In *2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA)*, pages 245–249, 2018. doi: 10.1109/CSPA.2018.8368720.

## Apêndice A

### Detalhes dos resultados

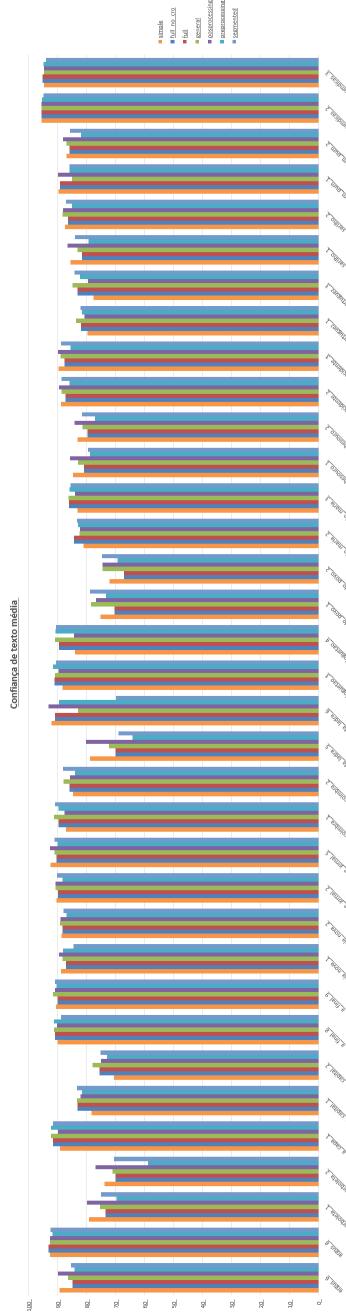


Figura 89: Valores de confiança média de texto das diferentes pipelines (alargado).

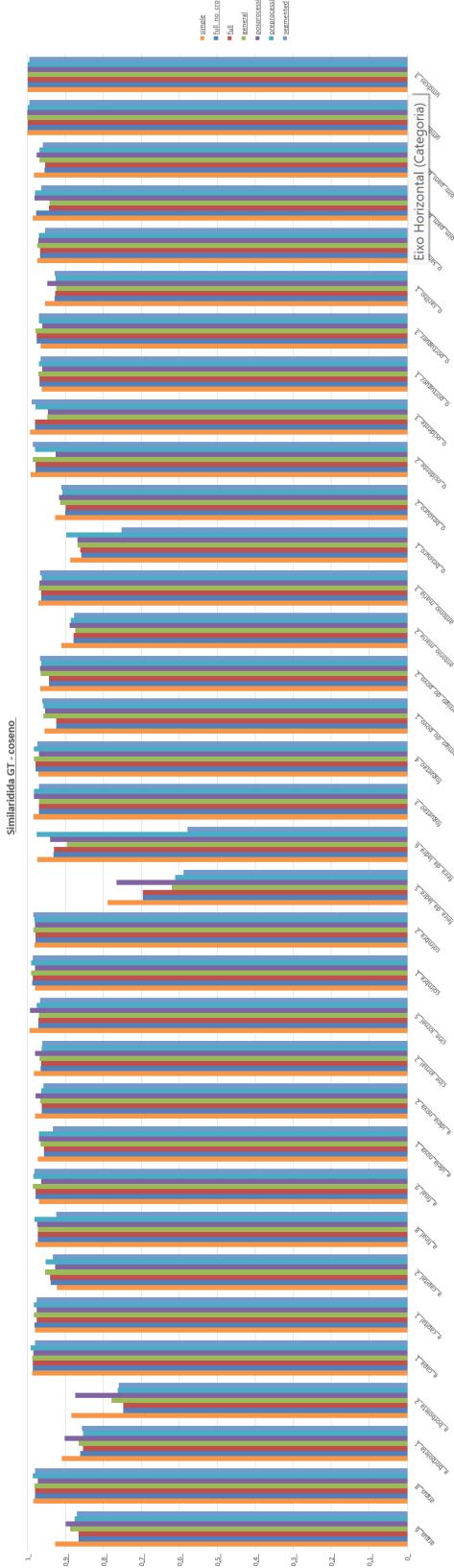


Figura 90: Valores de similaridade do texto (similaridade por cosseno) com GT das diferentes pipelines (alargado).

