UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
SISTEMAS DE INFORMAÇÃO

João Victor Nery Buchmeier de Oliveira

Método para extração automática de legendas embutidas em vídeos

Florianópolis
2025

João Victor Nery Buchmeier de Oliveira

Método para extração automática de legendas embutidas em vídeos

Trabalho de Conclusão de Curso submetido ao Curso de Graduação em Sistemas de Informação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para obtenção do título de Bacharel em Sistemas de Informação.
Orientador: Prof. Alexandre Gonçalves Silva, Dr.

Florianópolis
2025

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

de Oliveira, João Victor Nery Buchmeier
Método para extração automática de legendas embutidas em vídeos / João Victor Nery Buchmeier de Oliveira ; orientador, Alexandre Gonçalves Silva, 2025.
108 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Sistemas de Informação, Florianópolis, 2025.

Inclui referências.

1. Sistemas de Informação. 2. Visão Computacional. 3. Machine Learning. 4. Optical Character Recognition. 5. Processamento de Imagem. I. Silva, Alexandre Gonçalves. II. Universidade Federal de Santa Catarina. Graduação em Sistemas de Informação. III. Título.

João Victor Nery Buchmeier de Oliveira

Método para extração automática de legendas embutidas em vídeos

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de Bacharel em Sistemas de Informação e aprovado em sua forma final pelo curso de Graduação em Sistemas de Informação.

Florianópolis, 10 de Novembro de 2025.

Prof. Álvaro Junio Pereira Franco, Dr
Coordenador do Curso

Banca Examinadora:

Prof. Alexandre Gonçalves Silva, Dr.
Orientador
Universidade Federal de Santa Catarina

Prof. Elder Rizzon Santos, Dr.
Avaliador
Universidade Federal de Santa Catarina

Prof. Eduardo Camilo Inácio, Dr.
Avaliador
Universidade Federal de Santa Catarina

RESUMO

A área de Processamento de Imagem (PDI), Optical Character Recognition (OCR) e Machine Learning (ML) têm revolucionado a maneira como se lida com informações visuais, transformando imagens em dados legíveis e processáveis. Essas tecnologias desempenham um papel fundamental em diversos setores, desde a automação de processos industriais até o aumento da acessibilidade digital. Nesse sentido, este projeto de TCC propôs o desenvolvimento de um software para extração automática de legendas embutidas em vídeos, comumente denominadas *burned-in subtitles*, que são textos permanentemente inseridos nas imagens, fazendo parte integrante do conteúdo visual. Diferente de legendas convencionais, que podem ser ativadas ou desativadas pelo usuário e armazenadas separadamente como metadados, as *burned-in subtitles* não podem ser removidas ou modificadas. Isso cria desafios significativos, especialmente no contexto de acessibilidade, tradução e organização de conteúdo audiovisual. Dessa forma, utilizando-se das tecnologias citadas, o software se torna capaz de identificar e capturar as legendas diretamente das imagens de vídeo, convertendo-as em texto legível. O resultado é então salvo em arquivos no formato `.srt`, amplamente utilizado para sincronização de legendas em mídias audiovisuais. Nesse sentido, a extração de legendas possibilita outras aplicações, como a tradução das próprias legendas e a sumarização de filmes, facilitando o acesso a conteúdos em diferentes idiomas e permitindo a análise e síntese do conteúdo textual extraído, respectivamente. O trabalho envolveu a pesquisa e a aplicação de técnicas avançadas de PDI, ML e OCR, combinando métodos de segmentação, detecção de texto e reconhecimento óptico de caracteres para garantir uma extração precisa das legendas embutidas. Além disso, foi realizada uma análise crítica das ferramentas e algoritmos existentes, explorando abordagens baseadas em Deep Learning e Processamento de Imagem tradicional, para otimizar o desempenho da solução. Por fim, a avaliação da solução proposta foi feita em cima de 20 vídeos em inglês, utilizando das métricas Character Error Rate (CER) e Word Error Rate (WER), a fim de medir erros a nível de caracteres e palavras das legendas extraídas, respectivamente. Com isso, obteve-se 2,1% de erro para CER e 6,7% para WER no modo *relaxed* de avaliação, enquanto o modo *strict* obteve 2,7% e 9,8%, mostrando a eficácia da solução desenvolvida.

Palavras-chave: Processamento de Imagem. PDI. Optical Character Recognition. OCR. Extração de Legendas.

ABSTRACT

The field of study of Image Processing, Optical Character Recognition (OCR) and Machine Learning (ML) has revolutionized the way visual information is processed, turning images into readable and processable data. These technologies play a fundamental role in various sectors, from industrial automation to improving digital accessibility. In this context, this undergraduate research project proposed the development of a software for automatic extraction of burned-in subtitles from videos. Burned-in subtitles are permanently embedded texts in images, forming an integral part of the visual content. Unlike conventional subtitles, which can be activated or deactivated by the user and stored separately as metadata, burned-in subtitles cannot be removed or modified. This poses significant challenges, especially in the context of accessibility, translation and audiovisual content organization. By combining the mentioned technologies, the software is capable of identifying and extracting subtitles directly from video frames, converting them into readable text. The extracted text is then saved in the `.srt` format, widely used for subtitle synchronization in audiovisual media. This process enables additional applications, such as subtitle translation and film summarization, facilitating access to content in different languages and allowing textual analysis and synthesis. The research focused on the study and application of advanced Image Processing, ML, and OCR techniques, combining segmentation, text detection, and optical character recognition methods to ensure accurate extraction of the subtitles. Additionally, a critical analysis of existing tools and algorithms was conducted, exploring deep learning-based approaches alongside traditional image processing techniques to optimize performance. Ultimately, the proposed solution was evaluated on 20 English videos using the Character Error Rate (CER) and Word Error Rate (WER) metrics, which measure errors at the character and word levels of the extracted subtitles, respectively. As a result, error rates of 2,1% for CER and 6,7% for WER were obtained in the relaxed evaluation mode, while the strict mode achieved 2,7% and 9,8%, demonstrating the effectiveness of the developed solution.

Keywords: Image Processing. Optical Character Recognition. OCR. Machine Learning. ML.

LISTA DE FIGURAS

Figura 1 – Exemplo de imagem digital e mapeamento de níveis de cinza. 15
Figura 2 – Etapas fundamentais do PDI 16
Figura 3 – Aprendizado de Máquina Tradicional vs Deep Learning 18
Figura 4 – Exemplo de uma CNN e suas diferentes camadas 23
Figura 5 – Representação das skip connections 24
Figura 6 – Skip connections na ResNet-50 24
Figura 7 – Pipeline tradicional vs Pipeline DBNet 25
Figura 8 – Comparação entre modelos de detecção de texto no dataset MSRA-TD500 . 26
Figura 9 – Arquitetura RNN 28
Figura 10 – Abordagem metodológica para um sistema OCR 30
Figura 11 – Estrutura da rede CRNN 31
Figura 12 – Metodologia para detecção/reconhecimento de texto 33
Figura 13 – Pipeline do sistema ANPR 35
Figura 14 – Exemplo de funcionamento geral do software 41
Figura 15 – Fluxograma da arquitetura 42
Figura 16 – Metodologia passo a passo 45
Figura 17 – Fluxograma de calibração do posicionamento das legendas 58
Figura 18 – Fluxograma de extração textual de frame 59
Figura 19 – Fluxograma do processamento principal 61
Figura 20 – Exemplo de aplicação do PaddleOCR para legendas brancas 64
Figura 21 – Resultado de aplicação do PaddleOCR para legendas brancas 64
Figura 22 – Exemplo de aplicação do PaddleOCR para legendas amarelas 65
Figura 23 – Resultado de aplicação do PaddleOCR para legendas amarelas 65
Figura 24 – Resultado métricas de avaliação - Relaxed 70
Figura 25 – Resultado métricas de avaliação - Strict 70

LISTA DE ABREVIATURAS E SIGLAS

PDI	Processamento de Imagem
OCR	Optical Character Recognition
IA	Inteligência Artificial
ML	Machine Learning
SVM	Support Vector Machines
KNN	K-Nearest Neighbors
MLP	Multilayer Perceptron
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
CRNN	Convolutional Recurrent Neural Network
CTC	Connectionist Temporal Classification
RGB	Red, Green, Blue
ROI	Region of Interest
FPS	Frames Per Second
CLI	Command Line Interface

SUMÁRIO

1 INTRODUÇÃO 11
1.1 OBJETIVOS 12
1.1.1 Objetivo Geral 12
1.1.2 Objetivos Específicos 12
1.2 MÉTODO DE PESQUISA 13

2 FUNDAMENTAÇÃO TEÓRICA 14
2.1 PROCESSAMENTO DIGITAL DE IMAGENS (PDI) 14
2.2 MACHINE LEARNING (ML) 17
2.2.1 TÉCNICAS TRADICIONAIS DE MACHINE LEARNING 18
2.2.2 DEEP LEARNING 21
2.3 RECONHECIMENTO ÓPTICO DE CARACTERES (OCR) 29
2.3.1 SISTEMA DE OCR TRADICIONAL 29
2.3.2 APLICAÇÕES DE DEEP LEARNING PARA OCR 30

