

Universidade Federal do ABC
Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas
Trabalho de Graduação em Engenharia de Informação

**Estudo e aplicação de redes neurais profundas
na solução de detecção e reconhecimento de
texto em cenas**

Matheus Milani

**Maio de 2022
Santo André - SP**

Matheus Milani

**Estudo e aplicação de redes neurais profundas na solução
de detecção e reconhecimento de texto em cenas**

Trabalho de Graduação apresentado para conclusão da Graduação em Engenharia de Informação, como parte dos requisitos necessários para a obtenção do Título de Bacharel em Engenharia de Informação.

Universidade Federal do ABC
Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas
Trabalho de Graduação em Engenharia de Informação

Orientador: Murilo Bellezoni Loiola

Santo André - SP
Maio de 2022

Resumo

Segundo a ABNT, o resumo deve ressaltar o objetivo, o método, os resultados e as conclusões do documento. A ordem e a extensão destes itens dependem do tipo de resumo (informativo ou indicativo) e do tratamento que cada item recebe no documento original. O resumo deve ser precedido da referência do documento, com exceção do resumo inserido no próprio documento. Umas 10 linhas (...) As palavras-chave devem figurar logo abaixo do resumo, antecedidas da expressão Palavras-chave:, separadas entre si por ponto e finalizadas também por ponto.

Palavras-chaves: latex. abntex. editoração de texto.

Abstract

This is the english abstract.

Keywords: latex, abntex, text editoration.

Listas de ilustrações

Figura 1 – Ilustração das etapas de geração dos arquivos de ground truth para a etapa de treino do CRAFT. Fonte [1]	8
Figura 2 – Exemplificação do passo a passo para geração de anotações a nível de caracteres durante a etapa de treino do CRAFT. Fonte [1].	9
Figura 3 – Ilustração do pipeline de reconhecimento do CRNN. Fonte [2].	10
Figura 4 – Exemplo das imagens geradas a partir da detecção pelo CRAFT sobre uma imagem autoral. Fonte Própria	16
Figura 5 – Ilustração das etapas de processamento da solução proposta. Fonte Própria	19
Figura 6 – Exemplos de imagens do dataset ICDAR 2011. Fonte [3]	20
Figura 7 – Exemplos de imagens do dataset ICDAR 2013. Fonte [4]	21
Figura 8 – Interface de validação disponibilizada pela plataforma de desafios Robust Reading Competition. Fonte Própria	23
Figura 9 – Interface de validação disponibilizada pela plataforma de desafios Robust Reading Competition, com visualização de detalhes da avaliação por imagem. Fonte Própria	23
Figura 10 – Exemplo de resultado de reconhecimento. Imagem 52 do conjunto de validação.	28
Figura 11 – Comparação entre entrada e saída sobre a imagem 5 do set de validação do ICDAR 2011	29
Figura 12 – Exemplo de imagem com instâncias de texto bem pequenas em resolução. Imagem 28 do ICDAR 2011	30
Figura 13 – Demonstração de um falso positivo durante a avaliação da solução contra o dataset ICDAR 2013	31
Figura 14 – Exemplo de imagem com artefatos que trouxeram dificuldades para a localização correta do texto.	32
Figura 15 – Exemplo de texto sob vidro, com plano de fundo desafiadores para o reconhecimento.	32
Figura 16 – Exemplo de imagem com texto em desfoque	32
Figura 17 – Exemplo de imagem autoral onde a solução apresentou dificuldades com texto curvo e estilizado. Textos reconhecidos: “sney” e “intmalakingon”.	33
Figura 18 – Exemplo de imagem autoral onde uma região de texto muito longa não foi extraída muito bem, principalmente no início da palavra, o que dificultou o reconhecimento. Texto reconhecido: “permmusem”.	33

Figura 19 – Exemplo de imagem autoral com fontes bastante estilizadas que trouxe- ram dificuldade tanto para a detecção, quanto para o reconhecimento. Textos reconhecidos: “sey” e “fpan”	33
Figura 20 – Exemplo de reconhecimento com sucesso em condições desafiadoras em imagem autoral. Textos reconhecidos: “disneys”, “electrical”e “parade”.	34
Figura 21 – Exemplo de imagem autoral com alta precisão de reconhecimento. Tex- tos reconhecidos: “nelson”, “rolihlahla” “mandela”, “1918”, “2013”, “hamba”, “kahle”, “madiba”, “ve”, “honour”, “your”, “legacy”, “apartheid”, “museum”	34

Lista de tabelas

Tabela 1 – Avaliação de resultados sobre a base ICDAR 2011.	28
Tabela 2 – Avaliação de resultados sobre a base ICDAR 2013.	31

Lista de abreviaturas e siglas

ABNT Associação Brasileira de Normas Técnicas

abnTeX Normas para TeX

List of symbols

Γ Greek letter Gamma

Λ Lambda

ζ Greek letter minuscule zeta

\in Pertains

Sumário

1	INTRODUÇÃO	1
1.1	Scene Text Recognition	2
1.2	Objetivo	2
2	FUNDAMENTAÇÃO TEÓRICA	5
2.1	Aprendizado de Máquina e <i>Deep Learning</i>	5
2.1.1	Linguagens de Programação e <i>Frameworks</i>	6
2.2	Evolução do <i>Scene Text Recognition</i>	6
2.2.1	Detecção de Texto	6
2.2.1.1	CRAFT	7
2.2.2	Reconhecimento de Texto	9
2.2.2.1	CRNN	9
2.3	Considerações Finais	11
3	METODOLOGIA	13
3.1	Experimentação com soluções de reconhecimento e detecção de texto	13
3.2	Desenvolvimento da solução fim-a-fim a partir de detectores e reconhecedores de texto	16
3.3	Método de avaliação da solução proposta	19
3.3.1	Datasets	20
3.3.1.1	ICDAR 2011	20
3.3.1.2	ICDAR 2013	20
3.3.2	Métricas	21
3.3.3	Interface de Validação	22
3.4	Disponibilização da solução fim-a-fim em nuvem	23
3.4.1	Criação da Aplicação Web	24
3.5	Contêiner, Docker e Publicação	25
3.6	Considerações finais	25
4	RESULTADOS E DISCUSSÃO	27
4.1	Detecção e Reconhecimento	27
4.1.1	ICDAR 2011	27
4.2	ICDAR 2013	30
4.2.1	Imagens autorais	32
4.3	Disponibilização em nuvem	34

5	CONCLUSÃO	37
	REFERÊNCIAS	39
	APÊNDICES	41
	APÊNDICE A – ARQUIVO DE DESCRIÇÃO DO AMBIENTE VIRTUAL CONDA UTILIZADO NO DESENVOLVIMENTO	43
	APÊNDICE B – ARQUIVO DE DESCRIÇÃO DO AMBIENTE VIRTUAL CONDA UTILIZADO DURANTE A VALIDAÇÃO	47
	APÊNDICE C – ARQUIVO DOCKERFILE UTILIZADO NA PUBLICAÇÃO DA APLICAÇÃO EM NUVEM	49
	APÊNDICE D – EXEMPLO DE ARQUIVO DE RESULTADO PARA AVALIAÇÃO	51

1 Introdução

Termos como inteligência artificial e aprendizado de máquina já são praticamente constantes do cotidiano humano, mesmo que muitas vezes não visível, já estão presentes em muitas aplicações que muitas vezes sequer imaginamos, desde a assistente virtual dos dispositivos móveis e eletrônicos, até sistemas capazes de embasar diagnósticos médicos, como por exemplo uma aplicação capaz de auxiliar em diagnósticos de COVID-19 durante a pandemia do vírus SARS-COV2 [5].

Um escopo de aplicação de técnicas de aprendizado de máquina que tem evoluído bastante com o crescimento da popularidade e acessibilidade dos conceitos que envolvem *machine learning* são aplicações voltadas para reconhecimento de texto.

A escrita com certeza foi uma das grandes habilidades que a humanidade desenvolveu que mudou como relações e sociedades funcionavam, sendo adotada para transmissão e armazenagem de dados e informação, meio de comunicação e expressão.

Com os avanços da tecnologia e em especial, dos computadores, um grande desafio emergiu: Como fazer com que computadores entendam o que está escrito em documentos físicos? Umas das primeiras patentes para soluções de OCR (abreviação de *Optical Character Recognition*) data de 1929 [6], mas isso não nega o fato que a capacidade de transportar texto do meio físico para o meio digital de forma eficiente é um desafio interessante e que motiva pesquisas até hoje.

Em linhas gerais, o reconhecimento óptico de caracteres é uma ampla tarefa de reconhecimento de padrões e, para que máquinas consigam identificar os padrões presentes na instâncias de texto, elas precisam conhecer ao menos algumas características dos caracteres e do texto que serão reconhecidos. Para exemplificar, a Reading Machine de Tauschek [6] era uma aparelho mecânico que utilizava um disco de comparação, que continha o gabarito de cada um dos caracteres do alfabeto suportado. Esse foi o meio de "ensinar" a máquina a reconhecer um dado carácter.

Muitos anos depois, soluções ainda tem a missão de "treinar" computadores a identificar as características de caracteres e de textos como um todo e a principal ferramenta utilizada nos dias atuais são métodos sob o domínio de aprendizado de máquina, justamente pela capacidade de predição desses algoritmos dado um processo de treinamento. Ao longo deste trabalho de graduação outros conceitos que circundam o tópico de aprendizado de máquina serão introduzidos com um pouco mais de profundidade.

1.1 Scene Text Recognition

Um sub-conjunto de casos do espaço de aplicações de reconhecimento óptico de caracteres ganhou bastante tração no ultima década, impulsionada pelo alto poder computacional dos dispositivos modernos, acelerados por unidades gráficas, e a acessibilidade ao desenvolvimento de soluções de aprendizado de máquina. Esse sub-conjunto é conhecido como STR (abreviação de *Scene Text Recognition*, em inglês). Uma analogia para o STR seria aplicar soluções de OCR diretamente de fotos capturadas por uma câmera de um dispositivo móvel.

Como o nome sugere, STR classifica no sub-conjunto onde o problema a ser resolvido é a detecção e o reconhecimento do texto em imagens cotidianas, em cenas. A diferença entre um problema de STR comparado ao caso mais comum de OCR é, em termos simples, a aparência do texto e como ele será observado. Em uma imagem de cena, como por exemplo uma imagem da faixada de um supermercado, podemos ter textos em diferentes tamanhos, com diversas fontes, cores e orientações. Adicionalmente, por estarem muitas vezes sob influência do ambiente onde estão inseridos, outros fatores influenciam a observação desse texto, como iluminação, oclusão, danos devido ao clima, etc.

As soluções para problemas de STR, para lidarem com o nível de generalização necessário para reconhecer texto nos mais diversos casos, são largamente baseadas nos conceitos de deep learning, que demonstram ser capazes de ir um passo à frente no quesito reconhecimento de padrões em comparação às técnicas clássicas de processamento de imagem e conseguirem ser aplicadas com eficiência sobre imagens, sendo a aplicação das redes neurais convolucionais, comumente abreviadas para CNN (*Convolutional Neural Networks*, em inglês), um divisor de águas na evolução dessas soluções. [7, 8].

Assim, dada a importância do deep-learning, e em especial das CNNs nesse contexto de detecção e reconhecimento de texto, a próxima seção irá apresentar alguns conceitos básicos associados a essas estruturas.

1.2 Objetivo

Nulla malesuada risus ut urna. Aenean pretium velit sit amet metus. Duis iaculis. In hac habitasse platea dictumst. Nullam molestie turpis eget nisl. Duis a massa id pede dapibus ultricies. Sed eu leo. In at mauris sit amet tortor bibendum varius. Phasellus justo risus, posuere in, sagittis ac, varius vel, tortor. Quisque id enim. Phasellus consequat, libero pretium nonummy fringilla, tortor lacus vestibulum nunc, ut rhoncus ligula neque id justo. Nullam accumsan euismod nunc. Proin vitae ipsum ac metus dictum tempus. Nam ut wisi. Quisque tortor felis, interdum ac, sodales a, semper a, sem. Curabitur in velit sit amet dui tristique sodales. Vivamus mauris pede, lacinia eget, pellentesque quis,

scelerisque eu, est. Aliquam risus. Quisque bibendum pede eu dolor.

