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

Abril 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
Abril de 2022

Resumo

Nos últimos anos, os termos, aprendizado de máquina e aprendizado profundo ficaram populares e já são presentes em soluções disponíveis para o público geral, sendo detecção e reconhecimento de texto um tópico quente nesse contexto, especialmente quando imagens não são documentos. Este trabalho de graduação propõe o desenvolvimento de uma solução integrada de detecção e reconhecimento de texto de cenas com base em soluções do estado da arte, respectivamente, CRAFT (*Character Region Awareness for Text Detection*) e CRNN (*Convolutional Recurrent Neural Networks*), com uma prova de conceito da solução com uso de computação em nuvem. A solução avaliada nas bases de dados ICDAR 2011 e ICDAR 2013 atingiu uma média de 70% de precisão. Apesar do relativo sucesso, a solução apresentou limitações no reconhecimento de cenários mais complexos, típicos dos problemas de detecção e reconhecimento de texto em cenas, por exemplo, palavras com fontes estilizadas, com grande amplitude de tamanho de caracteres e palavras não horizontais, além do alto uso de recursos computacionais.

Palavras-chave: Aprendizado de Máquina. Aprendizado Profundo. Redes Neurais Profundas. Reconhecimento Óptico de Caracteres.

Abstract

Recently, concepts like machine-learning and deep-learning have become quite popular and have been already been present on customer facing applications. Text detection and recognition problems are also included, as many new developments were made in the past years, specially regarding non-document images. This graduation project presents the development of an end-to-end solution for scene text detection and recognition based on current state-of-the-art methods CRAFT (Character Region Awareness for Text Detection) and CRNN (Convolutional Recurrent Neural Network), also hosting a proof of concept application for the presented solution using cloud computing. The presented solution was benchmarked using ICDAR 2011 and ICDAR 2013 datasets and has averaged 70% precision. Despite the relative success, the proposed solution has its limitations regarding more difficult scene text detection and recognition cases and required high availability of computational resources.

Keywords: Machine-Learning. Deep-Learning. Deep Neural Networks. Optical Character Recognition.

Lista de ilustrações

Figura 1 – Imagem de uma faixada de comércio, ilustrando um caso de STR. Fonte [1] . . . 3
Figura 2 – Diferenças em alto nível entre aprendizado profundo e aprendizado de máquina referente às etapas de processamento. Fonte [2] . . . 5
Figura 3 – Exemplo de rede convolucional, ilustrando as camadas da rede neural. Fonte [2] . . . 7
Figura 4 – Ilustração de um exemplo de rede neural recorrente. Fonte [3] . . . 10
Figura 5 – Categorias de abordagens na solução de detecção de texto. Fonte [4] . . . 12
Figura 6 – Ilustração das etapas de geração dos arquivos de *ground-truth* para a etapa de treino do CRAFT. Fonte [5] . . . 14
Figura 7 – Exemplificação do passo a passo para geração de anotações ao nível de caracteres durante a etapa de treino do CRAFT. Fonte [5]. . . . 15
Figura 8 – Ilustração do *pipeline* de reconhecimento do CRNN. Fonte [6]. . . . 17
Figura 9 – Exemplo das imagens geradas a partir da detecção pelo CRAFT sobre uma imagem autoral. . . . 22
Figura 10 – Comando do gerenciador de versão Git para criação de sub-módulos. . . . 24
Figura 11 – Ilustração das etapas de processamento da solução proposta. . . . 25
Figura 12 – Exemplos de imagens da base de dados ICDAR 2011. Fonte [7] . . . 27
Figura 13 – Exemplos de imagens da base de dados ICDAR 2013. Fonte [1] . . . 27
Figura 14 – Interface de validação disponibilizada pela plataforma de desafios *Robust Reading Competition*. . . . 29
Figura 15 – Interface de validação disponibilizada pela plataforma de desafios *Robust Reading Competition*, com visualização de detalhes da avaliação por imagem. . . . 30
Figura 16 – Página inicial da interface de gestão da aplicação no Heroku. . . . 33
Figura 17 – Exemplo de resultado de reconhecimento. Imagem 52 do conjunto de validação. . . . 36
Figura 18 – Comparação entre entrada e saída sobre a imagem 5 do conjunto de validação do ICDAR 2011 . . . 37
Figura 19 – Exemplo de imagem com instâncias de texto pequenas em resolução. Imagem 28 do ICDAR 2011 . . . 38
Figura 20 – Demonstração de um falso positivo durante a avaliação da solução contra a base de dados ICDAR 2013. . . . 39
Figura 21 – Exemplo de imagem com artefatos que trouxeram dificuldades para a localização correta do texto. . . . 40

Figura 22 – Exemplo de texto sob vidro, com plano de fundo desafiadores para o reconhecimento. 40
Figura 23 – Exemplo de imagem com texto em desfoque. 40
Figura 24 – Exemplo de imagem autoral onde a solução apresentou dificuldades com texto curvo e estilizado. Textos reconhecidos: “sney” e “intmalakingon”. 41
Figura 25 – Exemplo de imagem autoral onde uma região de texto longa não foi extraída com sucesso, principalmente no início da palavra, o que dificultou o reconhecimento. Texto reconhecido: “permmusem”. 41
Figura 26 – Exemplo de imagem autoral com fontes estilizadas que trouxeram dificuldade tanto para a detecção, quanto para o reconhecimento. Textos reconhecidos: “sey” e “fpan”. 41
Figura 27 – Exemplo de reconhecimento com sucesso em condições desafiadoras em imagem autoral. Textos reconhecidos: “disneys”, "electrical"e “parade”. 42
Figura 28 – 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”. 42

Lista de tabelas

Tabela 1 – Avaliação de resultados sobre a base ICDAR 2011. 36
Tabela 2 – Avaliação de resultados sobre a base ICDAR 2013. 39

Lista de abreviaturas e siglas

CNN	*Convolutional Neural Network*
CRAFT	*Character Region Awareness for Text Detection*
CRNN	*Convolutional Recurrent Neural Network*
CTC	*Connectionist Temporal Classification*
DL	*Deep-Learning*
FCN	*Fully Convolutional Network*
HOG	*Histogram of Oriented Gradients*
ICDAR	*International Conference on Document Analysis and Recognition*
ILSVRC	*ImageNet Large Scale Visual Recognition Challenge*
KPN	*Kernel Proposal Network*
LSTM	*Long Short-Term Memory*
ML	*Machine-Learning*
OCR	*Optical Character Recognition*
ReLU	*Rectified Linear Unit*
RNN	*Recurrent Neural Network*
RPN	*Region Proposal Network*
STN	*Spatial Transform Network*
STR	*Scene Text Recognition*

Sumário

1 INTRODUÇÃO 1
1.1 *Scene Text Recognition* 2
1.2 Objetivo 3
1.3 Considerações finais 4

2 FUNDAMENTAÇÃO TEÓRICA 5
2.1 Aprendizado de Máquina e Aprendizado Profundo 5
2.1.1 Redes Neurais Convolucionais 6
2.1.2 Redes Neurais Recorrentes 9
2.2 Evolução do *Scene Text Recognition* 10
2.2.1 Detecção de Texto 11
2.2.1.1 CRAFT 13
2.2.2 Reconhecimento de Texto 15
2.2.2.1 CRNN 16
2.3 Considerações finais 17

3 METODOLOGIA 19
3.1 Experimentação com soluções de reconhecimento e detecção de texto 19
3.2 Desenvolvimento da solução fim-a-fim a partir de detectores e reconhecedores de texto 22
3.3 Método de avaliação da solução proposta 26
3.3.1 Base de Dados 26
3.3.1.1 ICDAR 2011 26
3.3.1.2 ICDAR 2013 27
3.3.2 Métricas 27
3.3.3 Interface de Validação 28
3.4 Disponibilização da solução fim-a-fim em nuvem 29
3.4.1 Criação da Aplicação Web 31
3.4.2 Contêiner, Docker e Implantação 31
3.5 Considerações finais 33

4 RESULTADOS E DISCUSSÃO 35
4.1 Detecção e Reconhecimento 35
4.1.1 ICDAR 2011 35
4.1.2 ICDAR 2013 38
4.1.3 Imagens autorais 40

4.2 Disponibilização em nuvem 42

5 CONCLUSÃO 45

REFERÊNCIAS 47

APÊNDICES 53

APÊNDICE A – ARQUIVO DE DESCRIÇÃO DO AMBIENTE VIRTUAL CONDA UTILIZADO NO DESENVOLVIMENTO 55

APÊNDICE B – ARQUIVO DE DESCRIÇÃO DO AMBIENTE VIRTUAL CONDA UTILIZADO DURANTE A VALIDAÇÃO 59

APÊNDICE C – ARQUIVO DOCKERFILE UTILIZADO NA IMPLANTAÇÃO DA APLICAÇÃO EM NUVEM . . . 61

APÊNDICE D – EXEMPLO DE ARQUIVO DE RESULTADO PARA AVALIAÇÃO 63

1 Introdução

Nos últimos anos, os termos, aprendizado de máquina e aprendizado profundo, ficaram populares e já estão presentes em soluções disponíveis para o público geral, desde a assistente virtual dos dispositivos móveis e eletrônicos de consumo, 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 [8]. São conceitos utilizados em aplicações de diversos segmentos, sendo um especialmente importante para esse trabalho de graduação, o de processamento de imagem. Alguns exemplos de funcionalidades atuais que, de alguma forma, se apoiaram em conceitos e técnicas relacionados ao aprendizado de máquina são funcionalidades "modo retrato" [9] e "modo noturno" [10] do aplicativo de câmera de dispositivos móveis, sendo que as referências citadas documentam os estudos dessas funcionalidades em dispositivos da linha Pixel, desenhados pela Google.

Um escopo em particular de aplicação de técnicas de aprendizado de máquina que evoluiu com o crescimento da popularidade e acessibilidade dos conceitos que envolvem aprendizado de máquina 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, 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 [11], 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 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 nas instâncias de texto, elas precisam conhecer ao menos algumas características dos caracteres e do texto que serão reconhecidos. Para exemplificar, a *Reading Machine* [11], proposta por Gustav Tauschek, era um aparelho mecânico que utilizava um disco de comparação, que guardava 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 na totalidade e a principal ferramenta utilizada atualmente são métodos sob o domínio do 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 tração na última 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 no nome em inglês *Scene Text Recognition*. Uma analogia para o STR seria aplicar soluções de OCR diretamente de fotos capturadas por uma câmera de um dispositivo móvel. Algumas soluções acessíveis ao público geral que lidam com STR pode-se citar aplicações como o *Google Lens* e o *Apple Live Text*, que permitem a extração de texto de imagens diretamente da câmera do dispositivo móvel, muitas vezes em situações similares aos que são explorados nos trabalhos que procuram resolver o reconhecimento de texto em cenas.

A principal 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 ou comércio, ilustrado pela Figura 1, 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 aprendizado profundo, que demonstram conseguir ir um passo à frente no quesito reconhecimento de padrões em comparação às técnicas clássicas de processamento de imagem. O uso em especial das redes neurais convolucionais, comumente abreviadas para CNN pelo nome em inglês *Convolutional Neural Networks*, um divisor de águas na evolução dessas soluções. [4, 12]. A própria aplicação *Google Lens*, citada anteriormente, utiliza redes convolucionais baseadas em RPN (*Region Proposal Networks)* e redes recorrentes do tipo LSTM (*Long Short-Term Memory*) para detectar e reconhecer texto nas imagens enviadas pelos usuários [13].