3 TRABALHOS RELACIONADOS 32
3.1 MULTI-LANGUAGE VIDEO SUBTITLE RECOGNITION WITH CONVOLUTIONAL NEURAL NETWORK AND LONG SHORT-TERM MEMORY NETWORKS 32
3.2 AN END-TO-END SYSTEM FOR SUBTITLE TEXT EXTRACTION FROM MOVIE VIDEOS 33
3.3 LICENSE PLATE RECOGNITION SYSTEM USING YOLOV5 AND CNN 34
3.4 EVE: TOWARDS END-TO-END VIDEO SUBTITLE EXTRACTION WITH VISION-LANGUAGE MODELS 36
3.5 IDENTIFICAÇÃO DE PROPRIEDADES DE PRODUTOS A PARTIR DA LEITURA AUTOMÁTICA DE RÓTULOS 38
3.6 COMPARATIVO 39

4 METODOLOGIA DE DESENVOLVIMENTO 41
4.1 RESUMO DA PROPOSTA 41
4.2 FERRAMENTAS E BIBLIOTECAS 42
4.2.1 DETECÇÃO E RECONHECIMENTO TEXTUAL 43
4.2.2 PÓS-PROCESSAMENTO TEXTUAL 43
4.2.3 IMAGEM E VÍDEO 44
4.3 METODOLOGIA 44
4.4 BASE DE VÍDEOS 46
4.5 MÉTRICAS DE AVALIAÇÃO 47

5 DESENVOLVIMENTO 49
5.1 ANÁLISE DE REQUISITOS 49
5.2 TECNOLOGIAS E DEPENDÊNCIAS 50
5.3 AMBIENTE DE EXECUÇÃO E REPRODUTIBILIDADE 50
5.4 ARQUITETURA E ORGANIZAÇÃO DO CÓDIGO 51
5.5 CLASSE CENTRAL E MÓDULOS 51
5.6 ALGORITMOS E HEURÍSTICAS 51
5.6.1 Definição de Atributos e Variáveis Estáticas 52
5.6.2 Métodos Auxiliares 53
5.6.3 Métodos Principais 56
5.7 INTERFACE DE LINHA DE COMANDO (CLI) 62

6 RESULTADOS 63
6.1 EXEMPLO DE APLICAÇÃO 63
6.1.1 Execução em frames com diferentes estilos de legenda 63
6.1.2 Execução completa para vídeo fictício 65
6.2 DESAFIOS E LIMITAÇÕES 66
6.2.1 Textos Ambientais na Região das Legendas 67
6.2.2 Baixo Contraste entre Legenda e Fundo 67
6.3 TESTES E VALIDAÇÃO DO SISTEMA 67
6.3.1 Ambiente de Execução e Reprodutibilidade 67
6.3.2 Métricas CER/WER 68
6.3.3 Tempo de Execução 71
6.3.4 Interpretação dos Resultados 72

7 CONCLUSÃO 75
7.1 CONSIDERAÇÕES FINAIS 75
7.2 TRABALHOS FUTUROS 75

REFERÊNCIAS 77

APÊNDICE A – CÓDIGO-FONTE 82

APÊNDICE B – ARTIGO SBC 93

1 INTRODUÇÃO

O avanço das tecnologias de Processamento Digital de Imagens (PDI), Reconhecimento Óptico de Caracteres (OCR) e Machine Learning (ML) tem proporcionado soluções inovadoras para transformar informações visuais em dados processáveis, facilitando a vida humana em diversos contextos. Contudo, existe um desafio na extração de legendas embutidas em vídeos. Diferentemente das legendas convencionais, essas legendas, denominadas *burned-in subtitles*, são fixadas diretamente no conteúdo visual, tornando a separação do texto e a sua utilização em outros contextos um processo complexo e pouco explorado.

A extração de legendas embutidas possui utilidade em diferentes contextos. Entre eles, destacam-se: a tradução de conteúdos, permitindo que materiais originalmente produzidos em um idioma cheguem a públicos mais amplos; o arquivamento de texto relacionado a vídeos, possibilitando a indexação textual para busca eficiente por termos; e a sumarização de conteúdos, fornecendo resumos automáticos de vídeos. Além disso, esses pontos colaboram para a acessibilidade, pois viabilizam a manipulação e conversão de legendas para diversos públicos, como pessoas surdas.

De forma mais abrangente, as técnicas empregadas neste trabalho podem ser estendidas para outras aplicações práticas. Por exemplo, em sistemas de navegação assistida para pessoas cegas, a extração de informações visuais pode auxiliar no deslocamento seguro ao identificar placas de sinalização ou informações contextuais em ruas e avenidas. Em veículos autônomos, mecanismos semelhantes permitem reconhecer sinais como placas de “STOP” ou instruções de trânsito essenciais para a condução segura.

Como mencionado anteriormente, a necessidade deste trabalho surge também das limitações enfrentadas por pessoas que dependem de recursos como tradução e audiodescrição para compreender conteúdos audiovisuais. Como aponta Pollion (2022), indivíduos surdos ou com perda auditiva recente relataram sentir-se mais incluídos e conectados à sociedade ao assistirem conteúdos com legendas *closed caption*, que oferecem flexibilidade ao serem facilmente removíveis ou editáveis. No entanto, a realidade das legendas embutidas dificulta a criação de versões acessíveis ou adaptadas, pois não podem ser facilmente removidas ou manipuladas. Além disso, Morris et al. (2016) destacam que o uso de *closed captions* pode melhorar a compreensão de conteúdos audiovisuais para estudantes, promovendo inclusão e acessibilidade.

Isso demonstra a relevância de ferramentas que automatizam a extração de legendas embutidas, facilitando a tradução e a preservação de conteúdos. Em vista disso, Aparício et al. (2015) ressaltam que a extração de informações a partir de legendas e roteiros pode gerar resumos automáticos de qualidade, cuja avaliação pode ser realizada por meio de métricas como ROUGE, amplamente utilizadas por sua forte correlação com julgamentos humanos.

Nesse cenário, é notável a importância de desenvolver um software capaz de identificar e capturar automaticamente essas legendas. Desse modo, o PDI desempenha um papel crucial em diversas aplicações, abrangendo desde diagnósticos médicos até monitoramento ambiental e reconhecimento automático. Segundo Gupta (2019), a evolução contínua dessa área se deve

a avanços em técnicas de imagem, aumento do poder computacional e crescente necessidade de aplicações inovadoras. Algumas técnicas conhecidas são a binarização adaptativa, muito utilizada para destacar caracteres em fundos complexos, a transformada de Hough, empregada na detecção de linhas e formas geométricas, e os filtros de convolução, que possibilitam a detecção de bordas e redução de ruídos.

Além disso, a área de ML tem sido fundamental para a detecção de objetos em imagens, permitindo uma análise visual mais precisa. Segundo Khan, Laghari e Awan (2021), técnicas supervisionadas e não supervisionadas são amplamente utilizadas para esse fim. O aprendizado não supervisionado permite a extração de características por meio de técnicas como segmentação por histogramas e *clustering*, a exemplo do algoritmo Fuzzy C-means. Já o aprendizado supervisionado utiliza dados rotulados para melhorar a precisão na identificação de objetos, a exemplo de técnicas como Support Vector Machines (SVM) e Convolutional Neural Networks (CNNs), tornando possível a implementação de sistemas robustos para análise visual em áreas como o reconhecimento de padrões.

Por último, a tecnologia denominada OCR é amplamente utilizada para a conversão eletrônica de texto impresso ou digitado em texto legível por máquina. Segundo Kumar et al. (2020), o processo de OCR envolve três etapas principais: digitalização do documento, identificação do texto e armazenamento do resultado em um formato editável e pesquisável. Esse método é amplamente aplicado para a digitalização de documentos oficiais, recibos, e-mails empresariais e extratos bancários, tornando a informação mais acessível e estruturada. Dessa forma, espera-se que, unindo as tecnologias citadas juntamente ao processamento de OCR, seja possível extrair legendas embutidas com alta precisão e convertê-las em um formato legível, como a extensão .srt, amplamente utilizada para sincronização de legendas, contribuindo para acessibilidade, preservação e tradução de conteúdos.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Desenvolver um software para extração automática de legendas embutidas (*burned-in subtitles*) em vídeos, utilizando técnicas de ML, PDI e OCR para a conversão das legendas em formato `.srt`. Com isso, espera-se contribuir para a acessibilidade de conteúdos, além de facilitar o arquivamento de texto relacionado a eles, possibilitando a indexação e sumarização dos conteúdos desses vídeos.