2 Fundamentação Teórica

Maecenas accumsan dapibus sapien. Duis pretium iaculis arcu. Curabitur ut lacus. Aliquam vulputate. Suspendisse ut purus sed sem tempor rhoncus. Ut quam dui, fringilla at, dictum eget, ultricies quis, quam. Etiam sem est, pharetra non, vulputate in, pretium at, ipsum. Nunc semper sagittis orci. Sed scelerisque suscipit diam. Ut volutpat, dolor at ullamcorper tristique, eros purus mollis quam, sit amet ornare ante nunc et enim.

Phasellus fringilla, metus id feugiat consectetur, lacus wisi ultrices tellus, quis lobortis nibh lorem quis tortor. Donec egestas ornare nulla. Mauris mi tellus, porta faucibus, dictum vel, nonummy in, est. Aliquam erat volutpat. In tellus magna, porttitor lacinia, molestie vitae, pellentesque eu, justo. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Sed orci nibh, scelerisque sit amet, suscipit sed, placerat vel, diam. Vestibulum nonummy vulputate orci. Donec et velit ac arcu interdum semper. Morbi pede orci, cursus ac, elementum non, vehicula ut, lacus. Cras volutpat. Nam vel wisi quis libero venenatis placerat. Aenean sed odio. Quisque posuere purus ac orci. Vivamus odio. Vivamus varius, nulla sit amet semper viverra, odio mauris consequat lacus, at vestibulum neque arcu eu tortor. Donec iaculis tincidunt tellus. Aliquam erat volutpat. Curabitur magna lorem, dignissim volutpat, viverra et, adipiscing nec, dolor. Praesent lacus mauris, dapibus vitae, sollicitudin sit amet, nonummy eget, ligula.

Cras egestas ipsum a nisl. Vivamus varius dolor ut dolor. Fusce vel enim. Pellentesque accumsan ligula et eros. Cras id lacus non tortor facilisis facilisis. Etiam nisl elit, cursus sed, fringilla in, congue nec, urna. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Integer at turpis. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Duis fringilla, ligula sed porta fringilla, ligula wisi commodo felis, ut adipiscing felis dui in enim. Suspendisse malesuada ultrices ante. Pellentesque scelerisque augue sit amet urna. Nulla volutpat aliquet tortor. Cras aliquam, tellus at aliquet pellentesque, justo sapien commodo leo, id rhoncus sapien quam at erat. Nulla commodo, wisi eget sollicitudin pretium, orci orci aliquam orci, ut cursus turpis justo et lacus. Nulla vel tortor. Quisque erat elit, viverra sit amet, sagittis eget, porta sit amet, lacus.

2.1 Aprendizado de Máquina e *Deep Learning*

Cras egestas ipsum a nisl. Vivamus varius dolor ut dolor. Fusce vel enim. Pellentesque accumsan ligula et eros. Cras id lacus non tortor facilisis facilisis. Etiam nisl elit, cursus sed, fringilla in, congue nec, urna. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Integer at turpis. Cum sociis natoque penatibus

et magnis dis parturient montes, nascetur ridiculus mus. Duis fringilla, ligula sed porta fringilla, ligula wisi commodo felis, ut adipiscing felis dui in enim. Suspendisse malesuada ultrices ante. Pellentesque scelerisque augue sit amet urna. Nulla volutpat aliquet tortor. Cras aliquam, tellus at aliquet pellentesque, justo sapien commodo leo, id rhoncus sapien quam at erat. Nulla commodo, wisi eget sollicitudin pretium, orci orci aliquam orci, ut cursus turpis justo et lacus. Nulla vel tortor. Quisque erat elit, viverra sit amet, sagittis eget, porta sit amet, lacus.

2.1.1 Linguagens de Programação e *Frameworks*

Cras egestas ipsum a nisl. Vivamus varius dolor ut dolor. Fusce vel enim. Pellentesque accumsan ligula et eros. Cras id lacus non tortor facilisis facilisis. Etiam nisl elit, cursus sed, fringilla in, congue nec, urna. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Integer at turpis. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Duis fringilla, ligula sed porta fringilla, ligula wisi commodo felis, ut adipiscing felis dui in enim. Suspendisse malesuada ultrices ante. Pellentesque scelerisque augue sit amet urna. Nulla volutpat aliquet tortor. Cras aliquam, tellus at aliquet pellentesque, justo sapien commodo leo, id rhoncus sapien quam at erat. Nulla commodo, wisi eget sollicitudin pretium, orci orci aliquam orci, ut cursus turpis justo et lacus. Nulla vel tortor. Quisque erat elit, viverra sit amet, sagittis eget, porta sit amet, lacus.

2.2 Evolução do *Scene Text Recognition*

Cras egestas ipsum a nisl. Vivamus varius dolor ut dolor. Fusce vel enim. Pellentesque accumsan ligula et eros. Cras id lacus non tortor facilisis facilisis. Etiam nisl elit, cursus sed, fringilla in, congue nec, urna. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Integer at turpis. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Duis fringilla, ligula sed porta fringilla, ligula wisi commodo felis, ut adipiscing felis dui in enim. Suspendisse malesuada ultrices ante. Pellentesque scelerisque augue sit amet urna. Nulla volutpat aliquet tortor. Cras aliquam, tellus at aliquet pellentesque, justo sapien commodo leo, id rhoncus sapien quam at erat. Nulla commodo, wisi eget sollicitudin pretium, orci orci aliquam orci, ut cursus turpis justo et lacus. Nulla vel tortor. Quisque erat elit, viverra sit amet, sagittis eget, porta sit amet, lacus.

2.2.1 Detecção de Texto

Vivamus eu tellus sed tellus consequat suscipit. Nam orci orci, malesuada id, gravida nec, ultricies vitae, erat. Donec risus turpis, luctus sit amet, interdum quis, porta sed,

ipsum. Suspendisse condimentum, tortor at egestas posuere, neque metus tempor orci, et tincidunt urna nunc a purus. Sed facilisis blandit tellus. Nunc risus sem, suscipit nec, eleifend quis, cursus quis, libero. Curabitur et dolor. Sed vitae sem. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Maecenas ante. Duis ullamcorper enim. Donec tristique enim eu leo. Nullam molestie elit eu dolor. Nullam bibendum, turpis vitae tristique gravida, quam sapien tempor lectus, quis pretium tellus purus ac quam. Nulla facilisi.

2.2.1.1 CRAFT

Character Region Awareness for Text Detection [1], ou simplesmente CRAFT, é um método de detecção publicado por Youngmin Baek et al. (2019), integrantes do time de pesquisa da empresa coreana Naver Corporation^[9], que apresentou resultados bastante competitivos quando comparado aos resultados estado-da-arte do momento, superando as melhores soluções do momento em acurácia de detecção com desempenho, capacidade de detecção em frames por segundo, competitiva com os melhores métodos já publicados.

Youngmin Baek et al. introduz um método de detecção a nível de caractere onde um modelo de rede convolucional FCN é criado a partir da reconhecida rede de extração de feature VGG-16^[10]. A arquitetura da rede CRAFT se inspirou na rede U-Net^[11] ao introduzir skip-connections, agregando características de alto e baixo nível entre os blocos de upsampling, que decodificam o mapa de predição em dois resultados ao final da rede:

- Mapa de predição de região de carácter (*Character Region Score*): probabilidades de cada pixel está localizado no centro de um carácter
- Mapa de predição de afinidade entre caracteres (*Character Afinity Score*): Probabilidades de cada pixel está localizado no centro da região entre caracteres

Como o resultado da rede é bem específico, as imagens de *ground truth* são geradas a partir de processamento de imagens. A partir das *bounding boxes* de cada caractere, um *heatmap* de uma distribuição gaussiana é projetada dentro de cada região de caractere, representando uma distribuição de probabilidade onde o centro da distribuição é o centro da região de caractere. Com isso tem-se o gabarito do primeiro resultado da rede. O *ground truth* para as regiões de afinidade envolve novamente projetar um heatmap gaussiano em uma região que, agora, é calculada em tempo de execução a partir dos *bounding boxes* de cada caractere. A Fig. 1 ilustra o método utilizado, que se baseia em calcular uma região retangular entre caracteres utilizando os centroides dos caracteres vizinhos e centroides de triângulos gerados em cada caractere.

Para treinar a rede, os autores se utilizaram de uma estratégia de aprendizado levemente supervisionado. Como as principais bases de imagens para treinamento não

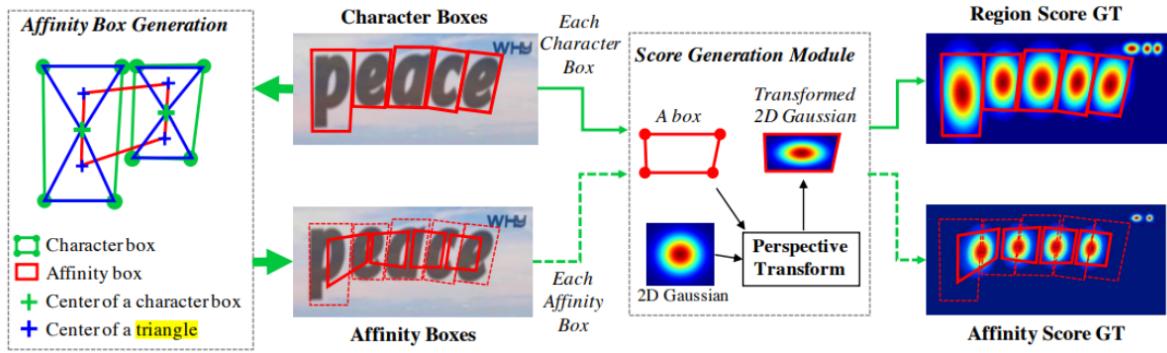


Figura 1 – Ilustração das etapas de geração dos arquivos de ground truth para a etapa de treinamento do CRAFT. Fonte [1]

contam com anotações a nível de caracteres, a rede primariamente é treinada com imagens com texto sintético, usando o dataset SynthText [12].

Para refinar o treino em datasets com imagens de cenas reais, os autores utilizam a própria rede treinada em texto sintético para predizer as regiões de caracteres das imagens de cena para gerar anotações a nível de caracteres para as imagens dos datasets utilizados com auxílio de métodos de processamento de imagem, conforme exemplificado na Fig. 2. O processo contém as seguintes etapas:

- *Cropping:* Extração das palavras que possuem região descrita nos arquivos de *ground truth* dos datasets de benchmark.
- *Character Split:* Processo de localização e segregação de cada caractere detectado pela rede treinada em base de dados sintética. A rede a partir das imagens provenientes da etapa de *Cropping*, predizendo as regiões onde a probabilidade de existir um caractere. Com a localização dessas regiões, é aplicado o algoritmo de segmentação conhecido como Watershed [13], cujo objetivo é expandir a área de um caractere a partir do centro da região de maior probabilidade até que as áreas de caracteres adjacentes se encontrem. Isso faz com que seja possível ajustar um bounding-box em volta de cada caractere observado.
- *Unwarping:* Uma vez em posse da capacidade de localizar todos os caracteres, obtida através da etapa de *Character Split*, pode-se projetar as coordenadas para as *bounding-boxes* de cada caractere de volta para a imagem original aplicando as operações inversas às aplicadas na etapa de *Cropping*.

Com essas labels geradas sobre as imagens reais dos datasets de benchmark, os gabaritos para o treinamento do modelo completo são gerados conforme explicado anteriormente e o treinamento da rede é refinado com esses novos exemplos.