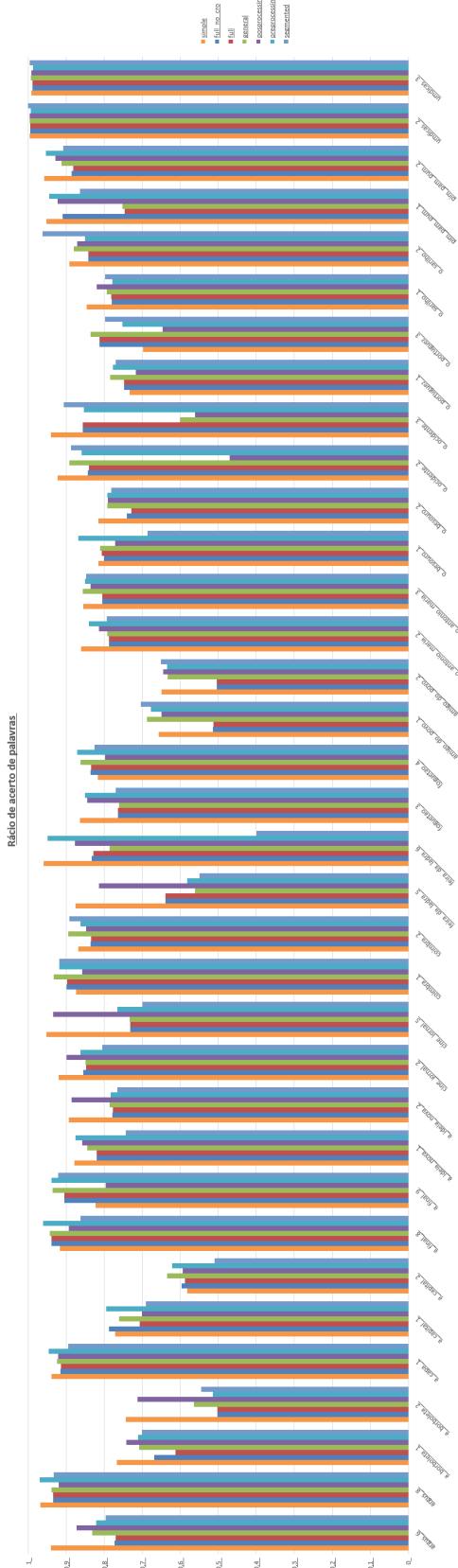


Figura 91: Ráios de aparição total de palavras da GT das diferentes pipelines (alargado).

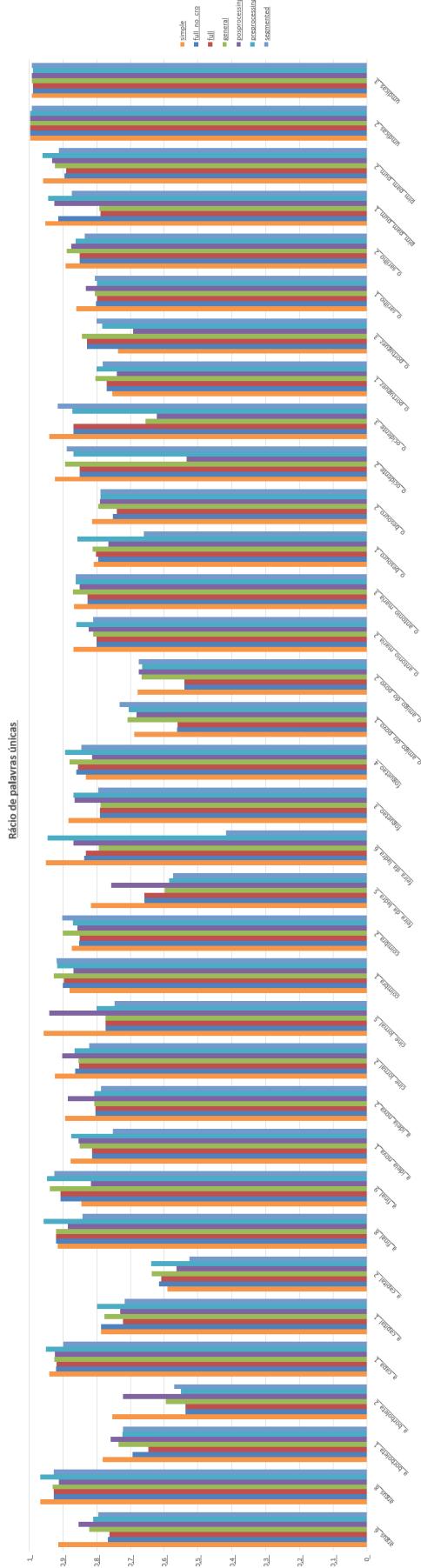


Figura 92: Ráios de aparição de palavras distintas da GT das diferentes pipelines (alargado).