1.1.2 Objetivos Específicos

- Analisar o estado da arte em algoritmos de Machine Learning, Processamento de Imagem e OCR;

- Analisar trabalhos correlatos;
- Propor um software em Python integrando técnicas de ML, PDI e OCR para o reconhecimento e extração de legendas em vídeo;
- Analisar os resultados obtidos com legendas extraídas manualmente, avaliando erros e acertos.

1.2 MÉTODO DE PESQUISA

O presente trabalho caracteriza-se como uma pesquisa aplicada e exploratória, com o objetivo de desenvolver uma ferramenta para extração de legendas embutidas em vídeos, utilizando técnicas de ML, PDI e OCR. O desenvolvimento será realizado em etapas organizadas da seguinte forma:

- **Etapa 1** - Fundamentação Teórica (seção 2): Realizar um levantamento teórico sobre técnicas de ML, PDI e OCR, abordando casos semelhantes do uso dessas tecnologias. Serão estudados artigos, dissertações e soluções open-source já existentes, com ênfase em ferramentas como TesseractOCR.
- **Etapa 2** - Trabalhos Relacionados (seção 3): Revisar trabalhos acadêmicos que envolvam aplicações de ML, PDI e OCR. Essa etapa guiará a escolha das técnicas e ferramentas mais adequadas para o projeto.
- **Etapa 3** - Desenvolvimento do Software (seção 5): Implementar o software em Python, integrando bibliotecas para o reconhecimento e extração de texto em vídeos. Serão utilizados vídeos com legendas de diferentes fontes, tamanhos e cores para garantir a robustez da aplicação.
- **Etapa 4** - Testes e Validação (seção 6): Aplicar testes funcionais com diversos vídeos para avaliar a precisão da extração das legendas. Comparar os resultados com legendas extraídas manualmente, identificando acertos e erros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo descreve os principais conceitos e tecnologias necessárias para a compreensão do trabalho e que serviram de embasamento para o desenvolvimento do software proposto. Assim, são abordados conceitos da Ciência da Computação que englobam assuntos sobre as áreas de Processamento Digital de Imagens (PDI), Machine Learning (ML), Deep Learning e Optical Character Recognition (OCR).

2.1 PROCESSAMENTO DIGITAL DE IMAGENS (PDI)

O Processamento Digital de Imagens (PDI) é uma área da ciência da computação que se dedica ao estudo e manipulação de imagens digitais com o objetivo de realçar informações, extrair características ou facilitar a interpretação automatizada por algoritmos. O resultado de um processamento pode ser uma nova imagem ou um conjunto de características extraídas. Entre as aplicações dessa área, estão a visão computacional, reconhecimento facial, realidade aumentada, reconhecimento óptico de caracteres (OCR), entre outros. Tais aplicações evidenciam a importância do PDI para diversas disciplinas, como medicina, transporte e segurança (Benchamardimath; Hegadi, 2014).

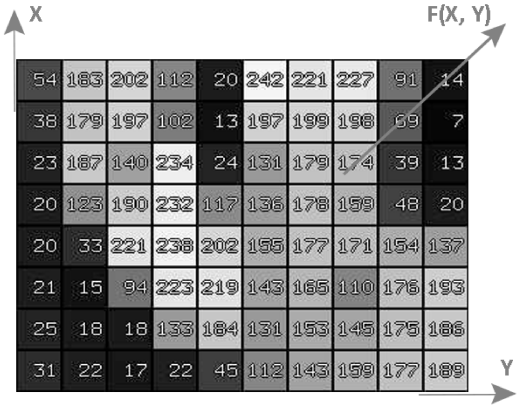
Uma imagem digital pode ser entendida como uma função bidimensional $f(x,y)$, onde x e y representam as coordenadas espaciais da imagem e $f(x,y)$ indica a intensidade de informações como intensidade ou cor naquele ponto (Benchamardimath; Hegadi, 2014). Tecnicamente, uma imagem digital é representada por uma matriz bidimensional de inteiros, onde cada valor corresponde a um pixel. Para uma imagem de 8 bits, por exemplo, os valores de intensidade variam entre 0 e 255, indicando tons de cinza que vão do preto ao branco (Singh; Sehgal, 2022). Já em imagens coloridas, cada pixel pode ser composto por três valores separados, correspondendo aos canais de cor Vermelho (R), Verde (G) e Azul (B), compondo 3 bytes no total, sendo cada um para cada valor do modelo RGB.

Assim, visando compreender as figuras a seguir, sabe-se que a primeira imagem representa como uma imagem monocromática é representada na forma de uma matriz de pixels, ilustrando os eixos $x \in [0, h-1]$, $y \in [0, w-1]$, onde h e w representam a altura e a largura da imagem, respectivamente. Além disso, os valores de intensidade de brilho ou cor são dados por $f(x,y) \in [0,255]$ para imagens de 8 bits. Em seguida, a figura 1 exemplifica uma imagem digital em níveis de cinza, acompanhada de uma ilustração dos valores dos níveis de cinza da região de interesse destacada na imagem do câmera.

Figura 1 – Exemplo de imagem digital e mapeamento de níveis de cinza.



(a) Imagem digital em níveis de cinza

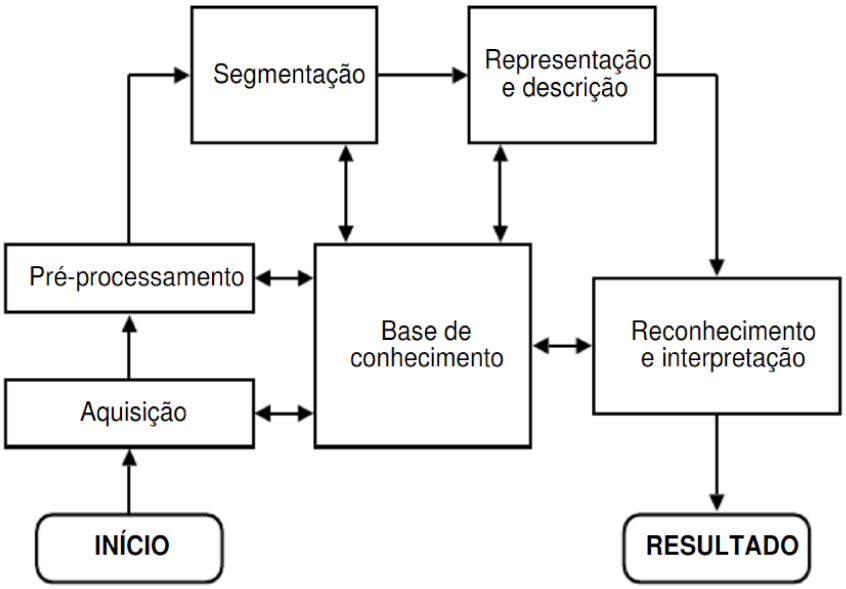


(b) Valores dos níveis de cinza da região da orelha do câmera

Fonte: Adaptado de Silva (2003) e Singh e Sehgal (2022)

O uso do PDI visa justamente manipular essas matrizes para melhorar a visualização humana ou permitir que máquinas extraiam informações significativas. Além disso, o uso de técnicas dessa área ocasiona menor perda de dados de imagens após pós-processamento para transmissão, armazenamento e percepção de máquina (Singh; Sehgal, 2022). Entre as diversas técnicas existentes, algumas que mostram-se interessantes para o campo de OCR são métodos como limiarização, suavização, detecção de bordas, análise morfológica e transformações de frequência, usadas para destacar as regiões de interesse de uma imagem e reduzir ruídos visuais. Para visualizar melhor, a figura 2 ilustra, de forma geral, os passos fundamentais da área de Processamento de Imagens (PDI).

Figura 2 – Etapas fundamentais do PDI



Fonte: Korting (2007)

Além dos conceitos bases, é importante entender que a aplicação do PDI compreende uma série de etapas fundamentais interligadas. Essas etapas contribuem para a análise e interpretação de imagens em diversos contextos. Conforme apresentado por Gonzalez e Woods (2018), o fluxo típico de processamento inicia-se com a aquisição da imagem, que pode ser descrita simplesmente como o fornecimento de uma imagem em formato digital. Esse primeiro passo frequentemente envolve um pré processamento, como dimensionamento ou correção de distorções.

A etapa de melhoria de imagem refere-se à manipulação de imagens para que o resultado seja mais adequado a uma aplicação específica, podendo incluir técnicas como realce de contraste, remoção de ruído e suavização. Essas técnicas de melhoria variam de acordo com o problema a se resolver, tendo influência do que o autor da aplicação considera como um bom resultado. Já a etapa de restauração de imagem, embora também busque a melhoria visual, baseia-se em modelos matemáticos ou estatísticos de degradação de imagens e, com isso, busca reconstruir a imagem original de maneira mais objetiva (Gonzalez; Woods, 2018).