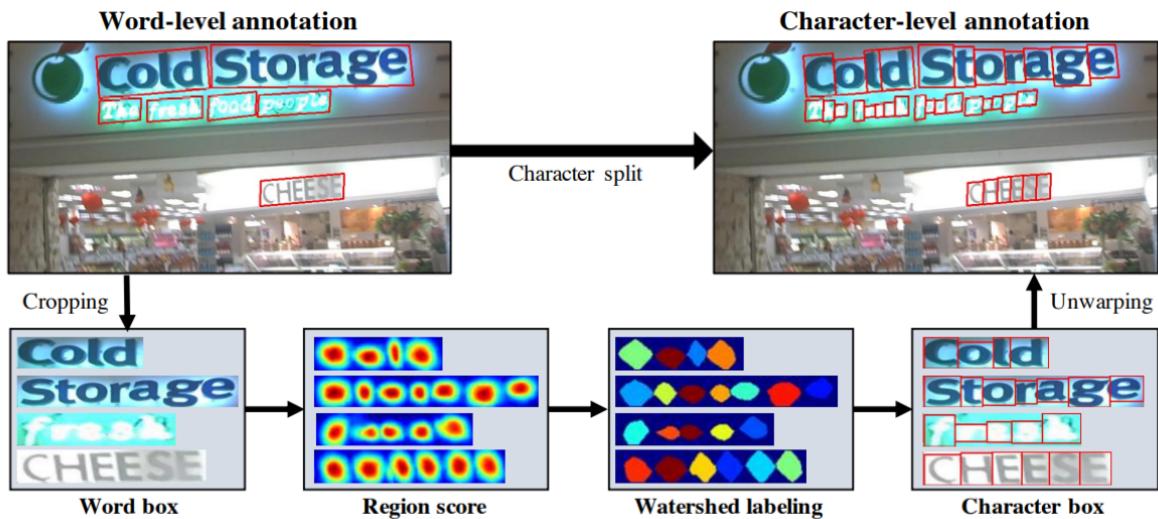


Figura 2 – Exemplificação do passo a passo para geração de anotações a nível de caracteres durante a etapa de treino do CRAFT. Fonte [1].

O CRAFT conta com um pós-processamento bastante simplificado em cima dos mapas de probabilidade que são gerados pela rede com o intuito de calcular os bounding boxes do texto localizado, que envolve, novamente com auxílio de métodos de visão computacional e processamento de imagem. Usando binarização e categorização, é possível unir as regiões de caracteres e de afinidade para extrair as coordenadas dos menores retângulos que encapsulam o resultado dessa união.

2.2.2 Reconhecimento de Texto

Vivamus eu tellus sed tellus consequat suscipit. Nam orci orci, malesuada id, gravida nec, ultricies vitae, erat. Donec risus turpis, luctus sit amet, interdum quis, porta sed, ipsum. Suspendisse condimentum, tortor at egestas posuere, neque metus tempor orci, et tincidunt urna nunc a purus. Sed facilisis blandit tellus. Nunc risus sem, suscipit nec, eleifend quis, cursus quis, libero. Curabitur et dolor. Sed vitae sem. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Maecenas ante. Duis ullamcorper enim. Donec tristique enim eu leo. Nullam molestie elit eu dolor. Nullam bibendum, turpis vitae tristique gravida, quam sapien tempor lectus, quis pretium tellus purus ac quam. Nulla facilisi.

2.2.2.1 CRNN

Convolutional Recurrent Neural Networks [2], introduzido por Baoguang Shi et al. é uma solução para o problema de reconhecimento de texto bastante popular, sendo bastante citada em novos trabalhos e sempre presente em trabalhos comparativos. Este método veio para resolver grandes dificuldade das soluções anteriores, por exemplo: lidar com entradas e saídas de comprimentos variados, possibilitar o aprendizado conhecido

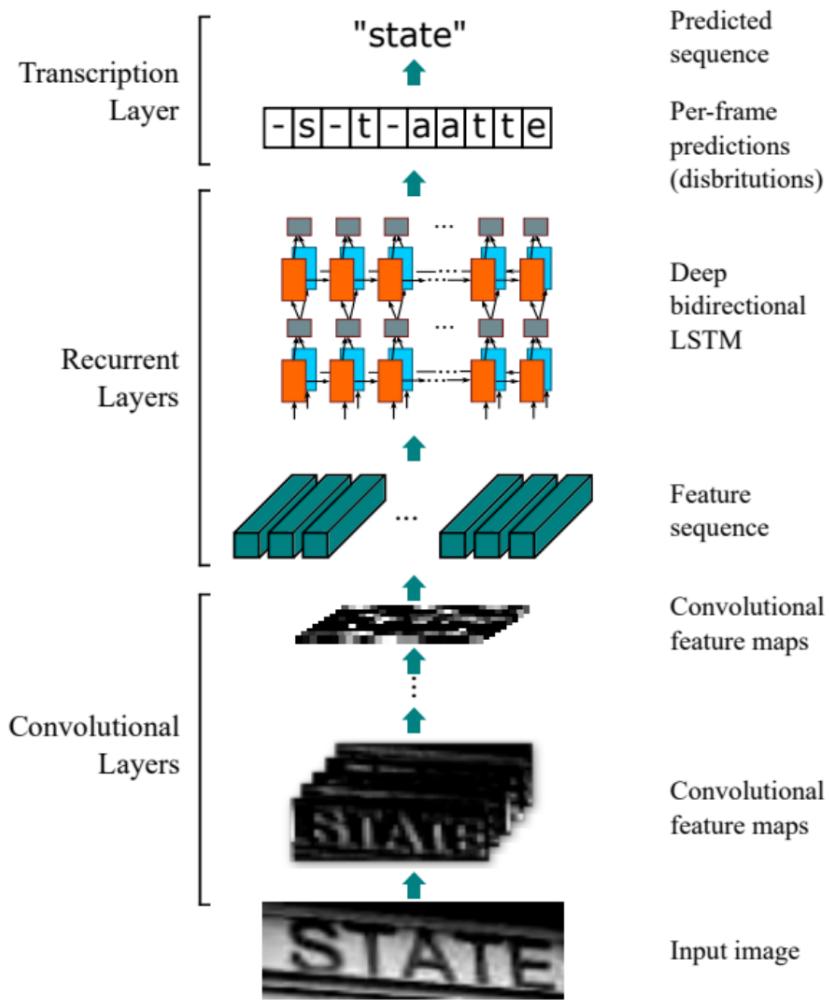


Figura 3 – Ilustração do pipeline de reconhecimento do CRNN. Fonte [2].

como end-to-end, isto é, aplicar uma função de perda sobre o resultado do reconhecimento e essa perda ser propagada para o aprendizado da rede extratora de características.

O CRNN, como o nome sugere, une uma rede neural convolucional, sem as camadas de predição totalmente conectadas, para gerar os feature maps sobre a imagem de entrada. Esses mapas são sequenciados para que alimentem a rede neural recorrente (RNN), que é responsável por conseguir decodificar cada sequência em um possível caractere. A Fig. 3 ilustra o pipeline de processamento que essa arquitetura executa.

A rede RNN que é implementada no CRNN faz uso de atributos LSTM (*Long-Short Term Memory*) bi-direcional, pois por estar visando reconhecimento de texto em cenas, muitas vezes carregar o contexto de sequências passadas e futuras é importante para diferenciar caracteres ou possibilitar o reconhecimento de um caractere, por exemplo, uma letra um pouco mais larga.

Os resultados obtidos foram bastante competitivos quando comparados aos métodos de reconhecimento anteriores, até mesmo superiores aos que já faziam uso dos conceitos

de deep learning, com o adicional de prover meios de aprendizado integrado da rede convolucional e recorrente como uma unidade, além de um modelo bem mais enxuto em número de parâmetros^[2].

Por lidar com o reconhecimento como um problema de sequência, o modelo pressupõe que a orientação do texto é necessariamente da esquerda para a direita, isso leva a uma limitação, que seria reconhecer textos com não exatamente horizontais e retilíneos.

2.3 Considerações Finais

Donec et nisl id sapien blandit mattis. Aenean dictum odio sit amet risus. Morbi purus. Nulla a est sit amet purus venenatis iaculis. Vivamus viverra purus vel magna. Donec in justo sed odio malesuada dapibus. Nunc ultrices aliquam nunc. Vivamus facilisis pellentesque velit. Nulla nunc velit, vulputate dapibus, vulputate id, mattis ac, justo. Nam mattis elit dapibus purus. Quisque enim risus, congue non, elementum ut, mattis quis, sem. Quisque elit.

3 Metodologia

Uma vez introduzida a temática deste trabalho graduação, seu objetivo e os principais conceitos que o circundam, é possível agora abordar as etapas percorridas para executá-lo e, eventualmente, possibilitar que outros, iniciantes no estudo de aplicações de detecção e reconhecimento de texto, consigam reproduzi-lo, provendo uma documentação detalhada do caminho tomado durante o desenvolvimento do trabalho.

Sob ótica de todo o ciclo de desenvolvimento deste trabalho de graduação, houveram quatro grande marcos que servirão para estruturar esse capítulo:

1. Experimentação com soluções de reconhecimento e detecção de texto.
2. Desenvolvimento da solução fim-a-fim a partir de detectores e reconhecedores de texto.
3. Avaliação da solução fim-a-fim proposta.
4. Publicação da solução em ambientes de computação em nuvem.

Dessa forma, as sub-seções a seguir detalharão cada uma dessas etapas, sob perspectiva da experiência do autor durante a execução deste trabalho de graduação.

3.1 Experimentação com soluções de reconhecimento e detecção de texto

Apesar de já ter sido mencionado a escolha do componente de detecção, CRAFT, e de reconhecimento, CRNN, não foi abordado o processo adotado que culminou nessa escolha. Nessa seção será abordado um pouco da motivação sobre essas escolhas, sob caráter mais técnico e prático sobre as soluções disponíveis publicamente, que são de fácil acesso através da internet.

A partir de uma rápida busca na internet, em especial na plataforma do GitHub, hospedeiro de repositórios de código fonte versionados pela ferramenta Git, famoso sistema controlador de versão, é possível encontrar diversos repositórios mencionando soluções para detecção e reconhecimento de texto em cenas, às vezes até mesmo repositórios oficiais, oferecidos pelos autores de trabalhos famosos na área, como é o próprio exemplo do detector de texto CRAFT.

No entanto, uma grande dificuldade para de fato conseguir executar o código desses repositórios é satisfazer os requisitos e dependências mínimas, muitas vezes implícitos, por

exemplo, a obrigatoriedade de possuir hardware com placa gráfica compatível com a versão utilizada no momento de desenvolvimento do trabalho original, a indisponibilidade dos modelos pré-treinados para rápida verificação de resultados, a base de código incompleta, sem todas as instruções para execução ou com dependências em versões antigas da linguagem de programação, bibliotecas e/ou *frameworks*, entre outros.

Um outro ponto de grande valor na escolha dos métodos aplicados neste trabalho é a compatibilidade entre os métodos de detecção e reconhecimento. Como o grande objetivo deste trabalho se baseia na integração dessas soluções para que trabalhem juntas, o caminho mais simples nessa linha é que ambos projetos consigam se executados dentro do mesmo ambiente de desenvolvimento, com dependências em bibliotecas e frameworks parecidos, se não, iguais, para que não houvessem conflitos entre ambos os projetos que trouxessem problemas em tempo de execução.

Na escolha do método de detecção de texto, a partir de uma revisão de trabalhos anteriores da área, foi considerado e, até certo ponto, experimentado, códigos oficiais e re-implementações públicas dos seguintes trabalhos:

TODO: listas trabalhos e repositórios e referencias

Por fim, o que foi mais acessível em relação aos pontos citados anteriormente foi o código-fonte do CRAFT. Os autores desenvolveram o modelo em Python, usando o framework PyTorch, que é bastante difundido na comunidade, o que facilita na escolha de projetos para integrar o reconhecimento de texto. Os autores também disponibilizaram os modelos pré-treinados para detecção de texto em cenas, o que é um grande diferencial, já que o procedimento de treinamento dessas soluções não são triviais, além de costumarem ser bastante custosos temporal e computacionalmente, o que desviaria o foco do objetivo desse trabalho. Adicionalmente, o código fonte contém instruções mínimas de como executar o modelo pré-treinado, que se mostraram suficientes para iniciantes sem muita experiência prévia com a linguagem de programação e em trabalhos de detecção de texto. Outro diferencial foi a possibilidade de executar o modelo pré-treinado em hardwares sem aceleração gráfica, mesmo que com a penalização no tempo de execução. Essa capacidade foi de grande valor, já que os componentes compatíveis não são baratos, dado o contexto atual de criptomoedas e escassez de chips gráficos no mercado.