A popular biblioteca OpenCV[1] é uma porta de entrada para iniciantes experimentares soluções de detecção e reconhecimento de texto mais robustas. No módulo de redes neurais profundas da biblioteca, é disponibilizado ferramentas para executar modelos pré-

[1] https://opencv.org/

Figura 1 – Imagem de uma faixada de comércio, ilustrando um caso de STR. Fonte [1]

treinados de detecção e de reconhecimento de texto, compatíveis com trabalhos relevantes na área, como EAST [14] e o CRNN [6], por exemplo.

A pesquisa e experimentação de novos métodos para resolver os mais diferentes problemas que permeiam o STR ainda é atual, com trabalhos publicados recentemente. Em [15], por exemplo, é proposta uma nova arquitetura de detecção, abreviada por KPN pelo nome em inglês *Kernel Proposal Network*), utilizando redes convolucionais e segmentação semântica para melhorar a detecção das fronteiras de palavras.

Já em [16], uma solução unificada de localização e reconhecimento é sugerida utilizando redes convolucionais inspirada em soluções de detecção de objetos baseadas em codificadores *Transformers* [17], que, no que lhes concerne, utilizam métodos de atenção. As regiões de texto são localizadas apenas por um ponto de referência central, não por uma área bem definidas como é comum. Posteriormente um decodificador é utilizado para gerar a sequência de caracteres no entrono do ponto de referência.

1.2 Objetivo

O principal objetivo deste trabalho de graduação é desenvolver e avaliar uma solução integrada de reconhecimento de texto em cenas a parir do uso de componentes para detecção e reconhecimento isolados que já fizeram parte do estado da arte, respectivamente CRAFT e CRNN.

Como objetivo complementar, de modo a gerar uma prova de conceito sobre a solução desenvolvida, a partir do uso de provedores de computação em nuvem, hospedar

uma aplicação *web* para disponibilizar o uso da solução desenvolvida na Internet.

1.3 Considerações finais

Dada a importância do aprendizado profundo, e em especial das CNNs nesse contexto de detecção e reconhecimento de texto, o próximo capítulo irá apresentar alguns conceitos básicos associados a essas estruturas.

2 Fundamentação Teórica

2.1 Aprendizado de Máquina e Aprendizado Profundo

O aprendizado de máquina é uma área no amplo espaço de estudo de inteligência artificial. O aprendizado de máquina envolve o estudo e aplicação de algoritmos especialistas em reconhecer padrões em bases de dados, muitas vezes não triviais para uma análise humana [18], e são capazes de, a partir da recorrente exposição a esses dados e aos padrões identificados, responder perguntas e predizer informações sobre novos dados nunca vistos. O aprendizado profundo é uma vertente na área de aprendizado de máquina e decorre a partir das redes neurais artificiais profundas, ou seja, com mais camadas ocultas entre a entrada e a saída da rede.

Uma das grandes diferenças entre aprendizado profundo e aprendizado de máquina é como a representação e extração de características acontece automatizadamente em métodos de aprendizado profundo. A maior profundidade das redes permite que características de diferentes níveis de abstração sejam montadas pelos diferentes níveis da rede [19], o que contrasta com a geral necessidade de etapas de pré-processamento sobre os dados de entrada de algoritmos de aprendizado de máquina para a montagem de características [18, 2]. A Figura 2 ilustra em alto nível essa diferença entre aprendizado de máquina e aprendizado profundo.

Técnicas e algoritmos, tanto de aprendizado de máquina quanto de aprendizado profundo, podem ser basicamente classificados em diferentes grupos [2]:

- Algoritmos de aprendizado supervisionado: baseiam-se na exposição de grande volumes de exemplos devidamente anotados, isso é, cada exemplo que será consumido

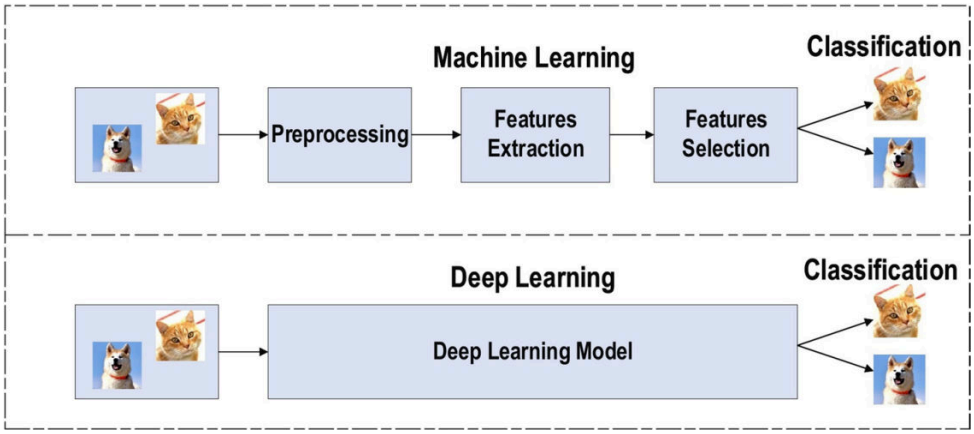

Figura 2 – Diferenças em alto nível entre aprendizado profundo e aprendizado de máquina referente às etapas de processamento. Fonte [2]

pelo modelo em treinamento deve conter também o resultado esperado na saída do algoritmo, para ser possível a comparação entre o resultado do modelo contra o valor de gabarito. A partir dessa comparação tem-se o eventual erro de predição, sendo iterativamente utilizado pelo algoritmo para refinar suas respostas em geral com no processo chamado *back-propagation*.

- Algoritmos de aprendizado semi-supervisionado: assim como o nome sugere, é uma abordagem válida quando não existe grande volume de dados com as anotações necessárias. Ainda assim, a depender da aplicação, a existência de um volume menor de dados anotados pode ser melhor que nenhum dado anotado, portanto ainda são importantes no treinamento.

- Algoritmos de aprendizado não supervisionado: É aplicável quando não existem bases de dados em volume com exemplos anotados. Dessa forma, o próprio algoritmo tentará derivar relacionamentos entre exemplos de modo a encontrar características intrínsecas aos dados de entrada.

Duas estruturas populares de aprendizado profundo, as redes neurais convolucionais e as redes neurais recorrentes, serão abordadas nas próximas seções. Ambas são base dos métodos de detecção e reconhecimento de texto utilizados ao longo do trabalho de graduação.

2.1.1 Redes Neurais Convolucionais

As redes neurais convolucionais, abreviadas por CNN pelo nome em inglês *Convolutional Neural Network*, é um modelo de rede neural com inspiração no córtex visual animal, uma estrutura cerebral que processa os estímulos visuais originados a partir dos olhos. Apesar da inspiração e o foco inicial em problemas de visão computacional, atualmente é um conceito já aplicado em outros segmentos de pesquisa, como processamento de sinais, processamento de voz e linguagem, para citar algumas dessas aplicações [2, 20].

As arquiteturas de redes convolucionais mais comuns são, em geral, composições de camadas de diferentes categorias, observáveis a partir da Figura 3, cada uma com um propósito específico:

1. **Camada de Convolução**: A camada de convolução tem como papel processar a imagem de entrada e extrair mapas de características baseado na aplicação de diferentes filtros bi-dimensionais (ou *kernels*) sobre a imagem em uma operação de convolução, de forma análoga ao uso de filtros morfológicos, no âmbito de processamento de imagem. Para cada filtro aplicado nas camadas de convolução, uma nova representação dos dados de entrada é gerada, sendo comumente chamado mapa

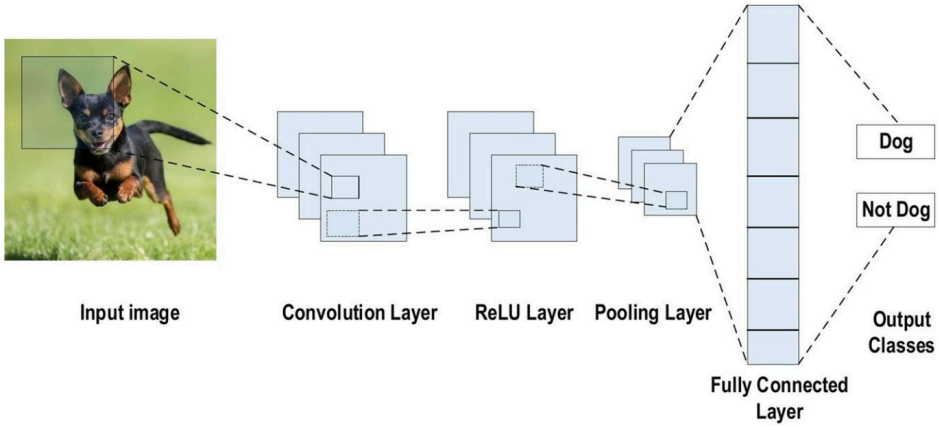

Figura 3 – Exemplo de rede convolucional, ilustrando as camadas da rede neural. Fonte [2]

de características. As camadas iniciais em geral conseguem entender características de mais alto nível, menos abstratas, por estarem mais próximas da entrada de dados. Camadas mais profundas extraem características mais abstratas, já que são ativadas por mapas de características provenientes das camadas mais iniciais [21].

A introdução de não-linearidade nas redes convolucionais decorre da aplicação de funções de ativação, normalmente ao final da operação de convolução. A função de ativação mais comum é a ReLU (*Rectified Linear Unit*, em inglês), um dos motivos é a sua eficiência computacional quando comparada a outras funções de ativação [22], como *sigmod* e *tanh*. A Equação 2.1 apresenta a definição matemática da função de ativação ReLU.

$$f(u) = max(0, u) \tag{2.1}$$

Ainda sobre a camada de convolução, existem quatro hiper-parâmetros que uma breve introdução se faz necessária:

- ***Stride***: É o passo da aplicação do filtro sobre a imagem. *Stride* igual a um significa que a convolução percorrerá a imagem de píxel em píxel.
- ***Receptive-Field***: É o tamanho dos filtros aplicados à imagem. *Receptive-Field* igual a dois significa que os filtros têm dimensões 2×2 píxeis.
- ***Depth***: É a profundidade de cada mapa de características e está diretamente relacionado ao número de filtros aplicados à imagem.
- ***Zero-Padding***: É a espessura da borda de zeros que deve ser aplicada na imagem pré-convolução. Durante a concepção de um modelo de CNN, é importante considerar as dimensões da imagem de entrada e dos mapas de características de cada camada de convolução. É importante que os filtros consigam percorrer a imagem sem extrapolar nenhum índice durante as iterações. Aplicar o

valor correto de *Zero-Padding* e *Stride* faz com que o tamanho dos mapas de características sejam previsíveis, facilitando a modelagem.

2. **Camada de Pooling**: A camada de *pooling* é responsável por fazer o *down-sampling* dos mapas de características, ou seja, reduz a dimensão desses mapas, o que, em geral, melhora o desempenho de convergência modelo, potencializa a generalização do que está sendo aprendido controlando o problema de *over-fitting*, diminuindo o número de parâmetros e ativações do modelo.

   A configuração mais comum de uma camada de *pooling* é o de filtros com *Receptive-Field* igual a dois aplicados com *Stride* igual a dois. Isso faz com que o mapa de características tenha sua altura e largura diminuídas pela metade, dessa forma reduzindo em 75% sua complexidade em quantidade de parâmetros da rede [23].

   Para saber os novos valores de ativação do mapa reduzido, a operação mais comum executada durante o processo de convolução a operação *max* ou *average*, com o intuito de conservar as ativações mais predominantes de cada mapa de características.

3. **Camada de Totalmente Conectada**: A camada *fully-connected* é comum de ser aplicada no final do *pipeline* de processamento de uma arquitetura de CNN pela sua capacidade de decisão. Essa camada é mais parecida com a estrutura comum de redes neurais artificiais, onde existe uma ou mais camadas de neurônios que recebem parcelas de ativação de todos os nós da camada anterior.