Outras etapas relevantes incluem o processamento morfológico, usado principalmente para extrair componentes de uma imagem que são relevantes na representação e descrição de formas e estruturas, e a segmentação, que consiste na divisão da imagem em regiões de interesse, como separar texto do fundo em uma imagem. Após a segmentação, quase sempre vêm a etapa de extração de *features*, nas quais são extraídas características relevantes da imagem, como contornos, texturas ou padrões (Gonzalez; Woods, 2018).

A etapa de reconhecimento de padrões, por sua vez, é onde ocorre a tomada de decisão automatizada baseada nas *features* extraídas. Em sistemas de OCR, por exemplo, essa

etapa corresponderia à identificação dos caracteres ou palavras presentes em uma imagem segmentada. Técnicas auxiliares como compressão, processamento multirresolução com wavelets e processamento de imagens coloridas também são aplicadas dependendo da natureza do problema (Gonzalez; Woods, 2018).

Ao final do fluxo, o sistema pode produzir uma imagem processada ou um conjunto de atributos úteis para tarefas posteriores, como a classificação de padrões. Por fim, a estrutura modular desse processo é sustentada por uma base de conhecimento, que guia a escolha das técnicas e parâmetros em cada etapa, ajustando o uso das etapas conforme necessidades específicas da aplicação (Gonzalez; Woods, 2018).

2.2 MACHINE LEARNING (ML)

Machine Learning (ML) é um subcampo da inteligência artificial que se baseia na ideia de que sistemas podem aprender com dados, identificar padrões e tomar decisões com mínima intervenção humana (Rathore; Mannepalli, 2021). Essa área do conhecimento tem várias aplicações na sociedade, como em linhas de produção, saúde, educação, transporte, entre outros. O objetivo do ML e da visão computacional é conferir à computadores a habilidade de obter dados, compreendê-los e tomar decisões baseado em resultados passados e presentes (Mahadevkar et al., 2022).

Seguindo essa ideia, a área de visão computacional, que permite que computadores extraiam e compreendam informações visuais, testemunhou um grande avanço devido a sua integração com técnicas de ML. A combinação de algoritmos avançados e alta disponibilidade de dados proporcionou grande progresso tecnológico para detecção de objetos, segmentação de imagem, detecção facial, entre outros (Zhang et al., 2023).

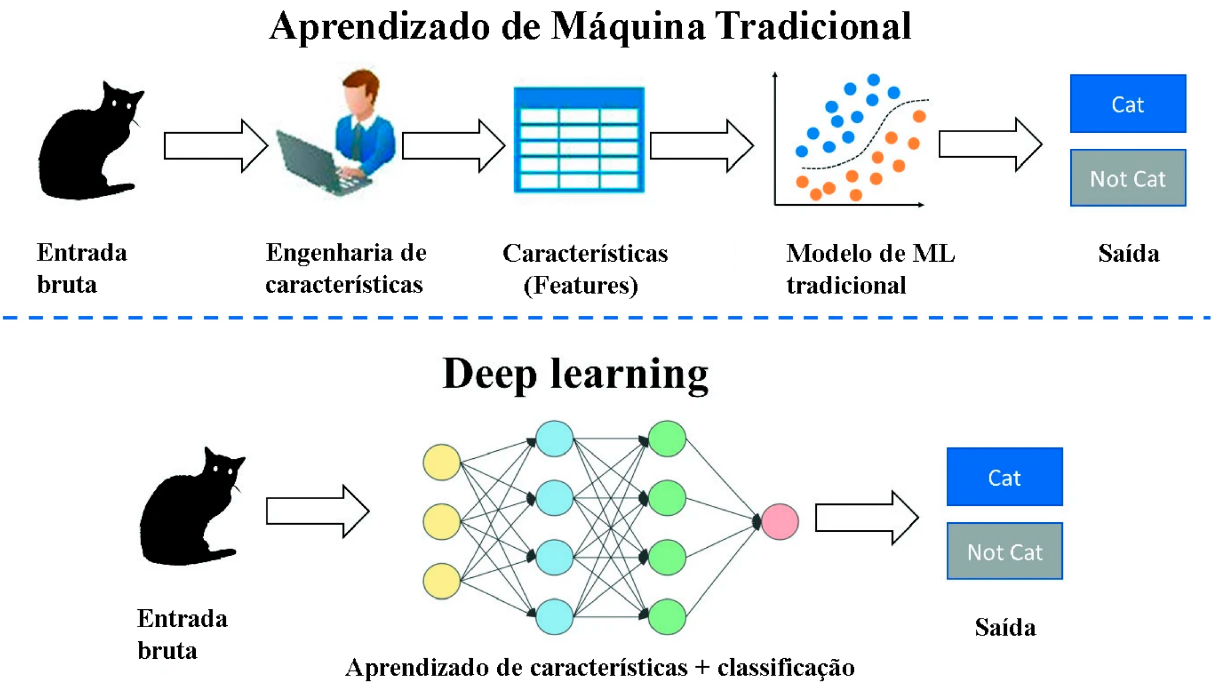
Existem diversos estilos de aprendizagem para atender as diferentes formas de interação entre dados e modelos. Entre eles, os principais são o aprendizado supervisionado e não supervisionado (Rathore; Mannepalli, 2021). De forma geral, o aprendizado supervisionado consiste no uso de dados rotulados para que o modelo aprenda a mapear entradas e saídas específicas, sendo muito utilizado em tarefas como classificação e regressão (Mahadevkar et al., 2022). Com isso, sua aplicação é eficaz em tarefas como reconhecimento de objetos e detecção de padrões visuais em imagens.

Por outro lado, o aprendizado não supervisionado trabalha com dados não rotulados, nos quais o computador executa tarefas complexas que buscam descobrir padrões ocultos sobre os dados e realizar sua própria categorização (Zhang et al., 2023). Em contraste com o supervisionado, esse tipo de aprendizado requer mais poder computacional e tempo de processamento para identificar estruturas nos dados (Zhang et al., 2023).

Mais recentemente, o avanço das técnicas de Deep Learning proporcionou melhorias significativas na extração de *features* em imagens, superando a abordagem manual utilizada em técnicas clássicas de ML. Esse progresso se deve, principalmente, à capacidade dos modelos modernos de aprender representações diretamente a partir de grandes volumes de dados rotu-

lados, tornando o processo mais automatizado e eficaz (Wanjari; Verma, 2025). Diante disso, torna-se relevante apresentar as técnicas tradicionais que serviram de base para esse progresso, assim como discutir o estado da arte atual, baseado em Deep Learning. A figura 3 ilustra bem a diferença em relação ao processo de aprendizado quanto à uma imagem, por exemplo.

Figura 3 – Aprendizado de Máquina Tradicional vs Deep Learning



Fonte: Adaptado de (Wang; Cao, 2021)

2.2.1 TÉCNICAS TRADICIONAIS DE MACHINE LEARNING

Antes do expressivo crescimento das técnicas de Deep Learning, o campo de visão computacional foi sustentado por técnicas clássicas de ML, que continuam sendo relevantes em muitos cenários, especialmente quando há limitação de dados ou recursos computacionais. Essas técnicas foram muito usadas para tarefas como classificação de objetos, reconhecimento de padrões, segmentação de imagens e detecção de anomalias (Mahadevkar et al., 2022). Nesse sentido, algoritmos como K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Árvores de Decisão e também Redes Neurais, são exemplos usados majoritariamente para tarefas de classificação, enquanto métodos de agrupamento como K-Means se destacaram no uso para segmentação de imagem (Mahadevkar et al., 2022).

- **K-Nearest Neighbors (KNN)**

  Um dos algoritmos mais simples e eficazes em tarefas de classificação supervisionada é o K-Nearest Neighbors (KNN). Essa técnica baseia-se na ideia de que dados similares

estão próximos uns dos outros em um espaço de *features*. Dado um novo ponto a ser classificado, o algoritmo calcula a distância entre esse ponto e os demais dados do conjunto de treinamento, selecionando os "k" vizinhos mais próximos, sendo "k" uma constante (Zhang et al., 2023). A classe mais comum entre esses vizinhos é então atribuída ao novo ponto. Em algumas variações, os votos de cada vizinho podem ser ponderados com base na distância, de forma que vizinhos mais próximos tenham maior influência na decisão final (Zhang et al., 2023).