Agora sobre a escolha do método de reconhecimento, como o detector escolhido usa a linguagem Python e o framework PyTorch, a lista de métodos disponíveis foi filtrada por esses critérios. Uma implementação do CRNN em PyTorch foi escolhida pois atendia os critérios de compatibilidade com a linguagem, framework e bibliotecas que o CRAFT depende, aliado ao fato do CRNN ser uma solução bastante popular, com implementação bastante sucinta e, novamente, de fácil execução a partir de um modelo pré-treinado, o que também foi fornecido pelo autor da implementação.

Durante todo o processo de experimentação e de desenvolvimento, a plataforma Paperspace foi de grande valor. A Paperspace é uma empresa provedora de computação em nuvem especializada em infraestrutura para aplicações de aprendizado de máquina, disponibilizando gratuitamente ambientes de Jupyter Notebooks em máquinas aceleração gráfica de forma gratuita para fins de estudo e experimentação, com características similares ao Google Colab.

Executar o CRAFT e o CRNN a partir dos respectivos repositórios originais não demandaram muitas configurações ou customizações específicas além da configuração correta do ambiente de desenvolvimento facilitado a ferramenta Conda, que será introduzida com mais detalhes na Seção 3.2. Ambos repositórios já continham *scripts* básicos para testar a predição a partir dos modelos pré-treinados, então, tanto para o CRAFT, quanto o CRNN, executar os modelos com imagens de testes já incluídas nos repositórios originais ou com outras imagens quaisquer demandava ajustes mínimos para ajustar as referencias para leitura da imagem de entrada e re-executar os *scripts* de predição.

Sobre as características das entradas e saídas dos modelos escolhidos, durante a detecção através do modelo CRAFT não existem restrições notáveis a respeito dos dados de entrada. São esperadas imagens, que podem conter tamanhos variados, e, contando que possam ser abertas pelo *script* de detecção, são passíveis de serem processadas pelo modelo. Como citado na Seção 2.2.1.1, a rede de detecção produz imagens que descrevem a localização de cada região de caractere e o nível de afinidade das vizinhanças dos caracteres detectados. O processamento dessas imagens brutas que a rede produz é disponibilizado pelos autores e faz parte do repositório original, portanto o código que é capaz de derivar as coordenadas de *bounding-boxes* para cada palavra já é disponibilizado pelos autores do CRAFT. A Figura 4 exemplifica os resultados de detecção originais do CRAFT e demonstram a capacidade de produzir tanto os arquivos brutos, que ilustram os resultados da rede de detecção, quanto uma visualização sobre a imagem de entrada, desenhandos os contornos retangulares em volta das palavras detectadas.

Agora sobre o modelo de reconhecimento, para as imagens de entrada, espera-se que contenham apenas uma palavra por imagem e devem respeitar limites de dimensão impostas pela rede que, na implementação utilizada, já são aplicadas durante a execução do *script* de predição, limitando a altura das imagens em 32 pixels. O resultado da predição é a impressão do texto reconhecido ao final da execução do programa.

A próxima sub-seção abordará, com um pouco mais de detalhes, como as duas soluções escolhidas foram integradas em uma mesma base de código e como os resultados do CRAFT foram processados para que pudessem alimentar a implementação do CRNN, completando a solução fim-a-fim do reconhecimento de texto.



Figura 4 – Exemplo das imagens geradas a partir da detecção pelo CRAFT sobre uma imagem autoral. Fonte Própria

3.2 Desenvolvimento da solução fim-a-fim a partir de detectores e reconhecedores de texto

A fim de aplicar o reconhecimento, utilizando o CRNN, exatamente sobre o texto detectado e localizado pelo CRAFT, a escolha foi utilizar o repositório com o código fonte do detector CRAFT como base inicial para a aplicação de reconhecimento fim-a-fim e, a partir dele, adicionar a solução de reconhecimento dentro da mesma base de código e fazer com que ambas tenham todas as dependências necessárias para conseguirem executar seu processamento. Em outras palavras, foi necessário preparar um ambiente de desenvolvimento Python com todas a dependências, a nível de linguagem de programação, bibliotecas e frameworks, instalados e sem conflitos, com tanto as bibliotecas que são dependências obrigatórias para a implementação do CRAFT quanto as bibliotecas que são utilizadas na implementação do CRNN.

Para a gestão de ambientes de desenvolvimento e suas dependências, foi utilizado uma ferramenta no ecossistema Python chamada Conda, mantida pela empresa Anaconda. Com essa ferramenta é possível criar ambientes virtuais Python e instalar todas as dependências necessárias dentro desses ambientes virtuais usando poucas instruções na linha de comando do computador. A própria ferramenta fornece meios para de gerir os

ambientes virtuais criados e otimiza a instalação de pacotes dentro desses ambientes de forma que conflitos sejam evitados, resolvendo as versões dos pacotes instalados de forma que sejam compatíveis entre si e compatíveis com a versão da linguagem Python escolhida para o ambiente. Uma grande vantagem de usar esse gerenciador de ambientes Python é a possibilidade de reproduzir qualquer ambiente virtual previamente configurado a partir de um arquivo de configuração, do tipo YAML, que lista todos os pacotes instalados em um ambiente virtual para que, a partir desse arquivo, a ferramenta consiga remontar o mesmo ambiente sem precisar de configurações manuais novamente.

Entrando mais no caminho percorrido para integrar as duas soluções, o primeiro passo foi replicar os repositórios para a conta pessoal do autor no GitHub. Esse processo dentro da plataforma se chama *fork* e em geral é bastante utilizado para possibilitar que outras pessoas consigam desenvolver coisas novas a partir da base de código original sem ter o risco de empurrar mudanças no repositório original inadvertidamente. Dessa forma, ambos os repositórios foram replicados para a conta pessoal do autor e todo o desenvolvimento deste trabalho foi feito nos repositórios replicados.

A partir dos repositórios replicados, foi utilizado um recurso da ferramenta de versionamento, o Git, que possibilita a composição de repositórios a fim de facilitar a sincronização das bases de códigos. O recurso em questão é a criação de sub-módulos em um repositório Git. Usando o comando abaixo a partir do repositório replicado do detector, foi gerado um relacionamento entre as duas bases de código, sendo que o repositório do CRNN agora é visto como um sub-módulo do repositório do CRAFT, e, a partir disso, é possível baixar, usar, evoluir a base de código do repositório do CRNN diretamente do repositório do CRAFT. Isso fez com que ambas soluções já tenham visibilidade uma da outra, mas ainda não fez com que trabalhem juntas.

```
git submodule add https://github.com/mmilani1/crnn.pytorch
```

Com as soluções na mesma base de código, uma etapa importante é a garantir que ambas ainda funcionam de forma isolada e se validar a compatibilidade das dependências de cada uma. Como mencionado na Seção 3.1, uma das motivações da escolha especificamente dessas duas implementações citadas foi justamente a maior proximidade em termos de frameworks e bibliotecas que utilizam, inclusive a possibilidade de ambas soluções poderem ser executadas sem a necessidade de aceleração gráfica. Dessa forma não houveram problemas em executar separadamente cada solução, mesmo compartilhando o mesmo ambiente virtual. O arquivo YAML com a descrição do ambiente virtual pode ser consultado diretamente do repositório com a implementação ou no Apêndice A.

A partir deste ponto já se torna possível trabalhar com os modelos de detecção e reconhecimento a fim de integrá-los, de alguma forma. O objetivo ao fim da integração é conseguir reconhecer texto em imagens de cena, para isso, a ideia é evoluir a extração de

resultados do modelo de detecção CRAFT para gerar uma lista de recortes das regiões de texto da imagem original e implementar um algoritmo que execute o modelo de reconhecimento CRNN para cada um desses recortes.

Conforme detalhado na Seção 3.1, a implementação do CRAFT fornecida é didática o suficiente com os resultados da detecção, gerando visualizações dos resultados junto com um arquivo de texto contendo as coordenadas das regiões de texto detectadas. Isso demonstra que, em algum ponto do código original as coordenadas dessas regiões de texto foram manuseadas pelos autores do modelo e, consequentemente, poderão ser manuseadas no desenvolvimento da solução integrada a fim de gerar os recortes que serão posteriormente consumidos pelo modelo reconhecimento. O Algoritmo 1 apresenta o pseudo-algoritmo com as etapas do processamento executado.

Algorithm 1 Pseudo-Código da integração entre a detecção e o reconhecimento de texto

```

 ← lerEntrada()
palavras ← []
regioesDeTexto ← executaDeteccaoCRAFT(imgem)
for regiaoDeTexto ∈ regioesDeTexto do
    recorte ← recortarTextoDaImagenOriginal(imgem, regiaoDeTexto)
    palavra ← executaReconhecimentoCRNN(recorte)
    palavras ← palavras + [palavra]
end for
return palavras
  
```

Para efetuar o recorte das regiões de texto localizadas e fornecidas na saída do modelo de detecção, foi utilizado um par de funções da biblioteca OpenCV, especializada em ferramentas voltadas para visão computacional. As funções utilizadas foram `cv2.getPerspectiveTransform` e `cv2.warpPerspective`. A primeira função é responsável por gerar uma matriz de transformação de áreas quadrangulares e recebe duas sequências de pontos. A primeira sequência contém os vértices da área quadrangular a ser transformada, no caso desse trabalho, são as coordenadas da *bounding-box* de uma palavra detectada. Já a segunda sequência de pontos representa os vértices da imagem de saída, para onde cada um dos pontos da primeira sequência serão transportados após a aplicação da transformação. A segunda função é responsável por aplicar a matriz de transformação, gerada a partir da execução do comando `cv2.getPerspectiveTransform`, sobre a imagem onde o texto foi detectado. Dessa forma conseguimos recortar as palavras detectadas da imagem original de forma cada é possível posteriormente passar as referências desses recortes para o modelo de reconhecimento diretamente, já que os dados de entrada esperados pelo CRNN são imagens que contém idealmente uma palavra para reconhecimento. A Figura 5 ilustra, em alto nível, as etapas do processamento que compõem as solução de STR fim-a-fim deste trabalho.