   Em problemas de classificação, essa camada é responsável por abstrair as características extraídas por camadas ocultas anteriores em valores de probabilidade, de modo a formalizar o resultado da predição [19]. A operação mais comum na saída da camada totalmente conectada é o $softmax$, exposta pela Equação 2.2.

$$softmax(y_j) = \frac{exp(y_j)}{\sum exp(y_j)} \tag{2.2}$$

   Apesar de ser um componente comum em arquiteturas de redes convolucionais, as camadas totalmente conectadas nem sempre estão presentes. Dependendo do propósito da rede, pode-se omitir a camada de classificação totalmente conectada e substituir por novas camadas de convolução com filtros de dimensão 1×1 píxeis para fazer o mesmo trabalho de classificação ou ainda usar novas camadas para de-convolução e *up-sampling* para a saída da rede ser uma imagem segmentada semanticamente [24]. Redes convolucionais que não contam com as últimas camadas densas e utilizam camadas convolucionais no seu lugar são normalmente chamadas FCN, *Fully-Convolutional-Networks*.

Quando aplicado a entradas visuais em forma de imagens, o tamanho final do modelo treinado e o número total de parâmetros de uma rede CNN é menor se comparada

com uma rede neural convencional. Em [25] é exemplificado uma rede neural convencional, por serem densas, ou seja, os neurônios das camadas da rede são todos interligados, o número de parâmetros treinados da rede cresce rápido com as dimensões da imagem de entrada. Em contrapartida, nas redes convolucionais, cada nó da rede é ativado por uma secção da imagem de entrada, descrito pelo hiper-parâmetro *Receptive-Field*, dessa forma, na operação de convolução, cada neurônio da rede tem a responsabilidade de identificar características de secções distintas da imagem de entrada.

A partir do uso dos componentes mais comum e básicos de uma rede convolucional em diferentes composições, é possível configurar arquiteturas mais complexas. Alguns trabalhos ao longo dos últimos anos exploraram novas arquiteturas de redes convolucionais, como, por exemplo, VGG [26], ResNet [27] e GoogLeNet [28]. Em [2] é feita uma revisão sobre as redes convolucionais mais relevantes da última década, introduzindo com mais detalhes cada arquitetura e suas motivações, com as principais evoluções entre cada uma das redes citadas. Uma base de imagens utilizada por alguns desses trabalhos é o ImageNet [29], que contém vasto número de exemplos que são base de uma competição de detecção e classificação de objetos, ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*) [30]. Unindo a robustez e capacidade de representação de características das redes convolucionais com a alta exposição a diversos exemplos através do ImageNet, as redes melhores colocadas no ILSVRC ao longo dos anos se tornaram referência em extração de características e, em diversos casos [5, 6], são utilizadas em soluções de contexto mais específicos justamente por técnicas como o *transfer-learning* [31].

2.1.2 Redes Neurais Recorrentes

As redes neurais recorrentes, abreviadas por RNN (*Recurrent Neural Networks*, em inglês) são populares no contexto de processamento de linguagem natural devido ao seu caráter de conseguir processar dados sequencialmente [2]. A principal característica das redes recorrentes é que a predição no instante de tempo T é estimulada pelo dado de entrada e pelo estado interno da rede no instante de tempo passado, $T-1$. De maneira simplista, essa característica se assemelha ao conceito de memória, onde a rede sempre considera tudo o que já foi processado em momentos anteriores para compor resultado do momento corrente, carregando o contexto temporalmente.

A versão dita canônica das redes neurais recorrentes tem sua estrutura ilustrada pela Figura 4, onde é possível observar que a representação aberta da rede se assemelha a uma cadeia de "neurônios" conectados. O fator de não-linearidade deriva do uso de funções como *tanh*, *sigmoid* e ReLu durante a computação do estado interno da rede em cada instante de tempo. As redes recorrentes também podem ser compostas em configurações diferentes, onde dois exemplos são redes recorrentes profundas, duas ou mais RNNs empilhadas [32], e redes recorrentes bi-direcionais [33], profundas ou não, que lidam

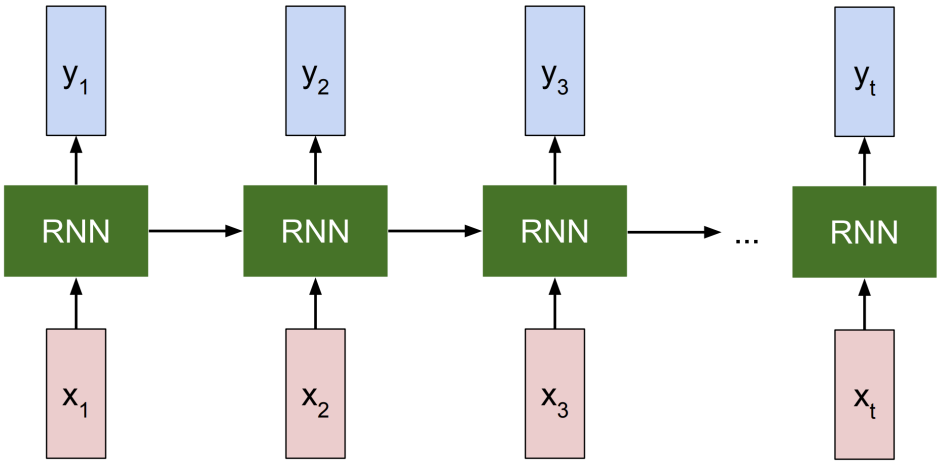

Figura 4 – Ilustração de um exemplo de rede neural recorrente. Fonte [3]

com problemas onde o contexto a *priori* e a *posteriori* são importantes.

Apesar das redes neurais recorrentes, na sua forma original, conseguirem lidar com entradas e saídas sequenciais de tamanhos arbitrários, empiricamente demonstrou problemas ao lidar com sequências longas, principalmente uma grande sensibilidade aos problemas de explosão e desaparecimento de gradiente em métodos de treinamento baseados em propagação do erro de predição [34]. Para lidar com esse problema, novas variantes de redes recorrentes foram idealizadas, sendo a que mais se popularizou foi a LSTM, abreviação para *Long Short-Term Memory* [35].

A principal mudança entre a rede RNN convencional para a rede LSTM foi a introdução de uma nova estrutura capaz de aprender a preservar contexto histórico na rede e podem ser interpretados como células de memória. A partir de novos meios treináveis de adicionar, preservar e limpar informações dessa célula, a rede se tornou mais capaz e eficiente na gestão dos parâmetros responsáveis por segurar o contexto histórico da rede, mitigando os problemas de *Vanishing* e *Exploding Gradients*.

2.2 Evolução do *Scene Text Recognition*

Optical Character Recognition, abreviado por OCR, e mais especificamente *Scene Text Recognition*, abreviado por STR, podem ser atacados como um sub-conjunto de problemas de identificação de objetos [4, 12] e esse fato trouxe diversas adaptações de métodos mais clássicos de detecção de objetos para o contexto de detecção de texto. O exemplo mais convencional de OCR, a detecção de texto manuscrito ou impresso, apesar de ser desafiador (em especial o caso de manuscritos), é tido como resolvido, com soluções de sucesso aplicadas em larga escala [4, 36], na maioria por apresentarem um contexto mais simples onde o texto está inserido, com planos de fundo mais consistentes, com menos ruídos, e padrões de mais fácil detecção quando comparado aos casos de STR.

Apesar da introdução sobre as redes convolucionais na seção anterior e a breve menção que as redes neurais convolucionais são amplamente utilizadas no espoco de problemas de detecção e reconhecimento de texto atualmente, vale mencionar que nem sempre foi assim. Trabalhos que estudaram e resolveram a detecção e reconhecimento de texto já eram conduzidos mesmo antes da ampla popularidade do aprendizado profundo e aliavam métodos de extração de características, mais artesanais e específicos para o contexto de aplicação, com modelos e algoritmos de aprendizado de máquina ditos clássicos para a realizar a classificação/predição.

Em [4, 12] é exposto que o processo de interpretação de texto contidos em imagens é composto por duas grandes etapas: A localização e/ou detecção do texto e o reconhecimento do texto. A etapa de detecção e/ou localização visa justamente responder se, dada uma imagem, existe texto nela e, caso positivo, adequadamente localizá-lo. Já o reconhecimento tem a responsabilidade de interpretar o texto contido na imagem, de modo a conseguir transformar regiões de texto em sequências de caracteres que possam ser utilizados por computadores.

A partir dessa interpretação, nessa seção uma breve revisão bibliográfica será exposta abordando a evolução das soluções de detecção e reconhecimento até a popularização dos métodos de aprendizado profundo. Serão abordados em maior detalhes dois trabalhos, um sobre detecção e outro sobre reconhecimento, pois farão parte do desenvolvimento prático deste trabalho de graduação.

2.2.1 Detecção de Texto

Em momentos anteriores ao crescimento de popularidade do aprendizado profundo, as soluções de detecção de texto em cenas precisavam de profunda análise e manuseio das imagens para conseguir identificar padrões em torno do problema para ser então possível, manualmente, construir e descrever uma série de características que seriam a base para identificar o texto na imagem. Todo esse processo é bastante artesanal, consumindo muito tempo humano, e as características produzidas, em geral, são especializadas no contexto de atuação, como citado por [19], ou seja, o reuso de uma característica montada no contexto de detecção de texto em outros problemas, como, por exemplo, o processamento de linguagem natural, seria difícil.

Para o problema de detecção de texto, ainda no período anterior ao uso do aprendizado profundo, [4, 12] constatam ser possível, de maneira geral, agrupar as soluções baseadas em aprendizado de máquina clássico em duas categorias:

- Soluções baseadas no conceito CCA (*Connected Components Analysis*, em inglês), que se baseiam em agregar regiões da imagem que compartilham características, como, por exemplo, cor, textura e extremidades [37]. Dois trabalhos relevantes nesse

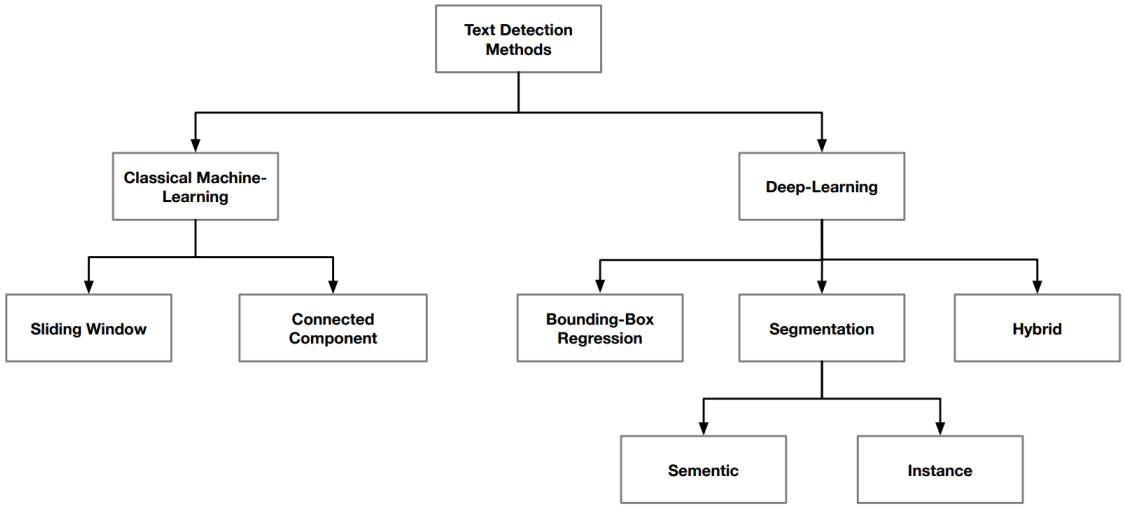

Figura 5 – Categorias de abordagens na solução de detecção de texto. Fonte [4]

segmento são os que originaram os métodos *Stoke Width Transform* (SWT) [38] e *Maximum Stable Extremal Regions* (MSER) [39].