- **Support Vector Machines (SVM)**

  O Support Vector Machine (SVM) é um algoritmo supervisionado muito utilizado para tarefas de classificação, sendo reconhecido por sua alta capacidade de generalização mesmo sem conhecimento prévio sobre os dados (Sharma; Kaushik; Gondhi, 2020). A ideia central do SVM é encontrar um hiperplano que maximize a margem entre as classes em um espaço de *features*, permitindo uma separação clara entre os dados (Mahadevkar et al., 2022). Para tratar problemas não linearmente separáveis, o SVM utiliza funções de *kernel* que projetam os dados em um espaço de maior dimensão onde a separação se torna possível. Entre os kernels mais comuns utilizados estão o linear, o tangente hiperbólico, o polinomial e o RBF (Radial Basis Function), também conhecido como kernel Gaussiano (Sharma; Kaushik; Gondhi, 2020). Devido à sua robustez e capacidade de lidar com alta dimensionalidade, o SVM continua sendo uma escolha eficiente para classificação em visão computacional.

- **Árvores de Decisão**

  As Árvores de Decisão representam uma técnica bastante utilizada quando o objetivo é realizar inferências baseadas em dados. Seu funcionamento parte do princípio de que os dados carregam consigo uma quantidade de informação mensurável, chamada de entropia, e que, por meio dessa medida, é possível selecionar os melhores classificadores para representar o conhecimento do conjunto de dados (Almeida; Lopes; Akamine, 2023). O passo inicial é o cálculo da entropia dos dados, considerando casos positivos e negativos com base em afirmações preliminares. Em seguida, escolhe-se aquela com maior ganho de informação para iniciar a árvore. A partir disso, a árvore é construída hierarquicamente, onde os nós representam condições de decisão baseadas nas variáveis do problema e os galhos indicam os possíveis caminhos de resposta (Almeida; Lopes; Akamine, 2023).

- **Random Forest**

  Além das Árvores de Decisão individuais, uma abordagem amplamente utilizada é o Random Forest, um algoritmo supervisionado que apresenta alta acurácia e robustez, especialmente em tarefas de classificação. Ele é particularmente eficaz em cenários com dados ruidosos e relações não lineares entre as variáveis (Jayalakshmi et al., 2025).

Essa técnica pertence à categoria de métodos de aprendizado em conjunto, conhecido como *ensemble learning*, combinando diversas Árvores de Decisão para formar um modelo mais estável e generalizável (Wibawa et al., 2024). O funcionamento do Random Forest baseia-se na construção de múltiplas árvores a partir de subconjuntos aleatórios do conjunto de dados e das variáveis disponíveis, em um processo conhecido como *bagging*. Durante a etapa de predição, cada árvore vota na saída prevista, sendo o resultado final determinado pela média das previsões (em problemas de regressão) ou pela maioria dos votos (em tarefas de classificação) (Wibawa et al., 2024).

Classificadores baseados nessa abordagem demonstram-se eficazes em problemas com alta dimensionalidade e complexidade, além de oferecerem estimativas da importância das variáveis. Esse recurso é particularmente útil em aplicações de visão computacional, auxiliando na identificação de *landmarks*, ou seja, pontos ou características relevantes em imagens (Wibawa et al., 2024).

- **K-Means**

  K-Means é um algoritmo de *clustering* não-supervisionado amplamente utilizado em tarefas de segmentação de imagem no contexto da visão computacional. Ele opera agrupando os dados, como os pixels de uma imagem, em "k" grupos distintos com base na similaridade de características como cor, por exemplo (Mahadevkar et al., 2022). Para realizar essa divisão, é comum utilizar uma métrica de distância, como a Euclidiana, para medir a proximidade entre os elementos (Brisinello et al., 2018). Segundo o mesmo autor, o K-Means pode ser aplicado diretamente na separação de regiões textuais em imagens com fundo colorido, permitindo a extração eficiente de texto ao segmentar os pixels em diferentes clusters com base em seus valores de cor, por exemplo.

- **Redes Neurais**

  - **Rede Neural Artificial**

    Segundo Haykin (2009), uma rede neural é um processador distribuído massivamente paralelo, composto por unidades de processamento simples, que possui uma propensão natural para armazenar conhecimento experimental e torná-lo disponível para uso. Assim, baseado em hipóteses como essas sobre o funcionamento do cérebro humano, Russell e Norvig (2009) definiu as Redes Neurais Artificiais (RNAs) como modelos computacionais projetados para modelar a forma como cérebro realiza uma tarefa. As RNAs são representadas por redes de unidades interconectadas, os neurônios artificiais, que processam sinais numéricos com base em suas conexões e pesos.

    Cada unidade em uma RNA recebe entradas, aplica pesos e funções de ativação, e transmite sua saída a unidades posteriores. Isso permite que redes neurais se comportem como classificadores, com capacidade de representar funções não lineares complexas (Russell; Norvig, 2009).

Nesse sentido, sabe-se que uma rede neural artificial típica é estruturada em camadas: Uma camada de entrada, onde os dados são recebidos; Uma ou mais camadas ocultas, onde o processamento ocorre e os padrões são aprendidos; Uma camada de saída, onde a resposta final é produzida (Haykin, 2009). Tais estudos tornaram-se a base para modelos mais sofisticados, como as redes neurais profundas (Deep Learning) e também os Perceptrons Multicamadas (MLPs).

– **Perceptron Multicamadas (MLP)**

O Perceptron Multicamadas (MLP) é uma arquitetura de rede neural *feedforward* composta por múltiplas camadas de neurônios: uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. Essa arquitetura surgiu como uma resposta às limitações de modelos neurais de camada única como o Perceptron de Rosenblatt, que servia para problemas linearmente separáveis somente (Haykin, 2009).

Dessa forma, o MLP se diferenciou por possuir algumas características fundamentais, sendo elas: o uso de funções de ativação não lineares e diferenciáveis, permitindo que a rede aborde problemas mais complexos; a presença de camadas ocultas, que atuam como extratoras automáticas de *features*; uma rede com alto grau de conectividade, controlada pelos seus pesos sinápticos. Essa estrutura permite que o MLP represente funções complexas e não lineares. O treinamento é geralmente realizado pelo algoritmo de retropropagação (backpropagation), que ajusta os pesos das conexões minimizando o erro entre a saída esperada e a saída produzida pela rede (Haykin, 2009).

No contexto da visão computacional, os MLPs são excelentes em capturar características espaciais de imagens, sendo empregadas em aplicações como reconhecimento de objetos e classificação de imagens (Nimma et al., 2024). Uma das primeiras CNNs a usarem MLPs para extração de *features* e classificação foi a LeNet-5. Após isso, MLPs destacaram-se em tarefas de reconhecimento de imagem também em arquiteturas como VGGNet e AlexNet (Nimma et al., 2024).

2.2.2 DEEP LEARNING

Deep Learning é uma subárea do Machine Learning que utiliza redes neurais profundas, compostas por múltiplas camadas ocultas. As aplicações dessa área de estudo são usadas para tarefas complexas como reconhecimento de texto, classificação de objetos e visão computacional (Khattar; Sheenam, 2023). Por conta da complexidade de suas tarefas, essa área de estudo requer um grande número de recursos para treinamento (Mahadevkar et al., 2022). Nas aplicações de visão computacional, os modelos empregam blocos estruturais específicos como camadas convolucionais, operações de *pooling* e camadas de normalização, que exploram a natureza espacial dos dados de imagem e reduzem drasticamente o número de parâmetros

aprendidos. Além disso, componentes como funções de perda e funções de ativação não lineares auxiliam algoritmos a extrair padrões de dados e produzir previsões para dados novos (Khattar; Sheenam, 2023). Com isso, uma das grandes vantagens de arquiteturas baseadas em Deep Learning é a habilidade de realizar extração automática de características, eliminando a necessidade de etapas manuais presentes nas abordagens tradicionais. Dessa forma, essa área do conhecimento trouxe avanços significativos em tarefas como classificação de imagens, segmentação semântica e detecção de objetos, por exemplo (Sharma; Kaushik; Gondhi, 2020).

- **Convolutional Neural Networks (CNNs)**

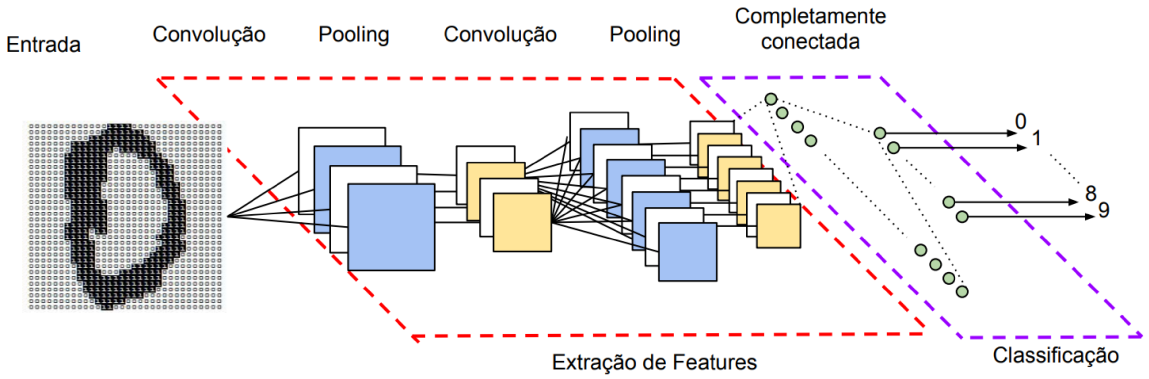
  As Redes Neurais Convolucionais (CNNs) são uma das arquiteturas mais populares do Deep Learning, com ampla aplicação em visão computacional. Suas tarefas incluem detecção de objetos, classificação de imagens, reconhecimento de caracteres, processamento de linguagem natural, entre outras (Sharma; Kaushik; Gondhi, 2020). Uma das principais vantagens das CNNs é que seus modelos não exigem extração manual de *features*, operando diretamente com os pixels brutos das imagens como entrada e aprendendo a realizar a classificação por conta própria (Sharma; Kaushik; Gondhi, 2020).