Ao fim dessas alterações, integrando os dois modelos de detecção e reconhecimento,

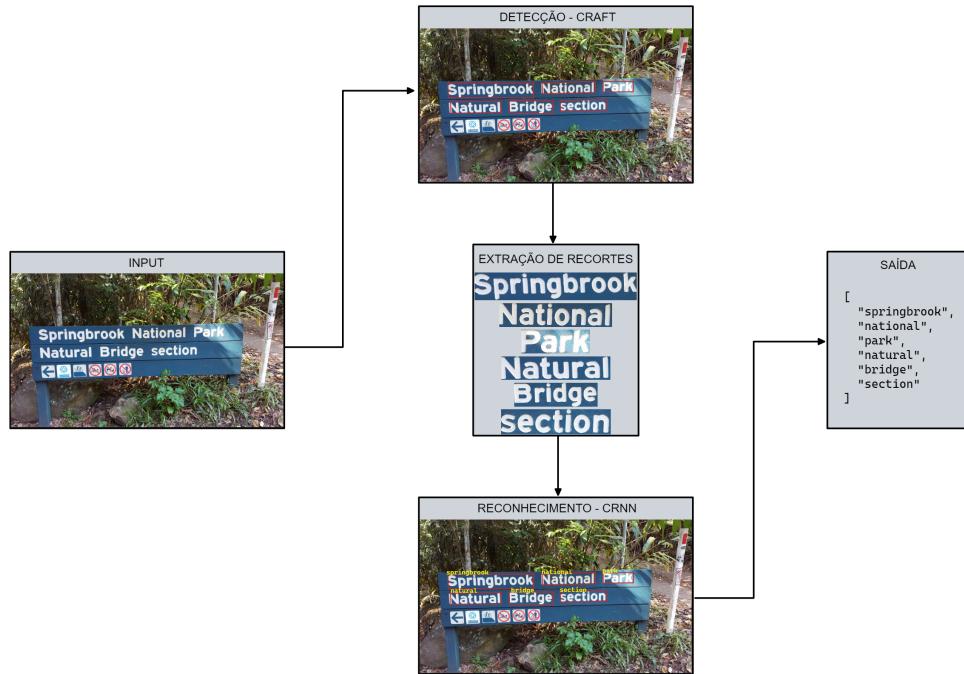


Figura 5 – Ilustração das etapas de processamento da solução proposta. Fonte Própria

já temos uma solução simples de reconhecimento de texto em cenas que pode ser classificada como *end-to-end*, que contempla tanto a detecção quanto o reconhecimento dentro de uma mesma aplicação. A próxima etapa é conseguir avaliar a eficácia desse novo desenvolvimento. A próxima seção introduzirá os métodos e métricas utilizadas para avaliar a solução.

3.3 Método de avaliação da solução proposta

A avaliação analítica da solução proposta neste trabalho é baseada na forma de avaliação da plataforma *Robust Reading Competition* (abreviação RRC), disponibilizado pelo Centro de Visão Computacional da Universidade Autônoma de Barcelona. Essa plataforma organiza competições em torno de problemas reais de visão computacional e tipicamente essas competições se relacionam com a Conferência Internacional sobre Análise e Reconhecimento de Documentos, abreviada como ICDAR, a partir no nome da conferência em inglês. Diversos datasets que são amplamente adotados para treino e avaliação de modelos de reconhecimento de texto levam no nome a menção ao ICDAR, por introduzirem a base de dados e os desafios propostos, fomentando novos trabalhos na área.

Essa seção irá introduzir dos datasets avaliados, as métricas utilizadas e o processo de avaliação da solução apresentada utilizando as ferramentas de avaliação disponibilizadas pela plataforma RRC.



Figura 6 – Exemplos de imagens do dataset ICDAR 2011. Fonte [3]

3.3.1 Datasets

Conforme mencionado anteriormente, os datasets dos desafios vinculados ao ICDAR são muito utilizados por trabalhos voltados à detecção e reconhecimento de texto. Ambos os modelos utilizados no desenvolvimento deste trabalho fazem referência aos datasets ICDAR: O CRAFT cita os datasets ICDAR 2013, ICDAR 2015 e ICDAR 2017 como bases de treinamento e de validação em quanto o CRNN se refere ao ICDAR 2003 e ICDAR 2013 como bases de imagens para validação. Os datasets utilizados para a avaliação deste trabalho foram o ICDAR 2011, base para o primeiro desafio da plataforma RRC chamado *Reading Text in Born-Digital Images*, e o ICDAR 2013, que é a base para o segundo desafio da plataforma RRC, chamado *Focused Scene Text*.

3.3.1.1 ICDAR 2011

O ICDAR 2011 conta com 410 imagens para treino e 141 imagens para validação, imagens que estão no contexto de anúncios de internet e anexos de e-mails, com palavras em geral na horizontal e anotadas com *bounding-boxes* retangulares a nível de palavras. As imagens são anotadas com a localização de cada palavra com coordenadas das *bounding-boxes* retangulares, que servem como gabarito para soluções de detecção e localização. Cada imagem também conta com a transição das palavras para servir como gabarito de soluções de reconhecimento conta A Figura 6 expõe algumas imagens deste dataset.

3.3.1.2 ICDAR 2013

O ICDAR 2013 conta com 219 imagens para treino e 233 imagens para validação . São imagens de cenas onde em geral o texto tem boa qualidade e está em destaque, na orientação horizontal. As anotações de gabarito são retangulares a nível de palavras e contam com a transição das mesmas, de forma similar às imagens do ICDAR 2011. A Figura 7 expõe algumas imagens deste dataset.



Figura 7 – Exemplos de imagens do dataset ICDAR 2013. Fonte [4]

3.3.2 Métricas

Em conjunto aos datasets, são fornecidas meios de avaliação para que cada nova solução que surgir possam ser avaliadas sob os mesmos termos, usando os mesmos conceitos e as mesmas métricas. Na plataforma RCC e nos datasets que eles disponibilizam são utilizados: Precisão, Recall e F1-Score (ou H-Mean). A Precisão é a relação dos acertos (verdadeiros positivos) sobre a contagem total de previsões positivas (verdadeiros positivos e falsos negativos), ou seja, dentre as previsões da solução, qual é a porcentagem de acerto. O Recall relaciona a quantidade de acertos com a quantidade total de casos verdadeiros. O F1-Score é basicamente a média harmônica das métricas Precisão e Recall.

$$\text{Precisão} = \frac{VP}{VP + FP} \quad (3.1)$$

$$\text{Recall} = \frac{VP}{VP + FN} \quad (3.2)$$

$$\text{F1Score} = 2 \times \frac{\text{Precisão} \times \text{Recall}}{\text{Precisão} + \text{Recall}} \quad (3.3)$$

Para determinar se uma previsão foi correta existem duas condições: A região de texto detectada deve satisfazer uma métrica característica de eficácia em detecção e localização das palavras na imagem chamada IoU, abreviação para *Intersection over Union*, além da transcrição predita ser exatamente igual ao gabarito. Para uma previsão ser considerada correta, o valor de IoU, que se resume à precisão da detecção, deve ser maior ou igual a 50%.

A métrica de IoU pode ser calculada a partir das coordenadas verdadeiras de uma região de texto, documentadas nas anotações dos datasets, e das coordenadas preditas pelo modelo de detecção. Ambos conjuntos de coordenadas descrevem uma área retangular, para os dois datasets utilizados. Com esses dois conjuntos de valores, pode-se avaliar tanto o tamanho da intersecção entre a área predita e a área esperada, quanto o tamanho da união entre ambas as áreas. O valor numérico da métrica IoU decorre, conforme sugerido pelo nome da métrica, da divisão entre o tamanho da intersecção sobre o tamanho da união entre as áreas preditas e as áreas esperadas.

3.3.3 Interface de Validação

Na plataforma do RRC é possível avaliar uma solução formalizando a submissão dos resultados diretamente no site da competição, dessa forma os resultados ficam registrados na plataforma e pode ser até ranqueada junto a outros trabalhos submetidos. Para cada desafio da plataforma existe o ranque de soluções, onde é pode-se observar resultados de outros trabalhos de maneira simples e, quando disponível, até consultar quem são os autores, se existe algum repositório publico para a base de código, visualizar o artigo do trabalho, etc.

No entanto não é a única opção. Existe a opção de efetuar o download dos *scripts* que calculam os resultados, nos mesmos moldes da avaliação remota, a fim de executar todo o processo localmente, que foi o método adotado para esse trabalho.

Para ambos os meios de validação, é necessário preparar arquivos de resultados, para cada imagem do dataset de validação, com os dados necessários para a avaliação. Tanto o ICDAR 2011 quanto o ICDAR 2013 demandam arquivos com o mesmo formato, que se assemelham justamente aos arquivos de gabarito dessas bases, e seguem o seguinte formato descrito abaixo. Um exemplo de arquivo de resultado foi disponibilizado no Apêndice D

- É necessário gerar um arquivo de texto, para cada imagem, nomeados respeitando o seguinte formato: `res_img_#.txt`, onde o `#` corresponde ao identificador numérico da imagem no dataset.
- Cada arquivo deve ter uma linha para cada palavra detectada e reconhecida.
- Cada linha deve conter os seguintes valores, separados por vírgula simples, exatamente na ordem especificada: posição do vértice esquerdo superior da bounding-box, vértice direto inferior da bounding-box e texto da transição.

Após gerar todos os arquivos necessários, resta agregá-los em um arquivo comprimido do tipo ZIP e, ou submeter a solução diretamente na plataforma da competição, ou executar a ferramenta de validação localmente. Os arquivos de execução local são escritos em Python e dependem de poucas bibliotecas. Executam um leve servidor web para servir a aplicação de validação. Durante a execução da validação deste trabalho, foi utilizado novamente a ferramenta Conda para gerir essas dependências. O Apêndice B contém o arquivo que descreve o ambiente virtual utilizado. A Figura 8 mostra um pouco da interface de usuário que a aplicação fornece, com as funcionalidades de submeter o arquivo comprimido de solução e a Figura 9 mostra a funcionalidade de verificação dos resultados sobre cada imagem, possibilitando a análise sobre onde as soluções funcionaram bem e onde não funcionaram tão bem, o que foi feito no contexto da solução proposta nesse trabalho no Capítulo 4.



Figura 8 – Interface de validação disponibilizada pela plataforma de desafios Robust Reading Competition. Fonte Própria

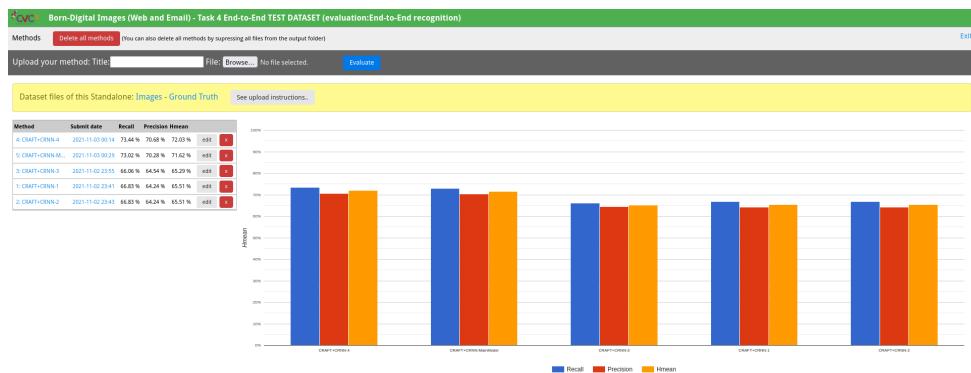


Figura 9 – Interface de validação disponibilizada pela plataforma de desafios Robust Reading Competition, com visualização de detalhes da avaliação por imagem. Fonte Própria

3.4 Disponibilização da solução fim-a-fim em nuvem

Uma vez com o desenvolvimento da integração entre os componentes concluída e avaliada, a próxima etapa para atingir os objetivos deste trabalho de graduação é executar uma prova de conceito sobre a solução proposta e disponibilizá-la em como uma aplicação web, hospedada em nuvem, assim possibilitando demonstrar que seria possível distribuir a funcionalidade para outros, possivelmente até evoluir para um serviço digital de reconhecimento de texto, vendido como produto.

Existem diversas provedoras de computação em nuvem que poderiam ser utilizadas, entre as mais notáveis estão a gigante AWS (Amazon Web Services) e a a própria Paperspace, utilizada sob-demanda por esse trabalho durante o desenvolvimento conforme citado na Seção 3.1. Ambas teriam infraestrutura para comportar a solução proposta por este trabalho, inclusive provendo máquinas com alto poder computacional dependendo da necessidade e dos limites financeiros do projeto. Como ambos serviços, apesar de possuírem limites de uso onde os recursos são gratuitos, não são de domínio do autor deste trabalho de graduação. Como uma terceira terceira opção, a plataforma escolhida para hospedar o desenvolvimento deste trabalho foi o Heroku. No entanto, mesmo tendo

escolhido uma plataforma específica para hospedar a aplicação, as ferramentas e o processo até a publicação em geral são parecidas, então, resguardadas as devidas proporções, o processo feito para publicar a aplicação no Heroku pode ser extrapolado para algo similar em outras provedores de infraestrutura.