- Soluções baseadas no conceito de SW, abreviação do nome em inglês *Sliding Windows*, que se baseiam em percorrer a imagem com janelas, de diferentes tamanhos, e, simplificadamente, aplicar testes para a existência ou não de texto na área da janela observada em conjunto a um método classificador, treinados para detectar texto baseado em características construídas para tal.

A introdução de técnicas de aprendizado profundo na solução da detecção de texto expandiu o leque de possibilidades em torno de como lidar com o problema, o que movimentou pesquisas sobre o tema, o que pode ser constatado com o número de novos trabalhos abordando o problema nas últimas duas décadas. Inicialmente, os primeiros trabalhos que aplicavam redes neurais as utilizavam como meio de extração de características, substituindo grande parte do processo de engenharia de características específicas, para posteriormente aplicar os métodos de processamento de imagens e visão computacional já conhecidos.

Posteriormente, alguns conceitos e técnicas mais gerais de detecção de objetos foram trazidos para o contexto de reconhecimento de texto, o que criou, de acordo com [4], três linhas de solução para o problema de detecção de texto, ilustradas na Figura 5:

- Soluções por regressão de *bounding-boxes*: resolvem o problema sob a ótica de detecção de objetos para localizar a *bounding-box* esperada diretamente. Conceitos e métodos como SSD [40], YOLO [41], R-CNN [42] são frequentes nos trabalhos que seguem essa linha, pois derivam dos métodos clássicos de detecção de objetos. Algumas

referências a trabalhos baseados em regressão de *bounding-boxes* são: TextBoxes [43], TextBoxes++ [44], SegLink [45], EAST [14], entre outros.

- Soluções por segmentação: resolvem o problema sob a ótica de segmentação semântica classificando cada píxel para segmentar áreas de texto e de não-texto. A partir do mapa de predição, as *bounding-boxes* podem ser extraídas por pós-processamento. Algumas referências a trabalhos baseados em segmentação são: *Multi-Oriented Text Detection with Fully Convolutional Networks* [46], *Scene Text Detection via Holistic Multi-Channel Prediction* [47], *PixelLink* [48], *TextSnake* [49], entre outros.
- Soluções híbridas: como a categoria sugere, essas soluções se utilizam de ideias da abordagem por regressão de *bounding-boxes* e de segmentação semântica em simultâneo, para tentar se apoiar sobre ambos os conceitos.

A próxima sub-seção introduzirá uma solução de detecção em específico, que pode ser classificada como baseada em segmentação, e será o método base de detecção de texto na parte prática deste trabalho de graduação.

2.2.1.1 CRAFT

Character Region Awareness for Text Detection [5], ou simplesmente CRAFT, é um método de detecção publicado por Baek et al.[5], pesquisadores da empresa coreana Naver Corporation[1], que apresentou resultados 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 quadros por segundo, competitiva com os melhores métodos já publicados.

Baek et al.[5] introduz um método de detecção ao nível de caractere onde um modelo de rede convolucional FCN é criado a partir da reconhecida rede de extração de características VGG-16 [26]. A arquitetura da rede CRAFT se inspirou na rede U-Net [50] ao introduzir *skip-connections*, agregando características de alto e baixo nível entre os blocos de *up-sampling*, 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 píxel está localizado no centro de um caractere.
- Mapa de predição de afinidade entre caracteres (*Character Afiinity Score*): probabilidades de cada píxel 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

[1] https://www.navercorp.com/en

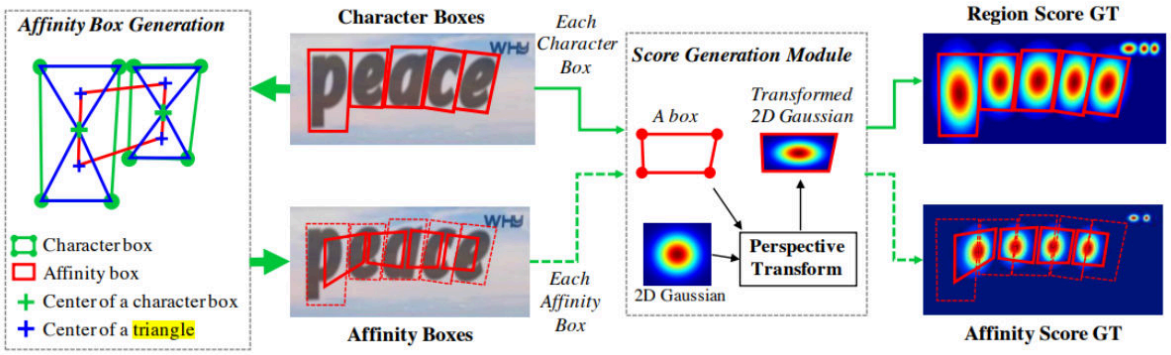

Figura 6 – Ilustração das etapas de geração dos arquivos de *ground-truth* para a etapa de treino do CRAFT. Fonte [5]

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 Figura 6 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 contam com anotações ao nível de caracteres, a rede primariamente é treinada com imagens com texto sintético, usando a base de dados SynthText [51].

Para refinar o treino em bases de dados 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 ao nível de caracteres para as imagens das bases de dados utilizados com auxílio de métodos de processamento de imagem, conforme exemplificado na Figura 7. O processo contém as seguintes etapas:

- *Cropping*: Extração das palavras que possuem região descrita nos arquivos de *ground-truth* das bases de dados de avaliação.

- *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* [52], 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

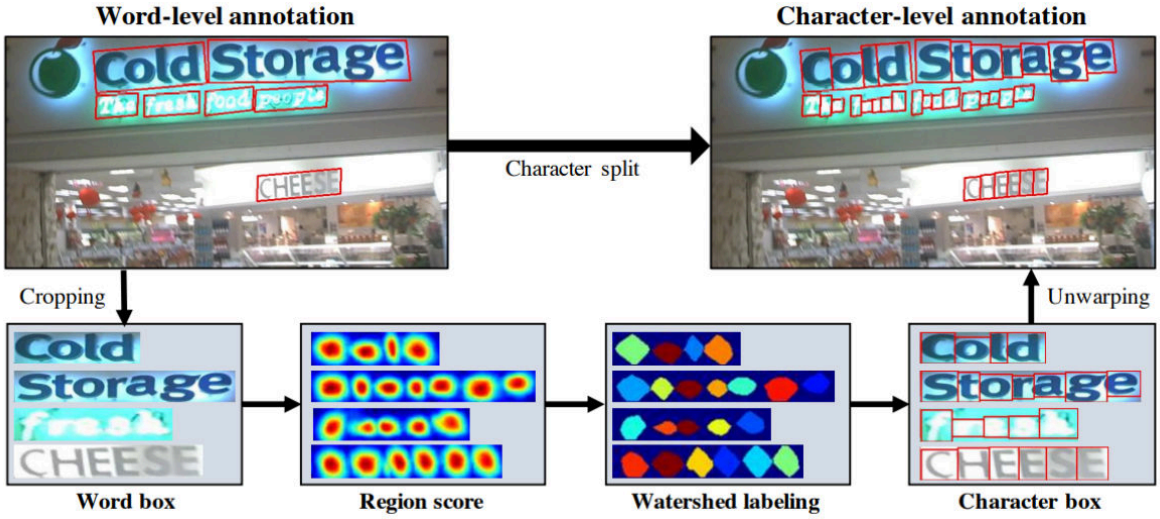

Figura 7 – Exemplificação do passo a passo para geração de anotações ao nível de caracteres durante a etapa de treino do CRAFT. Fonte [5].

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 anotações geradas sobre as imagens reais das bases de dados de avaliação, os gabaritos para o treinamento do modelo completo são gerados conforme explicado anteriormente e o treinamento da rede é refinado com esses novos exemplos.

O CRAFT conta com um pós-processamento simplificado em cima dos mapas de probabilidade 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

A evolução do segmento de problemas voltados ao reconhecimento de texto, de forma análoga aos avanços observados nas soluções dos problemas de detecção, também foi acelerada pela popularização do aprendizado profundo. Soluções anteriores a aplicação do aprendizado profundo faziam extenso uso de pré e pós processamento a partir de extração de características trabalhadas manualmente, como, por exemplo, HOG [53], aliado de algoritmos classificadores de aprendizado de máquina clássicos. Uma característica da maioria das soluções anteriores a adoção dos métodos de aprendizado profundo é que

seguem uma abordagem dita como *bottom-up* [4], onde resolvem o reconhecimento ao nível de caracteres para então concatenarem os resultados, formando palavras. Outros trabalhos abordaram o mesmo problema pela abordagem inversa, *top-down*, onde a palavra é reconhecida de uma vez só, no entanto, apresentam problemas em reconhecer palavras que não fazem parte do dicionário de treinamento da solução.

A partir do uso de redes profundas, algumas linhas para a solução do reconhecimento surgiram, mas a partir do levantamento presente em [4] e sugerido por [12], pode-se perceber que grande parte das soluções que já fizeram parte do estado da arte, de alguma forma, se apoiou nas redes convolucionais para codificação e extração de características. Para atender à característica sequencial do problema de reconhecimento de texto, diversos trabalhos recorrem a redes neurais recorrentes alimentadas pelos mapas de características resultantes das estruturas convolucionais.

Também por [4], observa-se uma dicotomia ao que se refere ao algoritmo de predição empregado pelas soluções do estado da arte. Muitas soluções se baseiam em CTC [54], abreviação do nome em inglês *Connectionist Temporal Classification*, e predição baseados em mecanismos de atenção [55].

Para introduzir com um pouco mais de detalhes um dos componentes base do desenvolvimento desse trabalho de graduação, a próxima seção abordará o método CRNN.

2.2.2.1 CRNN

Convolutional Recurrent Neural Networks [6], introduzido por Shi, Bai e Yao[6] é uma solução para o problema de reconhecimento de texto popular, citada em novos trabalhos e presente em trabalhos comparativos. Este método veio para resolver grandes dificuldade das soluções anteriores, por exemplo: lidar com entradas e saídas de largura variados, possibilitar o aprendizado conhecido como fim-a-fim, isto é, aplicar uma função de perda sobre o resultado do reconhecimento e essa perda ser propaganda 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 mapas de características sobre a imagem de entrada. Esses mapas são sequenciados para a rede neural recorrente, responsável por conseguir decodificar cada sequência em um possível caractere. A Figura 8 ilustra o *pipeline* de processamento que essa arquitetura executa.

A rede recorrente implementada no CRNN é da variação LSTM (*Long-Short Term Memory*) bi-direcional, pois, por visar reconhecer texto em cenas, onde existe muita variação na forma de como o texto é observado e 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.