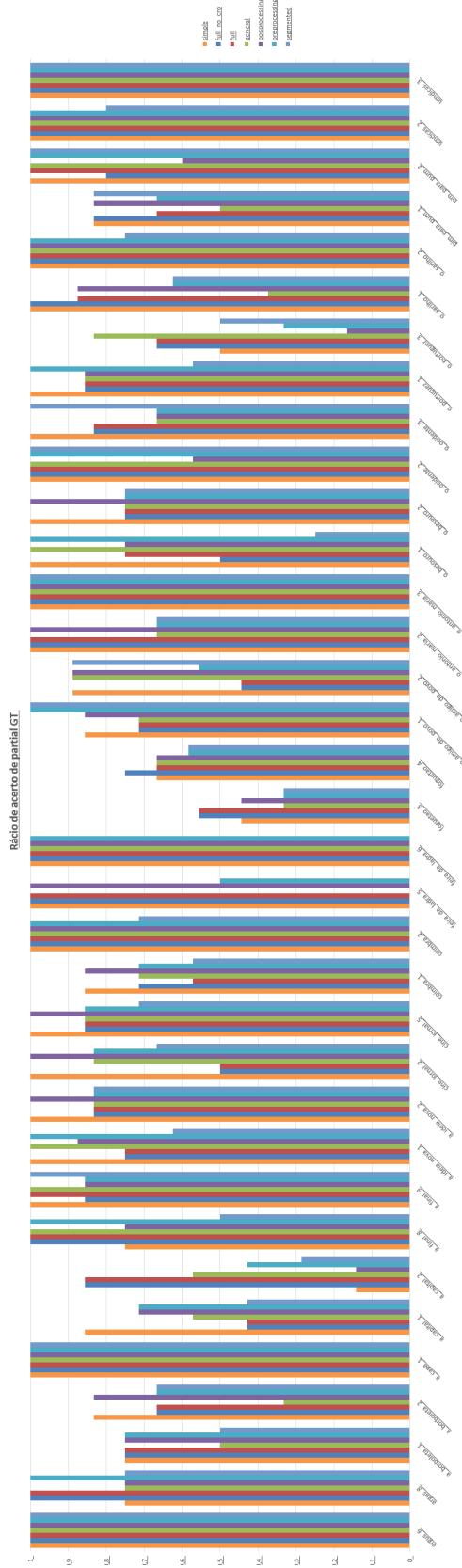


Figura 93: Ráios de aparição de linhas da Partial GT das diferentes pipelines (alargado).