  Essas redes são compostas por nós de processamento robustos e profundos, cuja popularidade se intensificou graças à maior disponibilidade de dados e de poder computacional (Mahadevkar et al., 2022) dos dias atuais. Inspiradas no funcionamento dos neurônios humanos, as CNNs demandam pouco pré-processamento, sendo bastante eficazes em diversas tarefas de análise de imagens (Manocha; Tewari, 2021).

  Segundo Manocha e Tewari (2021), uma CNN é estruturada em três tipos principais de camadas: convolucionais, de *pooling* e totalmente conectadas. As camadas convolucionais realizam a operação de convolução com filtros (ou *kernels*), gerando *feature maps* que capturam padrões visuais relevantes da imagem. As camadas de *pooling*, como *max pooling* e *average pooling*, reduzem a dimensionalidade desses mapas, o que melhora a generalização do modelo. Juntas, essas camadas compõem o bloco de extração de *features* (*feature extractor*). Já as camadas totalmente conectadas são responsáveis pela classificação com base nas representações extraídas. Arquiteturas pioneiras como LeNet-5, AlexNet e VGGNet fizeram uso extensivo de MLPs em suas camadas de decisão final, evidenciando a utilidade dessa estrutura para tarefas de visão computacional (Nimma et al., 2024).

  O treinamento de uma CNN é orientado por uma função de perda, que quantifica o erro entre a saída prevista e a esperada, e por otimizadores como Adam e RMSProp, que ajustam os pesos da rede de maneira eficiente e adaptativa, promovendo melhorias contínuas na acurácia do modelo (Manocha; Tewari, 2021). A figura 4 ilustra um exemplo de estrutura de uma CNN.

Figura 4 – Exemplo de uma CNN e suas diferentes camadas



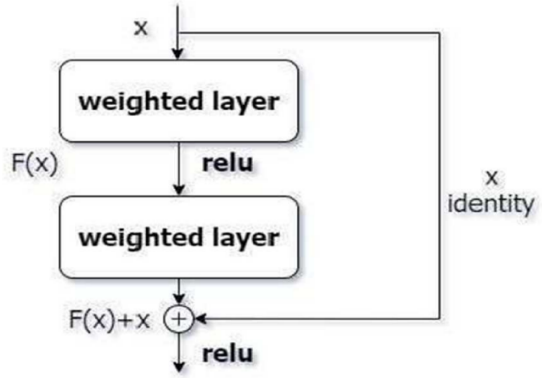
Fonte: (Vargas; Paes; Vasconcelos, 2016)

– **ResNet-50**

O ResNet é uma arquitetura de CNN profunda apresentada em 2015 por Kaiming He e sua equipe da Microsoft Research Asia. Ela introduziu o conceito inovador de *blocos residuais* (*residual blocks*), com as chamadas *skip connections*, permitindo a construção de redes com mais de 100 camadas treináveis, contornando a resistência imposta pelo problema *vanishing gradient*, fenômeno que dificultava o aprendizado em redes profundas tradicionais (He et al., 2016).

A motivação principal por trás do ResNet foi uma observação empírica que contradizia a crença de que redes mais profundas sempre trariam maior capacidade de aprendizado. Na prática, redes muito profundas apresentavam pior desempenho. Para resolver isso, os autores introduziram as *skip connections*, que facilitam a propagação dos gradientes ao permitir que informações fluam diretamente de camadas iniciais para camadas mais profundas, evitando a degradação do aprendizado (Arun et al., 2024). Na figura 5, é possível ver como é feito essa passagem de informação do *input* para camadas posteriores por meio das *skip connections*.

Figura 5 – Representação das skip connections

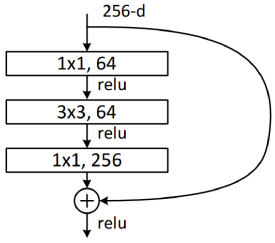


Fonte: (He et al., 2016)

A ResNet-50 é uma variação dessa arquitetura com 50 camadas, composta da seguinte forma:

* As camadas convolucionais começam a operação por meio da extração de *features* da imagem que vem como *input*, repassando isso para os blocos residuais (Arun et al., 2024).
* Os blocos residuais são compostos por duas camadas convolucionais seguidas de funções de ativação ReLU e batch normalization em cada bloco. Esses blocos, então, repassam os dados melhorados para o próximo bloco da arquitetura (Arun et al., 2024).
* A ResNet-50 possui *skip connections* modificadas, que no caso dela, pulam três camadas (ao contrário das versões menores que pulam duas), o que permite capturar *features* hierárquicas mais complexas da imagem, como pode ser visto na figura 6 (Arun et al., 2024).

Figura 6 – Skip connections na ResNet-50



Fonte: (He et al., 2016)

* Por fim, as camadas totalmente conectadas ao final da rede são responsáveis por mapear as *features* aprendidas para as classes alvo da tarefa (Arun et al., 2024).

Assim, a ResNet-50 compõe uma arquitetura integrada de partes convolucionais, residuais e totalmente conectadas, tornando-a altamente eficaz para tarefas de visão

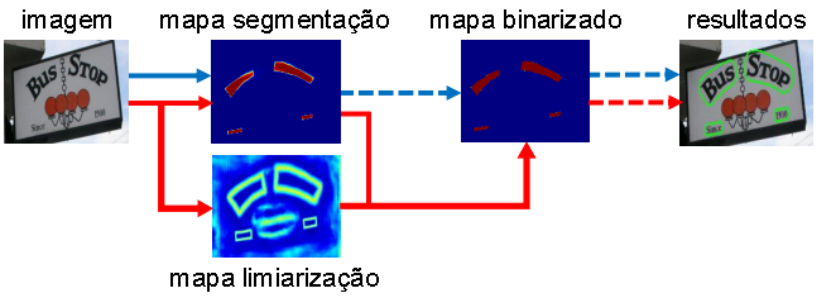
computacional, como classificação de imagens e detecção de objetos, por exemplo. Sua adoção ampla é atribuída à sua capacidade de generalização, profundidade treinável e desempenho robusto em bases de dados complexas, como ImageNet.

– **DBNet**

A Differentiable Binarization Network (DBNet) é uma arquitetura de detecção de texto baseada em segmentação, projetada para lidar com a complexidade de detectar textos em cenas reais, especialmente aqueles com formatos irregulares, como textos curvados ou inclinados. Abordagens baseadas em segmentação ganharam popularidade por sua flexibilidade, mas geralmente requerem uma etapa de pós-processamento crucial, que é a binarização de mapas de probabilidade para transformar os resultados da segmentação em *bounding boxes* ou regiões úteis (Liao et al., 2020).

Nesse contexto, a principal inovação da DBNet está na introdução do módulo de Differentiable Binarization (DB), que permite integrar esse processo diretamente na arquitetura da rede, tornando a binarização parte do processo de otimização do modelo. Isso possibilita que a rede aprenda de forma adaptativa os limiares ideais de binarização durante o treinamento, eliminando a necessidade de uma etapa separada e manual de pós-processamento (Liao et al., 2020). A figura 7 ilustra como a DBNet funciona em comparação a um pipeline tradicional para o mesmo objetivo.

Figura 7 – Pipeline tradicional vs Pipeline DBNet. As linhas azuis representam o pipeline tradicional e as vermelhas o do DBNet; As setas pontilhadas representam os operadores de inferência e as setas preenchidas representam operadores diferenciáveis tanto no treinamento quanto na inferência.



Fonte: Adaptado de Liao et al. (2020)

A validação desse modelo foi realizada utilizando-se cinco bases de dados de *benchmark*, comprovando os grandes avanços de performance da DBNet e demonstrando resultados satisfatórios tanto em acurácia de detecção quanto velocidade. Por exemplo, utilizando um backbone leve como o ResNet-18, a DBNet atingiu um F-measure de 82,8 com uma taxa de 62 *frames* por segundo (FPS) no *dataset* MSRA-TD500, evidenciando sua eficiência computacional e robustez em cenários reais. Além disso, utilizando o ResNet-50 como backbone, atingiu-se maior F-measure, mas menor taxa de FPS (Liao et al., 2020).