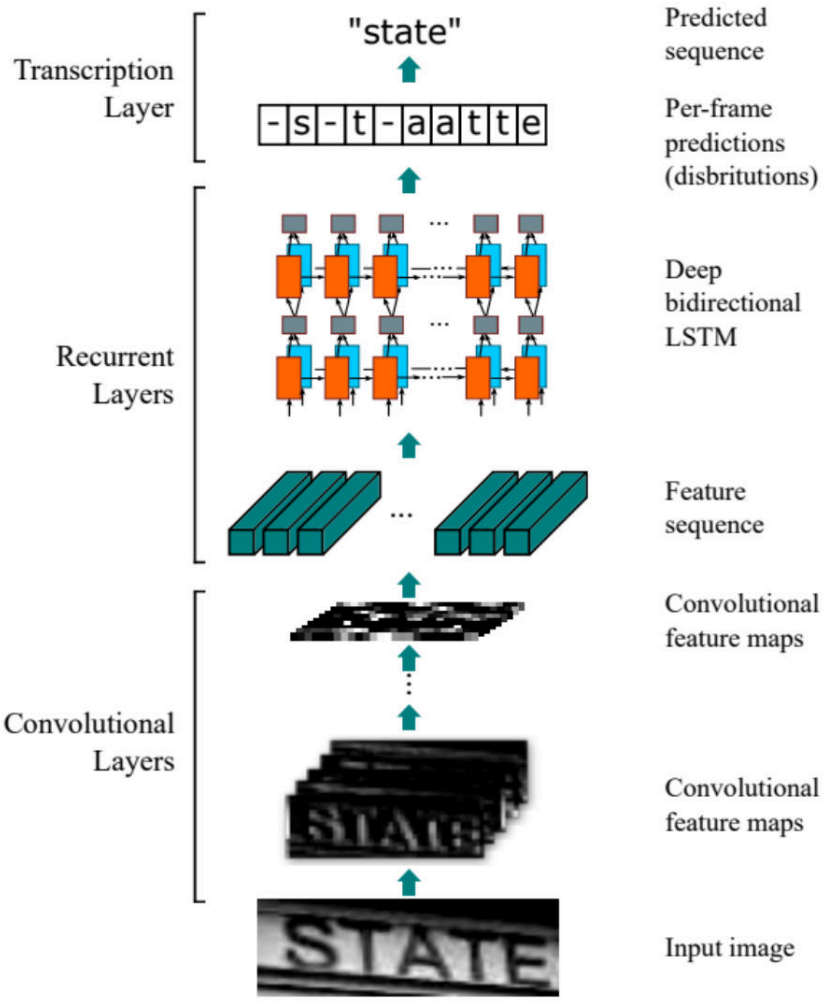

Figura 8 – Ilustração do *pipeline* de reconhecimento do CRNN. Fonte [6].

Os resultados obtidos foram competitivos quando comparados aos métodos de reconhecimento anteriores, até mesmo superiores aos que já usavam os conceitos de aprendizado profundo, com o adicional de prover meios de aprendizado integrado da rede convolucional e recorrente como uma unidade, além de um modelo mais enxuto em número de parâmetros [6].

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

Este capitulo introduziu brevemente os componentes mais populares de aprendizado profundo, as redes convolucionais e as redes recorrentes, e apresentou um pouco sobre a evolução da área de estudo de detecção e reconhecimento de texto, exemplificando a crescente adoção dos métodos de aprendizado profundo nas soluções dessa categoria de

problema. Adicionalmente, os métodos CRAFT e CRNN, componentes importantes na solução proposta por este trabalho de graduação, foram revisados com mais detalhes.

O próximo capítulo abordará as etapas percorridas no desenvolvimento da solução proposta, além de apresentar os meios de avaliação e implantação utilizados.

3 Metodologia

Uma vez introduzida a temática deste trabalho de 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 grandes 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. Implantação da solução em ambientes de computação em nuvem.

Dessa forma, as 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

Nessa seção será abordado um pouco das motivações sobre as escolhas pelos métodos de detecção e reconhecimento de texto, respectivamente CRAFT e CRNN, sob caráter mais técnico e prático, bem como os métodos escolhidos se diferem de outras soluções disponíveis publicamente na Internet.

A partir de uma rápida busca na Internet, em especial na plataforma do GitHub[1], 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 dificuldade para de fato conseguir executar o código desses repositórios é satisfazer os requisitos e dependências minímas, muitas vezes implícitos, por

[1] https://github.com

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.

Outro ponto na escolha dos métodos aplicados neste trabalho é a compatibilidade entre os métodos de detecção e reconhecimento. Como o objetivo deste trabalho se baseia na integração dessas soluções para trabalharem juntas, o caminho mais simples nessa linha é que ambos projetos consigam se executados no 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:

- EAST [14], com implementações dos usuários *argman*[2] e *jandz*[3]
- TextBoxes++ [44], com implementação do usuário *MhLiao*[4]
- PixelLink [48], com implementação do usuário *ZJULearning*[5]
- CRAFT [5], com implementação do usuário *clovaai*[6]

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[7], biblioteca popular na comunidade, facilitando 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 diferencial, já que os procedimentos de treinamento dessas soluções não são triviais, além de costumarem ser custosos temporal e computacionalmente, o que desviaria o foco do objetivo desse trabalho. Adicionalmente, o código-fonte contém instruções minímas de como executar o modelo pré-treinado, que se mostraram suficientes para iniciantes sem muita experiencia 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 equipamento físico sem aceleração gráfica, mesmo que com a penalização no tempo de execução. Essa capacidade

[2] https://github.com/argman/EAST
[3] https://github.com/janzd/EAST
[4] https://github.com/MhLiao/TextBoxes_plusplus
[5] https://github.com/ZJULearning/pixel_link
[6] https://github.com/clovaai/CRAFT-pytorch
[7] https://pytorch.org

foi importante, já que os componentes compatíveis não são baratos, dado o contexto atual de cripto-moedas e escassez de processadores gráficos no mercado [56].

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 popular, com implementação 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[8] foi utilizada. 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 com aceleração gráfica gratuitamente para fins de estudo e experimentação, com características similares ao Google Colab[9].

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[10], 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 referências 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 consegue derivar as coordenadas de *bounding-boxes* para cada palavra já é disponibilizado pelos autores do CRAFT. A Figura 9 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, desenhando os

[8] https://www.paperspace.com
[9] https://colab.research.google.com
[10] https://conda.io

Figura 9 – Exemplo das imagens geradas a partir da detecção pelo CRAFT sobre uma imagem autoral.

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 píxeis. 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 poderem alimentar a implementação do CRNN, completando a solução fim-a-fim do reconhecimento de texto.

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

De modo a aplicar o reconhecimento, utilizando o CRNN, exatamente sobre o texto detectado e localizado pelo CRAFT, escolheu-se 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, ao 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 utilizadas na implementação do CRNN.

Para a gestão de ambientes de desenvolvimento e suas dependências, foi utilizada 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, a primeira etapa foi replicar os repositórios para a conta pessoal do autor no GitHub. Esse processo na plataforma Github se chama *fork* e, em geral, é 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 de modo a facilitar a sincronização das bases de códigos. O recurso em questão é a criação de sub-módulos em um repositório Git. Usando o comando ilustrado na Figura 10 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.

Com as soluções na mesma base de código, uma etapa importante é garantir que ambas ainda funcionam isoladamente e validar a compatibilidade das dependências de

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

Figura 10 – Comando do gerenciador de versão Git para criação de sub-módulos.

cada uma. Como mencionado na Seção 3.1, uma das motivações da escolha especificamente dessas duas implementações citadas foi justamente a maior proximidade em *frameworks* e bibliotecas que utilizam, inclusive a possibilidade de ambas soluções poderem ser executadas sem a necessidade de aceleração gráfica. Dessa forma não houve 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 para 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 através de arquivos 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 de modo a 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

$imagem \leftarrow lerEntrada()$
$palavras \leftarrow []$
$regioesDeTexto \leftarrow executaDeteccaoCRAFT(imagem)$
for $regiaoDeTexto \in regioesDeTexto$ **do**
  $recorte \leftarrow recortarTextoDaImagemOriginal(imagem, regiaoDeTexto)$
  $palavra \leftarrow executaReconhecimentoCRNN(recorte)$
  $palavras \leftarrow paralvras + [palavra]$
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 respon-

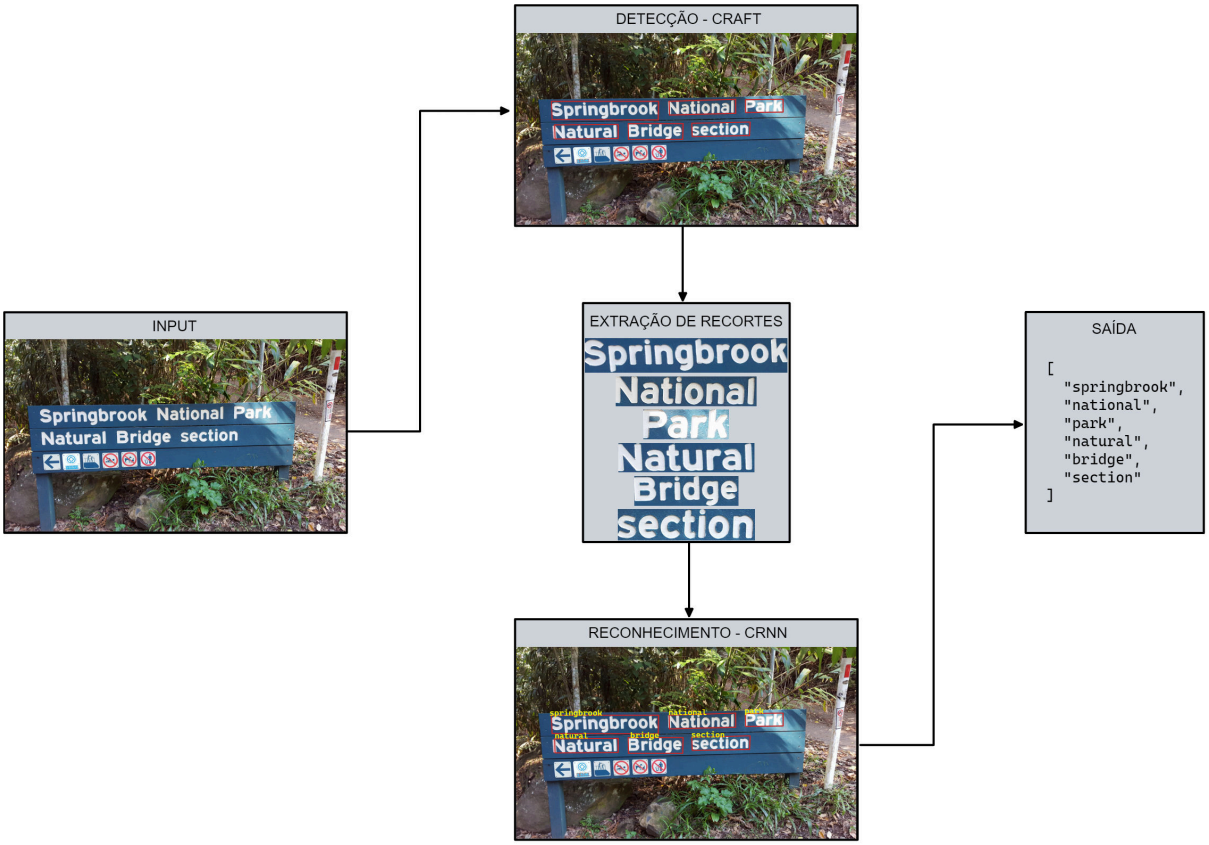

Figura 11 – Ilustração das etapas de processamento da solução proposta.

sável 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 pode-se recortar as palavras detectadas da imagem original e 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 11 ilustra, em alto nível, as etapas do processamento que compõem as soluções de STR fim-a-fim deste trabalho.

Ao fim dessas alterações, integrando os dois modelos de detecção e reconhecimento, 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*[11], (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 do nome da conferência em inglês. Diversas bases de dados 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 as bases de dados utilizadas para avaliação, as métricas utilizadas e o processo de avaliação da solução apresentada utilizando as ferramentas de avaliação disponibilizadas pela plataforma RRC.