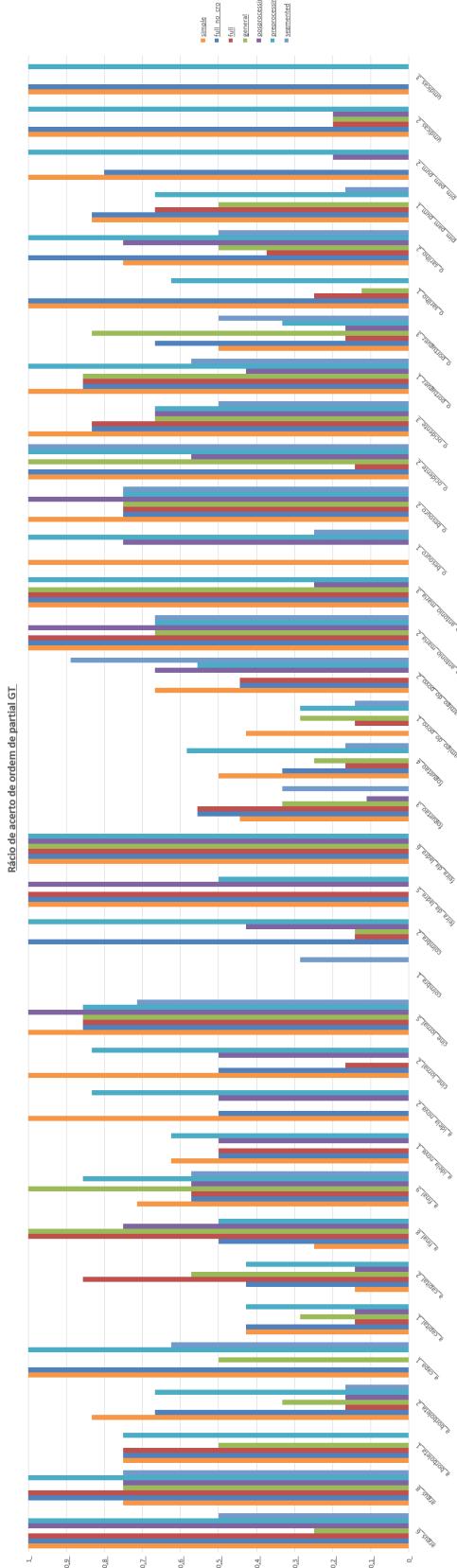


Figura 94: Ráios de acerto da ordem das linhas da Partial GT das diferentes pipelines (alargado).

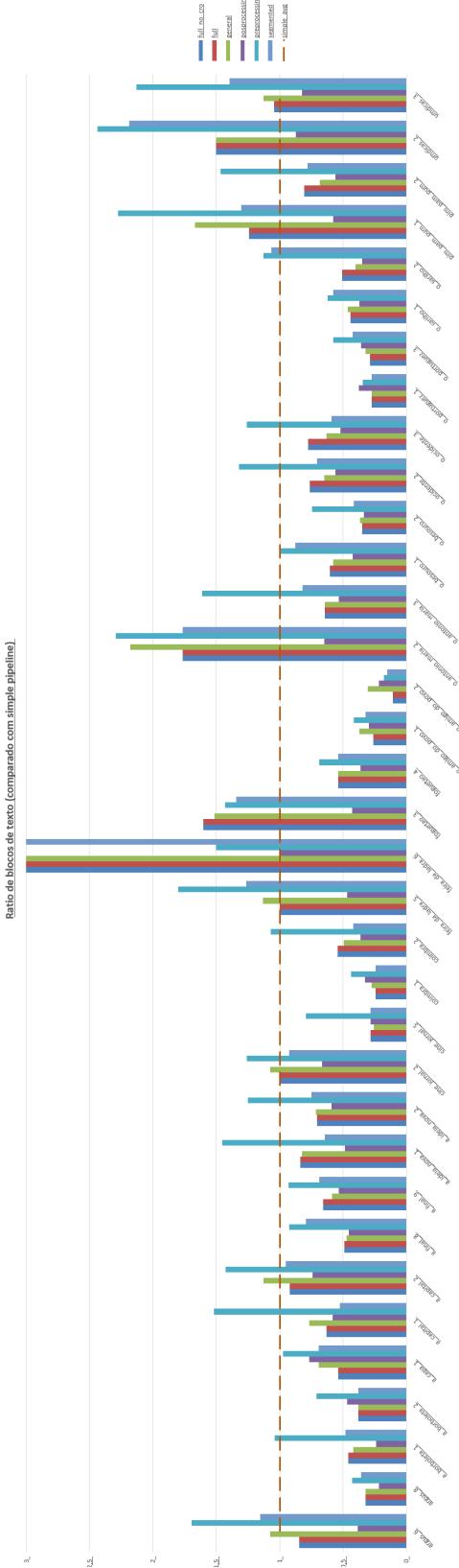


Figura 95: Ráios de número de blocos de texto relativo à pipeline simples (alargado).

## **Apêndice B**

## **Ferramentas**

- [Tesseract](#) : motor OCR utilizado
- [PySimpleGui](#) : base para criação de interface OSDOCR Editor
- [Matplotlib](#) : base para canvas de interação de OSDOCR Editor
- [Modelos de upscaling de imagem - waifu2x](#)
- [Modelos de melhoria de iluminação de documentos - HVI-CIDNet](#)