O Heroku é uma plataforma disponibilizada pela empresa Salesforce que age como uma camada de abstração sobre os provedores de nuvem e seus componentes de mais baixo nível, justamente para prover uma interface mais simples, facilitando a gestão de infraestrutura para desenvolvedores. As etapas para publicar uma aplicação para o Heroku envolvem poucos comandos e também existe a possibilidade de hospedar aplicações usando recursos gratuitos. As limitações do ambiente gratuito e suas implicações na execução da aplicação serão abordadas no Capítulo 4.

3.4.1 Criação da Aplicação Web

Até então, tudo que foi desenvolvido para este trabalho foi executado interagindo diretamente com os componentes da base de código, executando os *scripts* de detecção e reconhecimento através de uma linha de comando. Para expor o modelo integrado para na internet, a ideia foi criar uma aplicação web, nos moldes de cliente-servidor, que responde requisições pelo protocolo HTTP, utilizando a arquitetura REST. Para isso foi necessário fazer algumas adições à base de código para contemplar a implementação de uma aplicação web utilizando o framework Flask, que permite, com pouquíssima configuração, executar um servidor HTTP e possibilita que uma requisição seja atendida por um método da base de código.

Como o código existente esperava ser executado através dos *scripts* diretamente, dependendo de parâmetros de linha de comando por exemplo, foi necessário a criação de um novo arquivo contendo uma classe Python que centralizasse a execução dos métodos necessários para o funcionamento do reconhecimento fim-a-fim. Essa classe foi denominada **OcrRunner** e centraliza toda a orquestração do processamento, desde a preparação dos modelos junto ao framework PyTorch, até a execução de fato da detecção e do reconhecimento de forma integrada. Com isso, foi possível executar a solução fim-a-fim programaticamente.

Além da nova classe implementada, foi necessário criar um novo método que tem a responsabilidade atender a requisição originada pelo cliente. Esse método, com a ajuda do framework Flask, consegue receber um arquivo de imagem enviado pelo cliente e executar a solução proposta sobre a imagem enviada. Por fim, quando o reconhecimento termina, o retorno do método é a lista de palavras reconhecidas, que, automaticamente, é respondido ao cliente que fez a requisição.

Com essas poucas mudanças já é possível, de maneira mínima e muito simplificada, atender pedidos de reconhecimento de texto sob demanda. O próximo passo adotado foi

utilizar ferramentas e conceitos voltados a conteinerização da aplicação para facilitar a execução no ambiente remoto.

3.5 Contêiner, Docker e Publicação

Uma prática muito usual ao utilizar a computação em nuvem como meio de hospedar aplicações é gerar arquivos executáveis contendo tudo o que a aplicação precisa para funcionar: A base de código, as dependências, os binários, as configurações e tudo mais que for necessário, de maneira consistente e de fácil reproduzibilidade. Esse processo é por vezes chamado de conteinerização, onde um contêiner é esse agregado do código da aplicação e tudo o que for necessário para executá-la, de fato sendo executado. Uma das tecnologias de contêiner mais utilizadas é o Docker [14]. Mais detalhes sobre Docker e contêineres podem ser encontrados na página oficial da tecnologia.

A fim de gerar um contêiner da aplicação desenvolvida neste trabalho, o primeiro passo foi descrever como esse contêiner é construído, e, para isso, cria-se um arquivo chamado de Dockerfile. Nele consta uma série de comandos que são executados rigorosamente pelo Docker para criar uma imagem Docker. Ao executar a imagem pelo Docker, um contêiner é inicializado com o conteúdo da imagem executada e pode executar o código que foi compilado na imagem Docker. O arquivo Dockerfile pode ser consultado diretamente no repositório desse trabalho ou no Apêndice C.

Ter a aplicação "conteinerizada" economiza muita configuração na hora de publicá-la na nuvem, já que não se faz mais necessário configurar o servidor onde a aplicação será servida de forma específica para satisfazer as demandas de binários e bibliotecas para a aplicação em questão. Tudo o que é necessário para executar a aplicação já está definido na imagem do contêiner e estará pronto pra uso quando o contêiner foi iniciado, dessa forma, as etapas mencionadas nas Seções 3.1 e 3.2 referentes a configuração do ambiente Python e instalação de dependências através do Conda já foi resolvido na criação da imagem e o ambiente já estará pronto para uso dentro do contêiner. O servidor que hospeda os contêineres não precisa saber de especificidades de cada aplicação, sua responsabilidade é saber executar os contêineres e compartilhar recursos computacionais com eles. Dessa forma, a publicação da aplicação no Heroku se resume a execução de um comando a interface de linha de comando que a plataforma disponibiliza para publicar uma aplicação a partir de uma imagem Docker.

3.6 Considerações finais

De forma sucinta, nesse capítulo foi apresentado como ocorreu o desenvolvimento deste trabalho de graduação, mencionando como os modelos de detecção e reconhecimento

foram escolhidos, como e com quais ferramentas eles foram testados, integrados e, ao final da integração como a solução foi validada e publicada em nuvem.

O próximo capítulo abordará mais a fundo os resultados do desenvolvimento, com enfoque na capacidade de detecção e reconhecimento do modelo integrado e discutindo um pouco sobre a execução da aplicação em nuvem.

4 Resultados e Discussão

4.1 Detecção e Reconhecimento

4.1.1 ICDAR 2011

O conjunto de imagens de validação do ICDAR 2011 conta com 141 imagens e é importante de constatar que são imagens de relativa baixa resolução para os padrões atuais, onde as maiores contêm cerca de 315 mil pixels. São caracterizadas por estarem no contexto de imagens de anúncios e anexos de e-mails, com textos primariamente horizontais.

Ao aplicar a solução sobre os exemplos de validação do dataset, em termos de desempenho, todas as imagens foram processadas com sucesso em 36.31 segundos em um ambiente com aceleração gráfica, média de 257 ms por imagem. Para fins comparativos, a mesma tarefa agora executada sem a aceleração gráfica levou 328.44 segundos, cerca de 9 vezes mais tempo, executando na mesma máquina alocada no serviço Paperspace, que consta com 30GB de memória RAM, disponibilidade de processamento gráfico com NVIDIA QUADRO M4000 e processamento CPU baseado em instâncias AWS C4, disponibilizam 8 núcleos, 16 threads do processador Intel Xeon E5-2666, com clock de 2.9 GHz.

Apesar do maior tempo de processamento, o uso da solução em ambientes sem uma placa gráfica compatível ainda não se torna inviável, sobretudo considerando que a média de tempo de processamento por imagem foi de 2.32 segundos. No entanto, fica desencorajado a execução sobre um conjunto muito grande de imagens, já que o processamento corresponde a uma carga muito alta dependendo do hardware onde a solução for executada, uma vez que a execução em CPU alocou aproximadamente 17.4GB de memória, volume maior do que a quantidade total de grande parte dos sistemas mais domésticos.

Para executar o script de avaliação dos resultados da solução, o comando abaixo pode ser executado, fornecendo o caminho para um arquivo .ZIP que contém os arquivos de resultados para cada imagem do conjunto de validação.

```
python script.py -g=gt.zip -s=res_img_.zip
```

O resultado da melhor configuração foi o apresentado na Tabela 1. Em suma, a avaliação usa três métricas: Precisão, Recall e F1-Score. A Precisão é a relação dos acertos (verdadeiros positivos) sobre a contagem total de previsões positivas (verdadeiros positivos e falsos positivos), ou seja, dentre as previsões da solução, qual é a porcentagem de acerto. O Recall relaciona a quantidade de acertos com a quantidade total de casos verdadeiros. O F1-Score é basicamente a média harmônica das métricas Precisão e Recall.

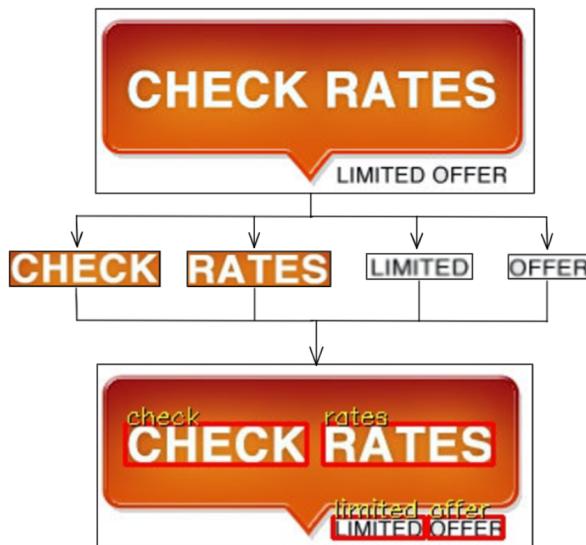


Figura 10 – Exemplo de resultado de reconhecimento. Imagem 52 do conjunto de validação.

```
Calculated!
{
    "precision": 0.7068273092369478,
    "recall": 0.7343532684283728,
    "hmean": 0.7203274215552524,
    "AP": 0
}
```

Tabela 1 – Avaliação de resultados sobre a base ICDAR 2011.

Precisão (%)	Recall (%)	F1-Score (%)
70.68	73.44	72.03

Em resumo, isso demonstra que a cada 100 palavras detectadas e processadas, aproximadamente 71 tiveram o texto corretamente extraído. É um resultado bem satisfatório para o trabalho desenvolvido e o nível de simplicidade que foi adotado. Estes números colocariam essa solução em nono lugar entre 18 outros trabalhos submetidos na plataforma de desafios Robust Reading Competition.

A Figura 10 exemplifica as etapas da solução apresentada. A imagem de entrada é processada pela rede de detecção que extrai as regiões de texto regredindo as coordenadas das bounding-boxes. Com isso, essas regiões são então cortadas da imagem original e alimentam a rede de reconhecimento para extração do texto decodificado.

Apesar do resultado satisfatório, o mais interessante é aprofundar não nos acertos, onde o modelo reconheceu as palavras certas, mas sim onde ele errou e identificar limitações e, eventualmente, futuras otimizações.



Figura 11 – Comparaçāo entre entrada e saída sobre a imagem 5 do set de validação do ICDAR 2011

Um dos casos mais evidentes onde a solução apresentou problemas foi na presença de caracteres especiais. Uma das limitações do modelo de reconhecimento é a lista de caracteres passíveis de reconhecimento, que contempla apenas os caracteres do alfabeto da língua inglesa, de A até Z, adicionado dos dígitos decimais, de 0 até 9. Este dicionário limitado restringe muito a capacidade de detecção em que não são tão difíceis de observar. Caracteres de pontuação, acentos, marcações como o cifrão (\$), entre outros casos, acabam dificultando o acerto do reconhecimento. A Figura 11 exemplifica esta constatação. Nela, é possível observar que o trecho que representa o preço do item anunciado na imagem 5 do conjunto de validação, que contém cifrão, vírgula e asterisco, não foi reconhecida com sucesso. Apesar de o método aproximar bem os caracteres desconhecidos, reconhecendo o cifrão (\$) como um cinco (5) e o asterisco (*) como a letra X, o resultado não é satisfatório nesse caso.

Outra dificuldade ficou aparente em imagens de baixa resolução, que acabam contendo regiões de texto ainda menores. Em geral, a regressão das *bounding-boxes* fica um pouco mais grosseira, mas ainda satisfatórias. Entretanto, a maior limitação foi o reconhecimento. As regiões de texto recortadas da imagem original acabam ficando bem pequenas e com pouca definição, o que trouxe dificuldades para o modelo pré-treinado CRNN utilizado. A Figura 12 exemplifica essa dificuldade. Pode-se observar que nenhum dos "títulos" de cada uma das fotos que estão presentes na Figura 12 foram localizadas com sucesso, mas não obtiveram a igualdade entre predição e gabarito (ficaram marcadas com uma área azul).