3.3.1 Base de Dados

Conforme mencionado anteriormente, as bases de dados dos desafios vinculados ao ICDAR são utilizados por trabalhos voltados à detecção e reconhecimento de texto. Ambos os modelos utilizados no desenvolvimento deste trabalho referem-se às bases de dados ICDAR: O CRAFT cita as bases de dados ICDAR 2013 [1], ICDAR 2015 [57] e ICDAR 2017 [58] como bases de treinamento e de validação. O CRNN faz referências ao ICDAR 2003 [59] e ICDAR 2013 como bases de imagens para validação. As bases de dados utilizadas para a avaliação deste trabalho foram o ICDAR 2011 [7], base para o primeiro desafio da plataforma RRC chamado *Reading Text in Born-Digital Images*, e o ICDAR 2013, sendo a base para o segundo desafio da plataforma RRC, chamado *Focused Scene Text*.

3.3.1.1 ICDAR 2011

O ICDAR 2011 [7] conta com 410 imagens para treino e 141 imagens para validação no contexto de anúncios de Internet e anexos de *e-mails*, com palavras em geral na horizontal e anotadas com *bounding-boxes* retangulares por 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. A Figura 12 expõe algumas imagens desta base de dados.

[11] https://rrc.cvc.uab.es/

Figura 12 – Exemplos de imagens da base de dados ICDAR 2011. Fonte [7]

Figura 13 – Exemplos de imagens da base de dados ICDAR 2013. Fonte [1]

3.3.1.2 ICDAR 2013

O ICDAR 2013 [1] 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 por palavras e contam com a transição das mesmas, similarmente às imagens do ICDAR 2011. A Figura 13 expõe algumas imagens desta base de dados.

3.3.2 Métricas

Em conjunto às bases de dados, são fornecidos 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 nas bases de dados 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 predições positivas (verdadeiros positivos e falsos negativos), ou seja, dentre as predições 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*.

$$Precisão = \frac{VP}{VP + FP} \tag{3.1}$$

$$Recall = \frac{VP}{VP + FN} \tag{3.2}$$

$$F1Score = 2 \times \frac{Precisão \times Recall}{Precisão + Recall} \tag{3.3}$$

Para determinar se uma predição 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 predição 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 das bases de dados, e das coordenadas preditas pelo modelo de detecção. Ambos conjuntos de coordenadas descrevem uma área retangular, para as duas bases de dados utilizadas. 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 pode-se observar resultados de outros trabalhos de maneira simples e, quando disponível, até consultar quem são os autores, se existe algum repositório público para a base de código, visualizar o artigo do trabalho, etc.

No entanto, submeter os resultados diretamente na plataforma para avaliação remota não é a única opção. É possível também efetuar o *download* dos *scripts* que calculam os resultados, nos mesmos moldes da avaliação remota, de modo a executar todo o processo localmente. Essa foi a alternativa de avaliação adotada para esse trabalho de graduação.

Para ambos os meios de validação, é necessário preparar arquivos de resultados, para cada imagem da base de dados 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.

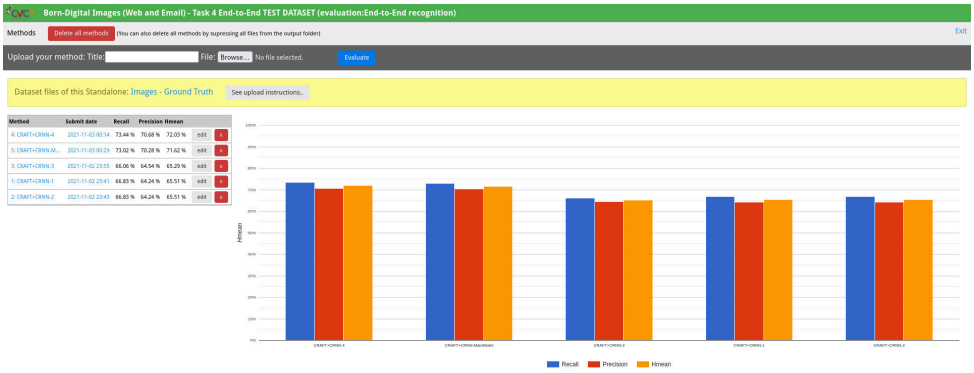

Figura 14 – Interface de validação disponibilizada pela plataforma de desafios *Robust Reading Competition*.

- 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 na base de dados.
- 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.

Um exemplo de arquivo de resultado foi disponibilizado no Apêndice D

Após gerar todos os arquivos necessários, resta agregá-los em um arquivo comprimido do tipo ZIP e executar a ferramenta de validação localmente. Os arquivos de execução local são escritos em Python e dependem de poucas bibliotecas. Executam um pequeno 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 14 mostra um pouco da interface de usuário que a aplicação fornece, com as funcionalidades de submeter o arquivo comprimido de solução e a Figura 15 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.

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

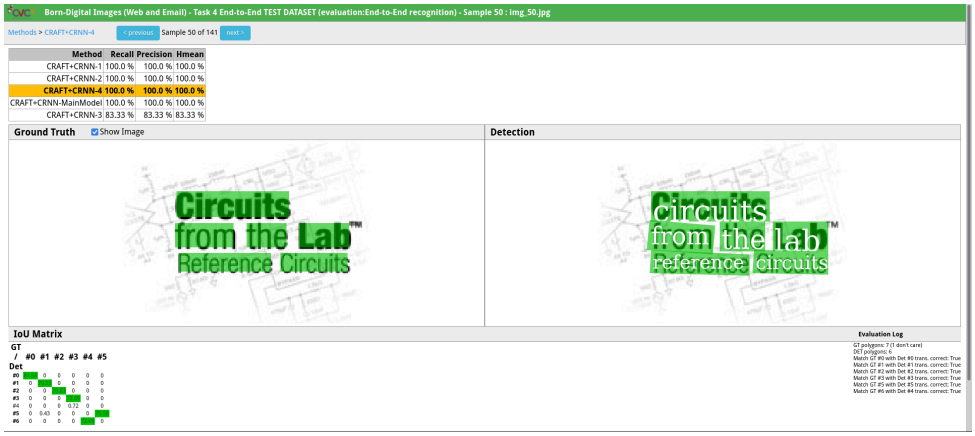

Figura 15 – Interface de validação disponibilizada pela plataforma de desafios *Robust Reading Competition*, com visualização de detalhes da avaliação por imagem.

funcionalidade para outros consumidores, possivelmente evoluindo 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[12] (Amazon Web Services) e a Paperspace, utilizada sob-demanda por esse trabalho durante o desenvolvimento conforme citado na Seção 3.1. Ambos serviços 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. No entanto, apesar de ambos serviços possuírem limites de uso onde os recursos são gratuitos, não são de domínio do autor deste trabalho de graduação. Como uma terceira opção, a plataforma escolhida para hospedar o desenvolvimento deste trabalho foi o Heroku[13]. Apesar da escolha de uma plataforma específica para hospedar a aplicação, as ferramentas e o processo até a implantação em geral são comuns entre plataformas, então, resguardadas as devidas proporções, o processo feito para implantar a aplicação no Heroku pode ser extrapolado para algo similar em outros provedores de infraestrutura.

O Heroku uma plataforma disponibilizada pela empresa Salesforce[14] 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 implantar uma aplicação para o Heroku envolvem poucos comandos e também pode hospedar aplicações usando recursos gratuitos. As limitações do ambiente gratuito e suas implicações na execução da aplicação serão abordadas no Capítulo 4.

[12] https://aws.amazon.com/pt
[13] https://heroku.com
[14] https://www.salesforce.com/

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 pelo ambiente de desenvolvimento. Para expor o modelo integrado na Internet, desenvolveu-se uma aplicação *web*, nos moldes de cliente-servidor, que responde requisições pelo protocolo HTTP, utilizando a arquitetura REST (*Representational State Transfer*) de comunicação. Para isso foi necessário fazer algumas adições na base de código para contemplar a implementação de uma aplicação *web* utilizando o framework Flask[15], 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, 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 integradamente. Com isso, foi possível executar a solução fim-a-fim programaticamente.

Além da nova classe implementada, foi necessário desenvolver um método com 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 realizou a requisição.

Com essas poucas mudanças já é possível, de maneira miníma e simplificada, atender pedidos de reconhecimento de texto sob demanda. O próximo passo adotado utilizou ferramentas e conceitos voltados a conteinerização da aplicação para facilitar a execução no ambiente remoto, descrito a seguir.

3.4.2 Contêiner, Docker e Implantação

Uma prática popular 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 bibliotecas, os binários, as configurações, entre outras dependências necessárias para a aplicação, de maneira consistente e de fácil reprodutibilidade. Esse processo é por vezes chamado “conteinerização”, onde um contêiner é esse empacotamento do código da aplicação e todas suas dependências, de fato sendo executado.

[15] https://flask.palletsprojects.com/en/2.1.x/

Uma das tecnologias de contêiner mais utilizadas é o Docker [60]. Mais detalhes sobre Docker e contêineres podem ser encontrados na página oficial da tecnologia.

Para gerar um contêiner da aplicação desenvolvida neste trabalho, a primeiro etapa é descrever como esse contêiner é construído, e, para isso, cria-se um arquivo chamado *Dockerfile*. No arquivo *Dockerfile* deve constar uma série de comandos executados rigorosamente pelo Docker para criar uma imagem Docker. Ao executar a imagem pelo Docker, um novo processo, um contêiner, é inicializado com o conteúdo da imagem executada e pode executar o código compilado na imagem Docker. O arquivo Dockerfile pode ser consultado diretamente no repositório desse trabalho ou no Apêndice C.

Uma dificuldade durante essa etapa de definição da imagem do contêiner e escrita do arquivo *Dockerfile* foi configurar adequadamente o ambiente virtual Python, gerenciado pelo Conda. Foram necessárias algumas tentativas sem sucesso de ativar o ambiente virtual corretamente dentro do contêiner para chegar na solução final demonstrada pelo Apêndice C. A criação do ambiente virtual a partir do arquivo de descrição de dependências do Conda, o *requirements.yml* e o comando de inicialização da aplicação *web* a partir do *framework* Flask são os comandos mais significativos do arquivo Dockerfile.

Ter a aplicação "conteinerizada" minimiza configuração para implantá-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. Todas as dependências necessárias para executar a aplicação já estão configuradas na imagem do contêiner e estarão pronto para 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 implantação da aplicação no Heroku se resume a execução de um comando a interface de linha de comando que a plataforma disponibiliza para implantar uma aplicação a partir de uma imagem Docker.

Para criar uma aplicação pelo Heroku, é necessária a criação de uma conta de usuário na plataforma e não tem nenhum custo. Pela interface de usuário da plataforma, criar uma aplicação gratuita é simples e direto, capaz de ser feita com poucas ações. A Figura 16 ilustra a visão da página de gestão da aplicação deste trabalho de graduação no Heroku.