Dessa forma, tais características tornam a DBNet uma escolha promissora para tarefas como a detecção de legendas embutidas, onde a velocidade e a precisão são igualmente relevantes. Na figura 8, é possível ver a comparação entre acurácia e velocidade entre alguns modelos de detecção de texto com o DBNet, utilizando a métrica F-measure, definida abaixo (Liao et al., 2020).

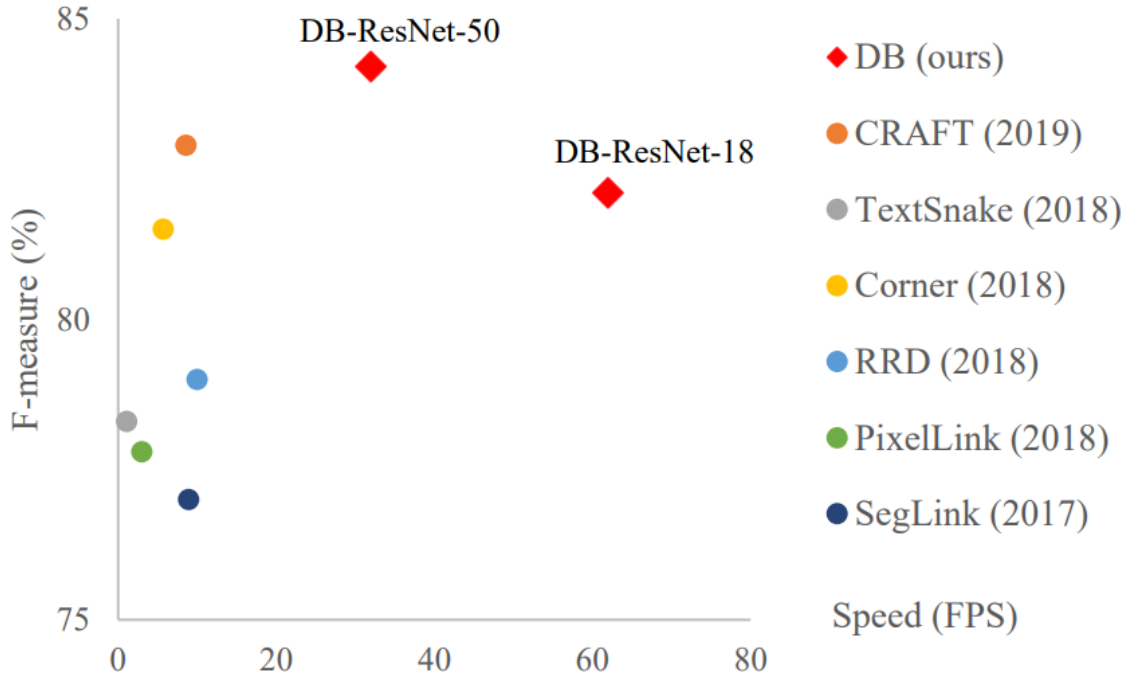
* VP — Verdadeiros Positivos
* FP — Falsos Positivos
* FN — Falsos Negativos

As fórmulas são definidas como:

$$\text{Precisão} = \frac{VP}{VP+FP} \quad \text{e} \quad \text{Revocação} = \frac{VP}{VP+FN}$$

$$F-Measure = \frac{2 \cdot \text{Precisão} \cdot \text{Revocação}}{\text{Precisão} + \text{Revocação}}$$

Figura 8 – Comparação entre modelos de detecção de texto no dataset MSRA-TD500. F-Measure é a média harmônica entre precisão e revocação. Quanto maior esse valor, melhor o equilíbrio entre os acertos positivos e a cobertura dos elementos relevantes.



Fonte: (Liao et al., 2020)

– **Visual Language Model (VLM)**

Vision Language Models (VLMs) são uma classe de sistemas de IA projetados para processar e compreender de forma conjunta informações visuais e textuais. Diferente de modelos tradicionais, que tratam imagens e textos separadamente, os VLMs são treinados em conjuntos de dados multimodais que associam imagens a descrições ou legendas, possibilitando a integração entre percepção visual e compreensão linguística. Essa característica os torna capazes de realizar tarefas como *image captioning*, resposta a perguntas visuais, recuperação de imagens baseada em texto e geração de diálogos multimodais (Verbovskiy, 2025).

Na essência, as VLMs combinam arquiteturas de visão computacional e processamento de linguagem natural, normalmente baseadas em *transformers*. Esses sistemas tipicamente usam codificadores de imagem (como Vision Transformers ou CNNs) para extrair *features* de imagens, e modelos de linguagem baseados em *transformers* para lidar com textos. Assim, um mecanismo de fusão multimodal realiza o alinhamento entre as duas modalidades, viabilizando o raciocínio conjunto.

Os VLMs também tem seu impacto na área de OCR, já que estudos recentes mostram que o rápido progresso desses modelos tem transformado tarefas de OCR. Tais estudos apontam que essas arquiteturas transformaram tarefas de OCR de algoritmos restritos para tornarem-se um componente essencial da compreensão multimodal. Nesse contexto, a acurácia do OCR passa a ser também uma métrica da capacidade de raciocínio visual-linguístico desses modelos (Zhang et al., 2024). Além disso, pesquisas recentes têm mostrado que VLMs generalistas podem ser refinados (*fine-tuning*) para tarefas específicas de OCR em contextos com poucos dados disponíveis. Por exemplo, Varela (2024) demonstraram que o modelo Qwen2-VL-2B-Instruct, após adaptação para textos históricos em malaio (escrita Jawi), atingiu taxas competitivas de erro de caracteres (CER) e palavras (WER), mesmo em cenários de baixa disponibilidade de dados anotados.

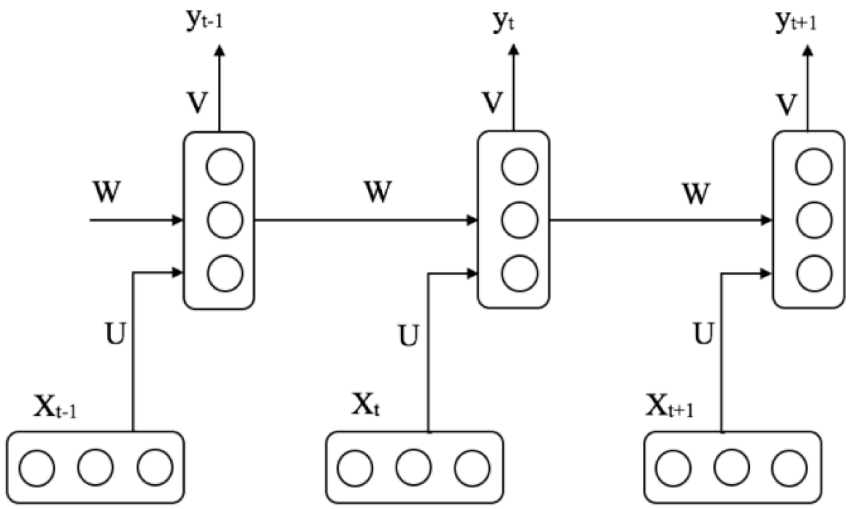
Embora o escopo desse trabalho utilize uma abordagem modular, baseada em técnicas bem definidas de PDI, ML e OCR, a literatura aponta que os VLMs representam um caminho natural para a evolução desse tipo de aplicação. Enquanto a solução proposta para esse trabalho separa as etapas de detecção de texto, processamento de imagem e reconhecimento de texto, os VLMs serviriam como uma abordagem *all-in-one* para esse tipo de problema. Assim, esses modelos serviriam como uma evolução futura para resolver problemas semelhantes ao do presente trabalho.

- **Recurrent Neural Networks (RNNs)**

As Redes Neurais Recorrentes (RNNs) são um tipo de arquitetura neural projetada para lidar com dados sequenciais, como séries temporais, linguagem natural e texto. Diferentemente das redes neurais tradicionais, as RNNs possuem conexões recorrentes que per-

mitem à rede manter uma "memória" das entradas anteriores por meio de estados ocultos que são atualizados ao longo do tempo. Isso permite que a rede aprenda dependências temporais e de contexto, o que é essencial para tarefas como reconhecimento de texto ou tradução automática (Mishra; Kaur, 2018).

Figura 9 – Arquitetura RNN



Fonte: (Mishra; Kaur, 2018)

- **Convolutional Recurrent Neural Networks (CRNNs)**

  As Redes Neurais Convolucionais Recorrentes (CRNNs) são uma arquitetura híbrida que combinam CNNs e RNNs, juntamente com a função de perda Connectionist Temporal Classification (CTC) para tarefas de reconhecimento de texto. Elas aproveitam a capacidade de extração de *features* de uma CNN com o poder de modelagem sequencial das RNNs. Essa arquitetura híbrida consiste de três componentes principais: as camadas convolucionais, camadas recorrentes e uma camada de transcrição (Goud et al., 2022).