Um outro detalhe que demonstra uma dificuldade mais global do problema de reconhecimento de texto em cenas é a fonte utilizada na frase “A Love Story”, na região superior da Figura 12. Apesar do reconhecimento ter relativo sucesso nesse exemplo, a localização não conseguiu segmentar corretamente todas as palavras da frase. A generalização da detecção e do reconhecimento para os mais diversos tipos de fontes é um dos principais problemas que motivam o uso de redes neurais profundas para reconhecimento de texto. [8]

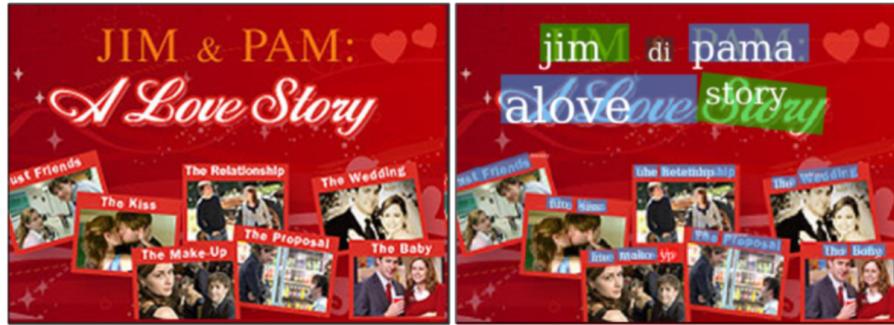


Figura 12 – Exemplo de imagem com instâncias de texto bem pequenas em resolução.
Imagen 28 do ICDAR 2011

O ICDAR 2011 apresenta casos mais complexos de detecção de reconhecimento de texto se comparado ao contexto de aplicações OCR convencionais, que lidam com reconhecimento de texto estruturado, sem grandes variações de fontes e planos de fundo, o que é bastante válido para avaliar a solução. No entanto, ainda não são casos reais de texto em cena, problema que motivou este trabalho. O dataset ICDAR 2013 contém exemplos mais característicos de texto de cena.

4.2 ICDAR 2013

O conjunto de imagens de validação do ICDAR 2013 conta com 233 imagens de tamanhos variados. As menores têm cerca de 640 pixels de altura e 480 pixels de comprimento, enquanto as maiores têm dimensões comparáveis à resolução 4K, com 3888 pixels de altura e 2592 pixels de comprimento. Vale ressaltar que as imagens passam por uma redução em resolução ao entrarem no modelo CRAFT, que maximiza a maior dimensão da imagem para 1280 pixels e ajusta a segunda dimensão mantendo a relação de aspecto original, justamente para otimizar o desempenho do método.

Ao aplicar a solução sobre os exemplos de validação do dataset, em termos de desempenho, todas as imagens foram processadas com sucesso em 207.74 segundos em um ambiente com aceleração gráfica, exatamente os mesmos recursos computacionais disponíveis na validação do ICDAR 2011, atingindo uma média de 892 ms por imagem. A média de tempo de processamento por imagem é cerca de 3.5 vezes maior quando comparado ao dataset anterior, ICDAR 2011. As predições não foram executadas sem aceleração gráfica por economia de recursos.

Com base nas mesmas métricas apresentadas na Seção 4.1.1, durante a discussão sobre os resultados contra o ICDAR 2011, a avaliação da solução no ICDAR 2013 demonstra resultados similares aos vistos na Seção 4.1.1, disponíveis na Tabela 2. Cerca de 7 acertos a cada 10 predições. Tendo em vista que é uma base um pouco mais desafiadora, é um resultado novamente satisfatório. Comparando Precisão e Recall, nota-se um desvio



Figura 13 – Demonstração de um falso positivo durante a avaliação da solução contra o dataset ICDAR 2013

maior, de aproximadamente 6 pontos percentuais, o que indica um maior número de falsos positivos, ou seja, regiões detectadas como regiões de texto que não estão nas anotações de gabarito. Em alguns casos, uma mesma palavra acabou sendo segmentada de maneira errada, em outros, regiões visivelmente de não-texto foram detectadas. A Figura 13 demonstra esse fenômeno.

```
Calculated!
{
    "precision": 0.7043390514631686,
    "recall": 0.7611777535441657,
    "hmean": 0.7316561844863733,
    "AP": 0
}
```

Tabela 2 – Avaliação de resultados sobre a base ICDAR 2013.

Precisão (%)	Recall (%)	F1-Score (%)
70.43	76.11	73.17

Entretanto, as principais dificuldades de detecção e reconhecimento são compartilhadas com o que foi observado durante a avaliação na base ICDAR 2011. Caracteres especiais, pontuação e acentos são limitações conhecidas do reconhecimento.

Por introduzir mais exemplos de imagens verdadeiramente com textos em ambientes naturais, algumas novas dificuldades apareceram. O desafio que provavelmente é mais evidente está relacionado aos eventuais artefatos nas imagens decorrente de reflexos em algumas imagens de texto sob vidro ou com incidência de iluminação de alta intensidade (*flashes* de câmeras fotográficas), onde, em alguns exemplos, a detecção foi sub-ótima (Figura 14) e em outros, o reconhecimento não foi o melhor (Figura 15).

Outra dificuldade conhecida de problemas de reconhecimento de texto em cenas está relacionada a palavras em destaque, o que pode ser relativamente comum, dado o contexto de uma imagem com diversos objetos na cena. Apesar de ser um exemplo



Figura 14 – Exemplo de imagem com artefatos que trouxeram dificuldades para a localização correta do texto.



Figura 15 – Exemplo de texto sob vidro, com plano de fundo desafiadores para o reconhecimento.

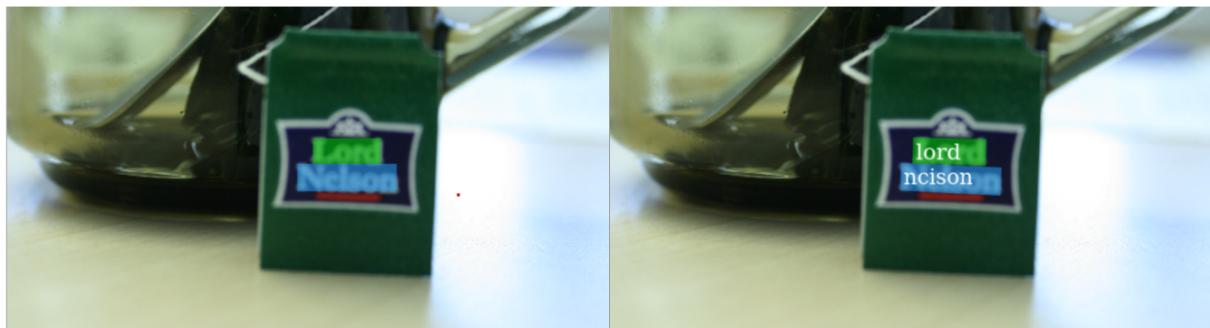


Figura 16 – Exemplo de imagem com texto em desfoco

mais simples, a Figura 16 demonstra esse caso. A etiqueta na infusão está visivelmente em desfoco e complicou o trabalho do reconhecimento por deixar os caracteres mais ambíguos.

4.2.1 Imagens autorais

Com caráter mais qualitativo, é interessante inferir a qualidade das soluções em imagens fora do contexto de datasets conhecidamente utilizados para treino e validação, para experienciar a qualidade do reconhecimento em imagens do dia-a-dia. Com esse objetivo, algumas imagens no álbum pessoal do autor deste trabalho foram selecionadas para passarem pela solução apresentada. Alguns exemplos estão disponíveis abaixo.

Algumas imagens utilizadas foram bastante desafiadoras para a solução, tanto em termos de localização quanto reconhecimento. Imagens com texto bem localizado e visível, alguns letreiros e placas tiveram bons resultados, mas alguns casos de palavras



Figura 17 – Exemplo de imagem autoral onde a solução apresentou dificuldades com texto curvo e estilizado. Textos reconhecidos: “sney” e “intmalakingon”.



Figura 18 – Exemplo de imagem autoral onde uma região de texto muito longa não foi extraída muito bem, principalmente no início da palavra, o que dificultou o reconhecimento. Texto reconhecido: “permmusem”.



Figura 19 – Exemplo de imagem autoral com fontes bastante estilizadas que trouxeram dificuldade tanto para a detecção, quanto para o reconhecimento. Textos reconhecidos: “sey” e “fpan”.

mais longas e curvadas, mesmo que levemente, não tiveram muito sucesso, especialmente se apresentarem caligrafia muito estilizada, como é possível observar na Figura 19. A respeito de textos curvados, melhorias na solução poderiam melhorar a situação, como a aplicação de soluções para transformar as imagens antes de passar pelo reconhecimento como, por exemplo, uma rede STN (*Spatial Transformer Network* [15]), a fim de obter uma imagem com o mínimo de curvatura possível, facilitando o trabalho de reconhecimento.

Um exemplo foi bastante interessante, pois contrariou as expectativas quanto ao reconhecimento, o que demonstra o potencial que a solução pode apresentar com algumas iterações de melhorias. A Figura 20 mostra um caso de sucesso na localização e reconhecimento onde o contexto no qual o texto está inserido tem bastante informação e a fonte do texto, principalmente do trecho escrito “Disney’s”, é bastante estilizado, e mesmo



Figura 20 – Exemplo de reconhecimento com sucesso em condições desafiadoras em imagem autoral. Textos reconhecidos: “disneys”, “electrical” e “parade”.



Figura 21 – Exemplo de imagem autoral com alta precisão de reconhecimento. Textos reconhecidos: “nelson”, “rolihlahla” “mandela”, “1918”, “2013”, “hamba”, “kahle”, “madiba”, “ve”, “honour”, “your”, “legacy”, “apartheid”, “museum”.

assim, o reconhecimento foi bem preciso.

Agora, quando o texto se encontra em situações bastante favoráveis, por exemplo, bem iluminado, sem oclusões, em geral disposto horizontalmente e com caligrafia não rebuscada, a solução atinge o seu potencial máximo, conseguindo localizar e interpretar com muita precisão. A Figura 21 demonstra esse caso, onde temos um cartaz de boas-vindas do museu do Apartheid, na África do Sul.

4.3 Disponibilização em nuvem

O resultado final da prova de conceito de publicar a aplicação final em um ambiente remoto através de computação em nuvem foi de relativo sucesso. Ao final de todas as etapas citadas em 3.4, a aplicação foi publicada na plataforma do Heroku com sucesso. No entanto, como constatado na Seção 4.1.1, a demanda de recursos computacionais da solução integrada foi relativamente grande, em especial a pressão sobre disponibilidade de memória RAM no sistema é grande.

O ambiente que é disponibilizado de graça pelo Heroku é bastante limitado em recursos, por questões óbvias, portanto já não era esperado uma boa performance, sobretudo pelo fato do ambiente não contar com aceleração gráfica.

A partir de alguns testes sobre aplicação publicada, pode-se observar que o ambiente não é de fato estável para a aplicação que foi publicada. Ao requisitar a execução do reconhecimento em imagens menores do dataset ICDAR 2011, com dimensões próximas de 300px por 300px ainda é possível obter uma resposta do servidor, porém para imagens maiores e mais pesadas, a aplicação ultrapassa os limites de memória que tem disponível para o contêiner e, como medida de segurança imposta pelo Heroku, o contêiner é desligado sem conseguir terminar o processo de reconhecimento.

5 Conclusão

Proin non sem. Donec nec erat. Proin libero. Aliquam viverra arcu. Donec vitae purus. Donec felis mi, semper id, scelerisque porta, sollicitudin sed, turpis. Nulla in urna. Integer varius wisi non elit. Etiam nec sem. Mauris consequat, risus nec congue condimentum, ligula ligula suscipit urna, vitae porta odio erat quis sapien. Proin luctus leo id erat. Etiam massa metus, accumsan pellentesque, sagittis sit amet, venenatis nec, mauris. Praesent urna eros, ornare nec, vulputate eget, cursus sed, justo. Phasellus nec lorem. Nullam ligula ligula, mollis sit amet, faucibus vel, eleifend ac, dui. Aliquam erat volutpat.