De forma geral, o Heroku disponibiliza três meios para implantar o código-fonte de uma aplicação nos servidores da plataforma:

1. Envio do repositório Git contendo o código-fonte;

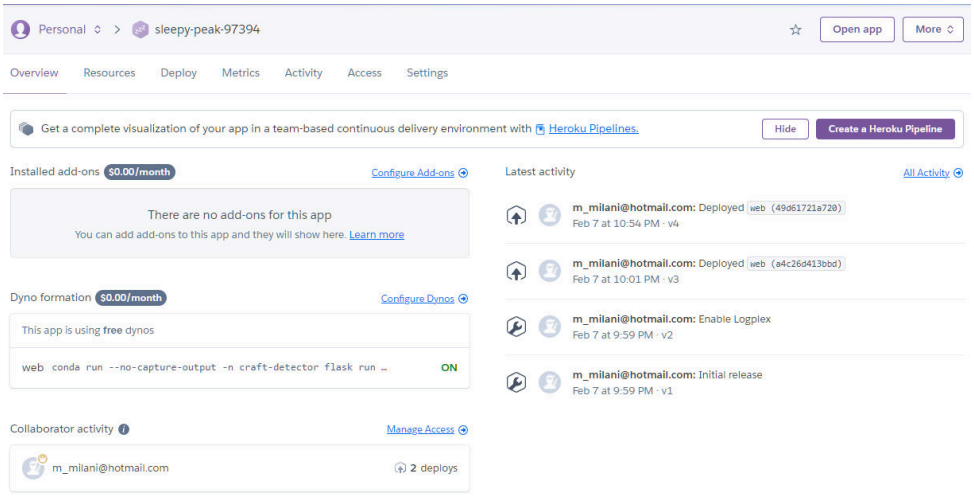

Figura 16 – Página inicial da interface de gestão da aplicação no Heroku.

2. Integração direta com repositórios Git através do Github;

3. Envio da imagem de contêiner.

A terceira opção foi a adotada, devido ao uso de ferramentas de conteinerização. A partir do uso da interface de linha de comando do Heroku, o principal comando são `heroku container:login`, para executar a autenticação com o repositório de imagens de contêineres Docker do Heroku, e o `heroku container:push web`, que utiliza o Docker para construir a imagem de contêiner a partir do arquivo Dockerfile e envia a imagem construída para o repositório de imagens do Heroku. Para iniciar a nova imagem submetida, é necessário usar o comando `heroku container:release web`, fazendo com que o servidor da aplicação comece a executar o contêiner. Mais detalhes e exemplos sobre o processo de implantação no Heroku utilizando Docker podem ser obtidos diretamente na documentação da plataforma[16].

3.5 Considerações finais

Sucintamente, 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 implantada 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.

[16] https://devcenter.heroku.com/articles/container-registry-and-runtime

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 constatar o fato que são imagens de relativa baixa resolução para os padrões atuais, onde as maiores contêm cerca de 315 mil píxeis. 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 da base de dados, em desempenho, todas as imagens foram processadas com sucesso em 36,31 segundos em um ambiente com aceleração gráfica, média de 257 milissegundos 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 30 GB 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 frequência de operação em 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 extenso de imagens, já que o processamento corresponde a uma carga muito alta dependendo do *hardware* onde a solução for executada, visto que a execução em CPU alocou 17,4 GB 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 comprimido que contém os 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 predições positivas (verdadeiros positivos e falsos positivos), ou seja, dentre as predições da solução, qual é a porcentagem de acerto.

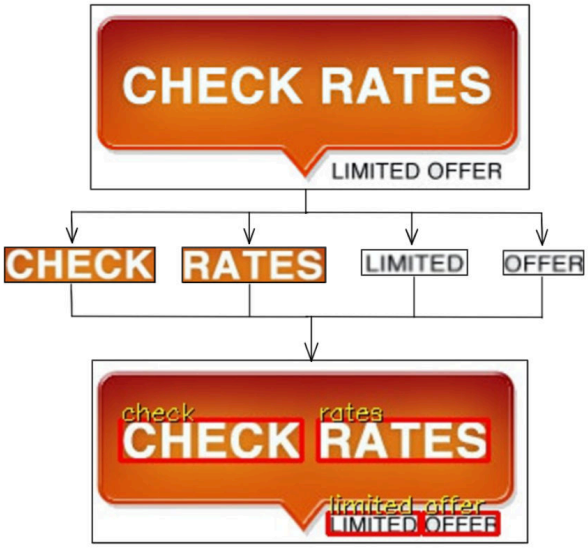

Figura 17 – Exemplo de resultado de reconhecimento. Imagem 52 do conjunto de validação.

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*.

```
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, cerca de 70 tiveram o texto corretamente extraído. É um resultado satisfatório para o trabalho desenvolvido e o nível de simplicidade adotado. Estes números colocariam essa solução em nono lugar entre 18 outros trabalhos submetidos na plataforma de desafios *Robust Reading Competition*.

A Figura 17 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.

Figura 18 – Comparação entre entrada e saída sobre a imagem 5 do conjunto de validação do ICDAR 2011

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.

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 a capacidade de detecção em que não são tão difíceis de observar. Caracteres de pontuação, acentos, marcações como o cifrão ($), entre outros casos, acabam dificultando o acerto do reconhecimento. A Figura 18 exemplifica esta constatação. Nela, é possível observar que o trecho que representa o preço do artigo anunciado na imagem 5 do conjunto de validação, que contém cifrão, vírgula e asterisco, não foi reconhecida com sucesso. Apesar de o método aproximar bem os caracteres desconhecidos, reconhecendo o cifrão ($) 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 pequenas e com pouca definição, o que trouxe dificuldades para o modelo pré-treinado CRNN utilizado. A Figura 19 exemplifica essa dificuldade. Pode-se observar que nenhum dos "títulos" de cada uma das fotos que estão presentes na Figura 19 foram localizadas com sucesso, mas não obtiveram a igualdade entre predição e gabarito (ficaram marcadas com uma área azul).

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 19. Apesar de o reconhecimento ter relativo sucesso nesse exemplo, a localização não conseguiu segmentar corretamente todas as palavras da frase. A generalização da

Figura 19 – Exemplo de imagem com instâncias de texto pequenas em resolução. Imagem 28 do ICDAR 2011

detecção e do reconhecimento para as mais diversas categorias de fontes é um dos principais problemas que motivam o uso de redes neurais profundas para reconhecimento de texto. [12]

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 válido para avaliar a solução. No entanto, ainda não são casos reais de texto em cena, problema que motivou este trabalho. A base de dados ICDAR 2013 contém exemplos mais característicos de texto de cena.

4.1.2 ICDAR 2013

O conjunto de imagens de validação do ICDAR 2013 conta com 233 imagens de tamanhos variados. As menores têm cerca de 640 píxeis de altura e 480 píxeis de comprimento, enquanto as maiores têm dimensões comparáveis à resolução 4K, com 3888 píxeis de altura e 2592 píxeis de largura. 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 píxeis 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 da base de dados, em desempenho, todas as imagens foram processadas com sucesso em 207,74 segundos em um ambiente com aceleração gráfica, utilizando os mesmos recursos computacionais disponíveis na validação do ICDAR 2011, atingindo uma média de 892 milissegundos por imagem. A média de tempo de processamento por imagem é 3,5 vezes maior quando comparado à base de dado 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

Figura 20 – Demonstração de um falso positivo durante a avaliação da solução contra a base de dados ICDAR 2013.

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 maior, cerca de 6 pontos percentuais, indicando mais 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 20 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 decorrentes de reflexos em algumas imagens de texto sob vidro ou com incidência de iluminação de alta intensidade (*flash* de câmeras fotográficas), onde, em alguns exemplos, a detecção foi sub-ótima (Figura 21) e em outros, o reconhecimento não foi o melhor (Figura 22).

Outra dificuldade conhecida de problemas de reconhecimento de texto em cenas está relacionada a palavras em destaque, o que pode ser relativamente comum, dado

Figura 21 – Exemplo de imagem com artefatos que trouxeram dificuldades para a localização correta do texto.

Figura 22 – Exemplo de texto sob vidro, com plano de fundo desafiadores para o reconhecimento.

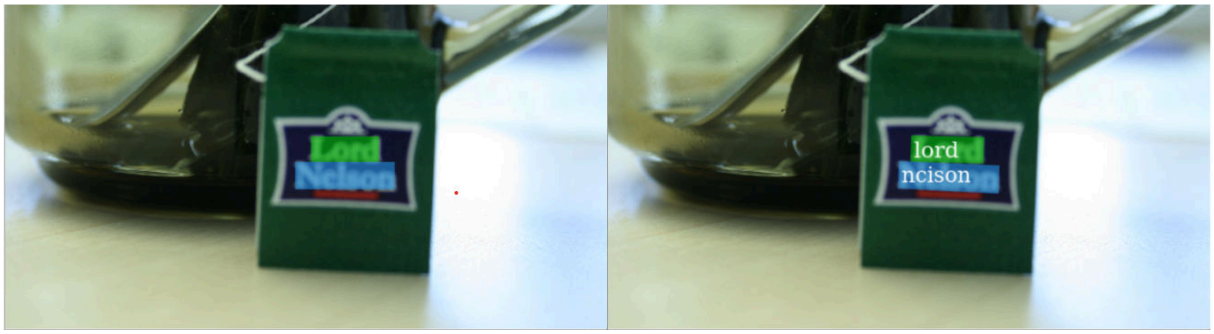

Figura 23 – Exemplo de imagem com texto em desfoque.

o contexto de uma imagem com diversos objetos na cena. Apesar de ser um exemplo mais simples, a Figura 23 demonstra esse caso. A etiqueta na infusão está visivelmente em desfoque e complicou o trabalho do reconhecimento por deixar os caracteres mais ambíguos.

4.1.3 Imagens autorais

Com caráter mais qualitativo, é interessante inferir a qualidade das soluções em imagens fora do contexto das bases de dados conhecidamente utilizadas 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 desafiadoras para a solução, tanto em localização

Figura 24 – Exemplo de imagem autoral onde a solução apresentou dificuldades com texto curvo e estilizado. Textos reconhecidos: “sney” e “intmalakingon”.

Figura 25 – Exemplo de imagem autoral onde uma região de texto longa não foi extraída com sucesso, principalmente no início da palavra, o que dificultou o reconhecimento. Texto reconhecido: “permmusem”.

Figura 26 – Exemplo de imagem autoral com fontes estilizadas que trouxeram dificuldade tanto para a detecção, quanto para o reconhecimento. Textos reconhecidos: “sey” e “fpan”.

quanto em reconhecimento. Imagens com texto bem localizado e visível, alguns letreiros e placas tiveram bons resultados, mas alguns casos de palavras mais longas e curvadas, mesmo que levemente, não tiveram amplo sucesso, especialmente se apresentarem caligrafia estilizada, como é possível observar na Figura 26. 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* [61]), para obter uma imagem com o mínimo de curvatura possível, facilitando o trabalho de reconhecimento.

Um exemplo foi interessante, pois contrariou as expectativas quanto ao reconhecimento, demonstrando o potencial que a solução pode apresentar com algumas iterações de melhorias. A Figura 27 mostra um caso de sucesso na localização e reconhecimento onde o

Figura 27 – Exemplo de reconhecimento com sucesso em condições desafiadoras em imagem autoral. Textos reconhecidos: “disneys”, "electrical" e “parade”.

Figura 28 – 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”.

contexto onde o texto está inserido tem muita informação e a fonte do texto, principalmente do trecho escrito “Disney’s”, é bastante estilizado, e mesmo assim, o reconhecimento foi preciso.

Agora, quando o texto se encontra em situações favoráveis, por exemplo, adequadamente 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 28 demonstra esse caso, onde temos um cartaz de boas-vindas do museu do Apartheid, na África do Sul.

4.2 Disponibilização em nuvem

O resultado da prova de conceito de implantar 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 implantada na plataforma do Heroku com sucesso. No entanto, como constatado na Seção 4.1.1, a demanda de recursos computacionais da

solução integrada foi é relativamente grande, em especial a pressão sobre disponibilidade de memória RAM no sistema é grande.

O ambiente disponibilizado de graça pelo Heroku é limitado em recursos, por questões óbvias, portanto já não era esperado um bom desempenho, sobretudo pelo fato do ambiente não contar com aceleração gráfica.

A partir de alguns testes sobre aplicação implantada, pode-se observar que o ambiente não é de fato estável. Ao requisitar a execução do reconhecimento em imagens menores da base de dados 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

Ao fim de todo o desenvolvimento desse trabalho de graduação a fim de recapitulação, observa-se uma solução capaz de reconhecer textos em fotografias onde as palavras ou instâncias de texto não necessariamente foram a motivação da fotografia e, em geral, não são documentos com texto impresso ou manuscrito em papel. Essas imagens são características do problema de *Scene Text Recognition* (STR).

A partir do re-uso de um detector e um reconhecedor de texto, populares em trabalhos na área, foi possível implementar e avaliar a integração dos dois métodos, obtendo no final uma composição capaz de resolver o problema de STR na maneira dita fim-a-fim, agregando as etapas de detecção e reconhecimento de uma vez só, com capacidade de acertar o reconhecimento em 70% das vezes, o que não é o melhor resultado observado no meio acadêmico, mas considerando que simplificações foram feitas para possibilitar a implementação, é um resultado satisfatório e cumpre o objetivo do trabalho.

Como observado, a solução desenvolvida apresentou dificuldades nos exemplos de avaliação e no ambiente remoto provisionado na nuvem e possibilidades de evolução para trabalhos futuros podem tentar atacá-las, como, por exemplo, refinar os parâmetros do modelo de detecção para melhorar os resultados para textos mais longos e textos não-horizontais, implementar método de transformação da região de texto para melhorar resultados do reconhecimento, treinar um novo modelo de reconhecimento com um dicionário maior de caracteres e até tentar otimizar o uso de recursos computacionais da solução.

Referências

1 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 4 vezes nas páginas 5, 3, 26 e 27.

2 ALZUBAIDI, L. et al. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, v. 8, 2021. Citado 4 vezes nas páginas 5, 6, 7 e 9.

3 BAIMYRZA, D. *Introduction to RNN*. 2021. Disponível em: <https://cs231n.github.io/rnn/>. Citado 2 vezes nas páginas 5 e 10.

4 RAISI, Z. et al. Text detection and recognition in the wild: A review. *ArXiv*, abs/2006.04305, 2020. Citado 6 vezes nas páginas 5, 2, 10, 11, 12 e 16.

5 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 6 vezes nas páginas 5, 9, 13, 14, 15 e 20.

6 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 5 vezes nas páginas 5, 3, 9, 16 e 17.

7 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 3 vezes nas páginas 5, 26 e 27.

8 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.

9 WADHWA, N. et al. Synthetic depth-of-field with a single-camera mobile phone. *ACM Transactions on Graphics (TOG)*, v. 37, p. 1 – 13, 2018. Citado na página 1.

10 LIBA, O. et al. Handheld mobile photography in very low light. *ACM Transactions on Graphics (TOG)*, v. 38, p. 1 – 16, 2019. Citado na página 1.

11 TAUSCHEK, G. *Reading machine*. 1929. 27 maio 1929. Citado na página 1.

12 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 5 vezes nas páginas 2, 10, 11, 16 e 38.

13 PATEL, R. *Giving Lens New Reading Capabilities in Google Go*. 2019. Disponível em: <https://ai.googleblog.com/2019/09/giving-lens-new-reading-capabilities-in.html>. Acesso em: 01 maio 2022. Citado na página 2.

14 ZHOU, X. et al. East: an efficient and accurate scene text detector. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2017. p. 5551–5560. Citado 3 vezes nas páginas 3, 13 e 20.

15 ZHANG, S.-X. et al. *Kernel Proposal Network for Arbitrary Shape Text Detection*. arXiv, 2022. Disponível em: <https://arxiv.org/abs/2203.06410>. Citado na página 3.

16 KIM, S. et al. *DEER: Detection-agnostic End-to-End Recognizer for Scene Text Spotting*. arXiv, 2022. Disponível em: <https://arxiv.org/abs/2203.05122>. Citado na página 3.

17 VASWANI, A. et al. Attention is all you need. *CoRR*, abs/1706.03762, 2017. Disponível em: <http://arxiv.org/abs/1706.03762>. Citado na página 3.

18 SHARMA, N.; SHARMA, R.; JINDAL, N. Machine learning and deep learning applications-a vision. *Global Transitions Proceedings*, v. 2, n. 1, p. 24–28, 2021. ISSN 2666-285X. 1st International Conference on Advances in Information, Computing and Trends in Data Engineering (AICDE - 2020). Disponível em: <https://www.sciencedirect.com/science/article/pii/S2666285X21000042>. Citado na página 5.

19 POUYANFAR, S. et al. A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 51, n. 5, p. 1–36, 2018. Citado 3 vezes nas páginas 5, 8 e 11.

20 KIRANYAZ, S. et al. 1d convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing*, v. 151, p. 107398, 2021. ISSN 0888-3270. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0888327020307846>. Citado na página 6.

21 CHOI, R. Y. et al. Introduction to Machine Learning, Neural Networks, and Deep Learning. *Translational Vision Science & Technology*, v. 9, n. 2, p. 14–14, 02 2020. ISSN 2164-2591. Disponível em: <https://doi.org/10.1167/tvst.9.2.14>. Citado na página 7.

22 NWANKPA, C. et al. Activation functions: Comparison of trends in practice and research for deep learning. *CoRR*, abs/1811.03378, 2018. Disponível em: <http://arxiv.org/abs/1811.03378>. Citado na página 7.

23 CONVOLUTIONAL Neural Networks for Visual Recognition. 2021. Disponível em: <https://cs231n.github.io/convolutional-networks/>. Citado na página 8.

24 LONG, J.; SHELHAMER, E.; DARRELL, T. Fully convolutional networks for semantic segmentation. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2015. Citado na página 8.

25 O'SHEA, K.; NASH, R. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015. Disponível em: <http://arxiv.org/abs/1511.08458>. Citado na página 9.

26 SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015. Citado 2 vezes nas páginas 9 e 13.

27 HE, K. et al. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. Disponível em: <http://arxiv.org/abs/1512.03385>. Citado na página 9.

28 SZEGEDY, C. et al. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. Disponível em: <http://arxiv.org/abs/1409.4842>. Citado na página 9.

29 DENG, J. et al. ImageNet: A Large-Scale Hierarchical Image Database. In: *CVPR09*. [S.l.: s.n.], 2009. Citado na página 9.

30 RUSSAKOVSKY, O. et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, v. 115, n. 3, p. 211–252, 2015. Citado na página 9.

31 TAN, C. et al. A survey on deep transfer learning. *CoRR*, abs/1808.01974, 2018. Disponível em: <http://arxiv.org/abs/1808.01974>. Citado na página 9.

32 PANG, B. et al. Deep RNN framework for visual sequential applications. *CoRR*, abs/1811.09961, 2018. Disponível em: <http://arxiv.org/abs/1811.09961>. Citado na página 9.

33 SCHUSTER, M.; PALIWAL, K. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, v. 45, n. 11, p. 2673–2681, 1997. Citado na página 9.

34 BENGIO, Y.; SIMARD, P.; FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, IEEE, v. 5, n. 2, p. 157–166, 1994. Citado na página 10.

35 HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural computation*, v. 9, p. 1735–80, 12 1997. Citado na página 10.

36 BAEK, J. et al. What is wrong with scene text recognition model comparisons? dataset and model analysis. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, p. 4714–4722, 2019. Citado na página 10.

37 YE, Q.; DOERMANN, D. Text detection and recognition in imagery: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 37, 06 2015. Citado na página 11.

38 EPSHTEIN, B.; OFEK, E.; WEXLER, Y. Detecting text in natural scenes with stroke width transform. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, p. 2963–2970, 2010. Citado na página 12.

39 NEUMANN, L.; MATAS, J. A method for text localization and recognition in real-world images. In: *ACCV*. [S.l.: s.n.], 2010. Citado na página 12.

40 LIU, W. et al. Ssd: Single shot multibox detector. In: SPRINGER. *European conference on computer vision*. [S.l.], 2016. p. 21–37. Citado na página 12.

41 REDMON, J. et al. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 779–788. Citado na página 12.

42 GIRSHICK, R. Fast r-cnn. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. [S.l.: s.n.], 2015. Citado na página 12.

43 LIAO, M. et al. Textboxes: A fast text detector with a single deep neural network. In: *Thirty-first AAAI conference on artificial intelligence*. [S.l.: s.n.], 2017. Citado na página 13.

44 LIAO, M.; SHI, B.; BAI, X. Textboxes++: A single-shot oriented scene text detector. *IEEE transactions on image processing*, IEEE, v. 27, n. 8, p. 3676–3690, 2018. Citado 2 vezes nas páginas 13 e 20.

45 SHI, B.; BAI, X.; BELONGIE, S. Detecting oriented text in natural images by linking segments. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 2550–2558. Citado na página 13.

46 ZHANG, Z. et al. Multi-oriented text detection with fully convolutional networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 4159–4167. Citado na página 13.

47 YAO, C. et al. Scene text detection via holistic, multi-channel prediction. *arXiv preprint arXiv:1606.09002*, 2016. Citado na página 13.

48 DENG, D. et al. Pixellink: Detecting scene text via instance segmentation. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. [S.l.: s.n.], 2018. v. 32, n. 1. Citado 2 vezes nas páginas 13 e 20.

49 LONG, S. et al. Textsnake: A flexible representation for detecting text of arbitrary shapes. In: *Proceedings of the European conference on computer vision (ECCV)*. [S.l.: s.n.], 2018. p. 20–36. Citado na página 13.

50 RONNEBERGER, O.; FISCHER, P.; BROX, T. U-net: Convolutional networks for biomedical image segmentation. In: *MICCAI*. [S.l.: s.n.], 2015. Citado na página 13.

51 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 14.

52 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 14.

53 DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. In: *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*. USA: IEEE Computer Society, 2005. (CVPR '05), p. 886–893. ISBN 0769523722. Disponível em: <https://doi.org/10.1109/CVPR.2005.177>. Citado na página 15.

54 GRAVES, A. et al. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In: *Proceedings of the 23rd international conference on Machine learning*. [S.l.: s.n.], 2006. p. 369–376. Citado na página 16.

55 BAHDANAU, D.; CHO, K.; BENGIO, Y. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2015. Citado na página 16.

56 MOLLOY, D. *The great graphics card shortage of 2020 (and 2021)*. 2021. Disponível em: <https://www.bbc.com/news/technology-55755820>. Acesso em: 15 abril 2022. Citado na página 21.

57 KARATZAS, D. et al. Icdar 2015 competition on robust reading. In: *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. [S.l.: s.n.], 2015. p. 1156–1160. Citado na página 26.

58 IWAMURA, M. et al. Icdar2017 robust reading challenge on omnidirectional video. In: *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. [S.l.: s.n.], 2017. v. 01, p. 1448–1453. Citado na página 26.

59 LUCAS, S. et al. Icdar 2003 robust reading competitions. In: *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.* [S.l.: s.n.], 2003. p. 682–687. Citado na página 26.

60 MERKEL, D. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, v. 2014, n. 239, p. 2, 2014. Citado na página 32.

61 JADERBERG, M. et al. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015. Disponível em: <http://arxiv.org/abs/1506.02025>. Citado na página 41.

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
  - lcms2=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=h27cfd23_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=py37h27cfd23_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=py37h27cfd23_1
- qt=5.9.7=h5867ecd_1
- readline=8.1=h27cfd23_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=py37h27cfd23_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=h27cfd23_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 implantação 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
```