Referências

- 1 BAEK, Y. et al. Character region awareness for text detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2019. p. 9365–9374. Citado 4 vezes nas páginas [5](#), [7](#), [8](#) e [9](#).
- 2 SHI, B.; BAI, X.; YAO, C. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 39, p. 2298–2304, 2017. Citado 4 vezes nas páginas [5](#), [9](#), [10](#) e [11](#).
- 3 KARATZAS, D. et al. Icdar 2011 robust reading competition - challenge 1: Reading text in born-digital images (web and email). In: *Proceedings of the 2011 International Conference on Document Analysis and Recognition*. USA: IEEE Computer Society, 2011. (ICDAR '11), p. 1485–1490. ISBN 9780769545202. Disponível em: <<https://doi.org/10.1109/ICDAR.2011.295>>. Citado 2 vezes nas páginas [5](#) e [20](#).
- 4 KARATZAS, D. et al. Icdar 2013 robust reading competition. In: *2013 12th International Conference on Document Analysis and Recognition*. [S.l.: s.n.], 2013. p. 1484–1493. Citado 2 vezes nas páginas [5](#) e [21](#).
- 5 ZHAO, W.; JIANG, W.; QIU, X. Deep learning for covid-19 detection based on ct images. *Scientific Reports*, v. 11, 2021. Citado na página [1](#).
- 6 Gustav Tauschek. *Reading machine*. US2026330A, 27 maio 1929. Citado na página [1](#).
- 7 RAISI, Z. et al. Text detection and recognition in the wild: A review. *ArXiv*, abs/2006.04305, 2020. Citado na página [2](#).
- 8 LONG, S.; HE, X.; YAO, C. Scene text detection and recognition: The deep learning era. *International Journal of Computer Vision*, v. 129, p. 161–184, 2020. Citado 2 vezes nas páginas [2](#) e [29](#).
- 9 NAVER Corporation. Disponível em: <<https://www.navercorp.com/en/>>. Citado na página [7](#).
- 10 SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015. Citado na página [7](#).
- 11 RONNEBERGER, O.; FISCHER, P.; BROX, T. U-net: Convolutional networks for biomedical image segmentation. In: *MICCAI*. [S.l.: s.n.], 2015. Citado na página [7](#).
- 12 GUPTA, A.; VEDALDI, A.; ZISSERMAN, A. Synthetic data for text localisation in natural images. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, p. 2315–2324, 2016. Citado na página [8](#).
- 13 KORNILOV, A. S.; SAFONOV, I. V. An overview of watershed algorithm implementations in open source libraries. *Journal of Imaging*, v. 4, n. 10, 2018. ISSN 2313-433X. Disponível em: <<https://www.mdpi.com/2313-433X/4/10/123>>. Citado na página [8](#).

- 14 MERKEL, D. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, v. 2014, n. 239, p. 2, 2014. Citado na página 25.
- 15 JADERBERG, M. et al. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015. Disponível em: <<http://arxiv.org/abs/1506.02025>>. Citado na página 33.

Apêndices

APÊNDICE A – Arquivo de descrição do ambiente virtual Conda utilizado no desenvolvimento

```

name: craft-detector
channels:
- defaults
dependencies:
- _libgcc_mutex=0.1=main
- _openmp_mutex=4.5=1_gnu
- _pytorch_select=0.1=cpu_0
- blas=1.0=mkl
- bzip2=1.0.8=h7b6447c_0
- ca-certificates=2021.10.26=h06a4308_2
- cairo=1.16.0=hf32fb01_1
- certifi=2021.10.8=py37h06a4308_2
- cffi=1.14.6=py37h400218f_0
- click=8.0.3=pyhd3eb1b0_0
- cloudpickle=2.0.0=pyhd3eb1b0_0
- cycler=0.10.0=py37_0
- cytoolz=0.11.0=py37h7b6447c_0
- dask-core=2021.8.1=pyhd3eb1b0_0
- dataclasses=0.8=pyh6d0b6a4_7
- dbus=1.13.18=hb2f20db_0
- expat=2.4.1=h2531618_2
- ffmpeg=4.0=hcdf2ecd_0
- flask=2.0.2=pyhd3eb1b0_0
- fontconfig=2.13.1=h6c09931_0
- freeglut=3.0.0=hf484d3e_5
- freetype=2.10.4=h5ab3b9f_0
- fsspec=2021.8.1=pyhd3eb1b0_0
- glib=2.69.1=h5202010_0
- graphite2=1.3.14=h23475e2_0
- gst-plugins-base=1.14.0=h8213a91_2
- gstreamer=1.14.0=h28cd5cc_2

```

- harfbuzz=1.8.8=hffaf4a1_0
- hdf5=1.10.2=hba1933b_1
- icu=58.2=he6710b0_3
- imageio=2.9.0=pyhd3eb1b0_0
- intel-openmp=2019.4=243
- itsdangerous=2.0.1=pyhd3eb1b0_0
- jasper=2.0.14=h07fcdf6_1
- jinja2=3.0.2=pyhd3eb1b0_0
- jpeg=9d=h7f8727e_0
- kiwisolver=1.3.1=py37h2531618_0
- lcm2=2.12=h3be6417_0
- ld_impl_linux-64=2.35.1=h7274673_9
- libffi=3.3=he6710b0_2
- libgcc-ng=9.3.0=h5101ec6_17
- libgfortran-ng=7.5.0=ha8ba4b0_17
- libgfortran4=7.5.0=ha8ba4b0_17
- libglu=9.0.0=hf484d3e_1
- libgomp=9.3.0=h5101ec6_17
- libmklml=2019.0.5=0
- libopencv=3.4.2=hb342d67_1
- libopus=1.3.1=h7b6447c_0
- libpng=1.6.37=hbc83047_0
- libstdcxx-ng=9.3.0=hd4cf53a_17
- libtiff=4.2.0=h85742a9_0
- libuuid=1.0.3=h1bed415_2
- libvpx=1.7.0=h439df22_0
- libwebp-base=1.2.0=h27cf23_0
- libxcb=1.14=h7b6447c_0
- libxml2=2.9.12=h03d6c58_0
- locket=0.2.1=py37h06a4308_1
- lz4-c=1.9.3=h295c915_1
- markupsafe=2.0.1=py37h27cf23_0
- matplotlib=3.3.2=h06a4308_0
- matplotlib-base=3.3.2=py37h817c723_0
- mkl=2020.2=256
- mkl-service=2.3.0=py37he8ac12f_0
- mkl_fft=1.3.0=py37h54f3939_0
- mkl_random=1.1.1=py37h0573a6f_0
- ncurses=6.2=he6710b0_1

```
- networkx=2.6.2=pyhd3eb1b0_0
- ninja=1.10.2=hff7bd54_1
- numpy=1.19.2=py37h54aff64_0
- numpy-base=1.19.2=py37hfa32c7d_0
- olefile=0.46=py37_0
- opencv=3.4.2=py37h6fd60c2_1
- openjpeg=2.4.0=h3ad879b_0
- openssl=1.1.1m=h7f8727e_0
- packaging=21.0=pyhd3eb1b0_0
- partd=1.2.0=pyhd3eb1b0_0
- pcre=8.45=h295c915_0
- pillow=8.3.1=py37h2c7a002_0
- pip=21.2.2=py37h06a4308_0
- pixman=0.40.0=h7f8727e_1
- py-opencv=3.4.2=py37hb342d67_1
- pycparser=2.20=py_2
- pyparsing=2.4.7=pyhd3eb1b0_0
- pyqt=5.9.2=py37h05f1152_2
- python=3.7.11=h12debd9_0
- python-dateutil=2.8.2=pyhd3eb1b0_0
- pytorch=1.8.1=cpu_py37h60491be_0
- pywavelets=1.1.1=py37h7b6447c_2
- pyyaml=5.4.1=py37h27cf23_1
- qt=5.9.7=h5867ecd_1
- readline=8.1=h27cf23_0
- scikit-image=0.14.2=py37he6710b0_0
- scipy=1.1.0=py37h7c811a0_2
- setuptools=58.0.4=py37h06a4308_0
- sip=4.19.8=py37hf484d3e_0
- six=1.16.0=pyhd3eb1b0_0
- sqlite=3.36.0=hc218d9a_0
- tbb=2021.3.0=hd09550d_0
- tbb4py=2021.3.0=py37hd09550d_0
- tk=8.6.11=h1ccaba5_0
- toolz=0.11.1=pyhd3eb1b0_0
- torchvision=0.2.1=py37_0
- tornado=6.1=py37h27cf23_0
- typing-extensions=3.10.0.2=hd3eb1b0_0
- typing_extensions=3.10.0.2=pyh06a4308_0
```

- werkzeug=2.0.2=pyhd3eb1b0_0
- wheel=0.37.0=pyhd3eb1b0_1
- xz=5.2.5=h7b6447c_0
- yaml=0.2.5=h7b6447c_0
- zlib=1.2.11=h7b6447c_3
- zstd=1.4.9=haebb681_0

prefix: /var/home/mmilani/.conda/envs/craft-detector

APÊNDICE B – Arquivo de descrição do ambiente virtual Conda utilizado durante a validação

```

name: str-eval
channels:
- defaults
dependencies:
- _libgcc_mutex=0.1=main
- _openmp_mutex=4.5=1_gnu
- ca-certificates=2021.10.26=h06a4308_2
- certifi=2021.10.8=py37h06a4308_0
- ld_impl_linux-64=2.35.1=h7274673_9
- libffi=3.3=he6710b0_2
- libgcc-ng=9.3.0=h5101ec6_17
- libgomp=9.3.0=h5101ec6_17
- libstdcxx-ng=9.3.0=hd4cf53a_17
- ncurses=6.2=he6710b0_1
- openssl=1.1.1l=h7f8727e_0
- pip=21.0.1=py37h06a4308_0
- python=3.7.11=h12debd9_0
- readline=8.1=h27cf23_0
- setuptools=58.0.4=py37h06a4308_0
- sqlite=3.36.0=hc218d9a_0
- tk=8.6.11=h1ccaba5_0
- wheel=0.37.0=pyhd3eb1b0_1
- xz=5.2.5=h7b6447c_0
- zlib=1.2.11=h7b6447c_3
- pip:
  - numpy==1.21.3
  - polygon3==3.0.9.1
prefix: /var/home/mmilani/.conda/envs/str-eval

```


APÊNDICE C – Arquivo Dockerfile utilizado na publicação da aplicação em nuvem

```
FROM continuumio/miniconda3:latest

ARG FLASK_ENV='production'
ENV FLASK_ENV $FLASK_ENV
ENV FLASK_APP '/app/app.py'

WORKDIR /app
COPY ./requirements-cpu.yml /app/requirements-cpu.yml
RUN conda env create -f /app/requirements-cpu.yml

ADD . /app

CMD [
    "conda",
    "run",
    "--no-capture-output",
    "-n",
    "craft-detector",
    "flask",
    "run",
    "--host=0.0.0.0",
    "--port=\$PORT"
]
```


APÊNDICE D – Exemplo de arquivo de resultado para avaliação

3181,1200,3574,1200,3574,1278,3181,1278,mandela
2622,1211,3154,1211,3154,1289,2622,1289,rolihlahla
2241,1217,2600,1217,2600,1294,2241,1294,nelson
3430,1305,3591,1305,3591,1383,3430,1383,2013
3237,1311,3436,1311,3436,1388,3237,1388,1918
3403,1581,3598,1589,3595,1660,3400,1651,legacy
3032,1593,3253,1593,3253,1660,3032,1660,honour
3261,1599,3397,1591,3401,1653,3264,1661,your
2667,1604,2905,1604,2905,1676,2667,1676,madiba
2921,1604,3021,1604,3021,1660,2921,1660,ve
2462,1615,2644,1615,2644,1687,2462,1687,kahle
2205,1632,2450,1623,2453,1695,2208,1703,hamba
3354,1913,3617,1894,3623,1974,3359,1993,museum
3016,1934,3362,1911,3367,1990,3021,2014,apartheid