  Ainda segundo Goud et al. (2022), sabe-se que as camadas convolucionais atuam inicialmente extraindo automaticamente sequências de *features* da imagem de entrada, como padrões de bordas e texturas, sendo essa etapa baseada nas camadas convolucionais e max-pooling de uma CNN padrão, tirando as camadas totalmente conectadas. Em seguida, essas sequências são repassadas para uma rede recorrente (geralmente uma RNN bidirecional), que realiza previsões para cada *frame* da sequência. A capacidade das RNNs em capturar dependências temporais permite à CRNN interpretar corretamente a ordem e o contexto dos caracteres. Por fim, a camada de transcrição traduz essas previsões em uma sequência textual final.

  Outro ponto importante é que a CRNN é geralmente treinada com a função de perda CTC, que permite o aprendizado de alinhamentos entre imagens e textos sem a necessidade de uma correspondência explícita entre entrada e saída. Isso possibilita o treinamento *end-*

to-end, mostrando-se útil em contextos onde a segmentação de caracteres é difícil ou desnecessária, tornando a CRNN uma solução poderosa e eficiente para OCR moderno.

2.3 RECONHECIMENTO ÓPTICO DE CARACTERES (OCR)

O Optical Character Recognition (OCR) é uma tecnologia da área de visão computacional que permite a conversão de texto presente em imagens, documentos digitalizados, manuscritos e vídeos em texto codificável e pesquisável por máquina. Nesse sentido, essa tecnologia permitiu que documentos escaneados se tornassem mais que simples arquivos de imagens, podendo ser transformados em documentos completamente acessíveis com textos reconhecidos por computadores (Singh; Bacchuwar; Bhasin, 2012). Essa tecnologia tem aplicações em diversas áreas, principalmente para análise de documentos, como registros financeiros, médicos e dados pessoais. Além disso, também é utilizado para extração de texto em cenas naturais e reconhecimento de placas de veículos, sendo essa última aplicação comumente conhecida como sistemas Automatic Number Plate Recognition (ANPR) (Raj; Gupta; Malhotra, 2022).

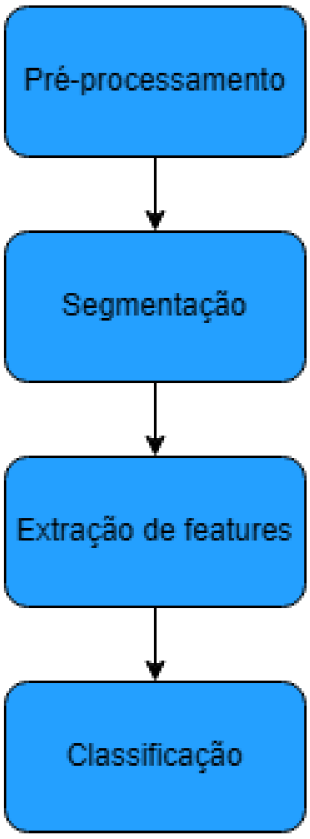
2.3.1 SISTEMA DE OCR TRADICIONAL

Tradicionalmente, o OCR funciona com base em pipelines compostos por etapas bem definidas, sendo elas: aquisição da imagem, pré-processamento da imagem, segmentação de caracteres, extração de *features* e, finalmente, classificação de caracteres, conforme destaca Srivastava, Verma e Sharma (2022). Nesse sentido, é importante entender a finalidade de cada uma dessas etapas.

Primeiramente, após a aquisição da imagem, é aplicada a etapa de pré-processamento, em que a imagem é refinada por meio de técnicas como binarização, remoção de ruído, redimensionamento, entre outras. Essas técnicas servem de suporte para a etapa seguinte, a segmentação, que divide o conteúdo textual em níveis como parágrafos, linhas, palavras e caracteres, utilizando métodos como perfis de projeção ou detecção de *blobs*. Em seguida, realiza-se a extração de *features*, aspecto-chave para o reconhecimento, com uso de abordagens como análise de contornos, texturas ou transformadas como Fourier e Wavelet (Srivastava; Verma; Sharma, 2022).

Na etapa de classificação, algoritmos como KNN, SVM ou redes neurais são empregados para associar os vetores de características aos respectivos caracteres. Por fim, ocorre o reconhecimento, onde os caracteres são decodificados em texto estruturado, buscando máxima acurácia. Essa arquitetura clássica de sistemas OCR é amplamente descrita por Srivastava, Verma e Sharma (2022) e também apresentada de forma consolidada por Eikvil (1993). *Engines* tradicionais como o Tesseract seguem majoritariamente esse pipeline convencional em suas versões clássicas (Smith, 2007). Por fim, é possível ver um fluxograma das etapas explicadas na figura 10.

Figura 10 – Abordagem metodológica para um sistema OCR



Fonte: Adaptado de (Srivastava; Verma; Sharma, 2022)

2.3.2 APLICAÇÕES DE DEEP LEARNING PARA OCR

Mais recentemente, com o avanço da área de Inteligência Artificial (IA), métodos baseados em Deep Learning revolucionaram a forma como são feitas tarefas de reconhecimento de caracteres e diversas outras tarefas no campo de visão computacional. O uso desses modelos eliminam a necessidade do uso de métodos explícitos para extração de *features*, como é feito tradicionalmente (Manocha; Tewari, 2021). Eventos como a competição ImageNet em 2011 impulsionaram o desenvolvimento de arquiteturas profundas, trazendo grande atenção ao uso de redes neurais para tarefas de reconhecimento visual, incluindo OCR (Liu et al., 2023). Assim, abordagens baseadas em ML permitem que os computadores tomem decisões com base em dados, sem a necessidade de programação manual explícita (Sharma; Kaushik; Gondhi, 2020).

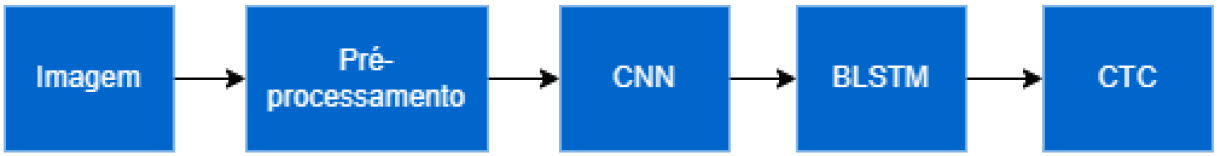
Nesse sentido, Sharma, Kaushik e Gondhi (2020) destacam que as CNNs vêm sendo amplamente utilizadas em tarefas de OCR devido à sua capacidade de extrair automaticamente características relevantes de imagens, sem necessidade da extração manual de características. As CNNs aceitam como entrada imagens em diferentes formatos, seja em escala de cinza, binárias ou RGB, e aprendem, por meio de filtros convolucionais, a identificar bordas, formas e padrões úteis para o reconhecimento de caracteres (Sharma; Kaushik; Gondhi, 2020). Esses padrões são gradualmente refinados ao longo de camadas convolucionais e de pooling, que

produzem mapas de ativação utilizados pelas camadas totalmente conectadas para realizar a classificação final dos caracteres (Manocha; Tewari, 2021).

Dessa forma, essa abordagem se mostrou eficaz diante a desafios visuais como ruídos, distorções geométricas e variações de fonte e escrita. Um dos primeiros modelos de sucesso foi o LeNet, aplicado ao reconhecimento de dígitos manuscritos. A partir dele, outras arquiteturas como AlexNet, VGGNet, ResNet e GoogleNet também passaram a ser exploradas para OCR (Sharma; Kaushik; Gondhi, 2020).

Um exemplo recente e relevante do estado da arte em OCR é a arquitetura proposta por Liu et al. (2023), que combina as redes DBNet, CRNN e a função de perda CTC. O pipeline proposto funciona da seguinte forma: O DBNet é utilizado para detecção de texto, sendo capaz de detectar regiões textuais em fundos complexos, mesmo sob ruído, baixa resolução ou variações de fundo. Em seguida, a rede CRNN é aplicada na tarefa de reconhecimento, combinando extração de *features* com a modelagem da sequência textual. Por fim, é aplicada a técnica CTC como função de perda, permitindo o treinamento da rede de forma end-to-end, sem a necessidade de alinhamento entre entrada e saída. É possível ver um diagrama com a estrutura da CRNN na figura 11.

Figura 11 – Estrutura da rede CRNN. O diagrama representa o fluxo de entrada de uma imagem, que passa por um pré-processamento antes de ser alimentado para uma CNN (Convolutional Neural Network), que passa, em seguida, por uma BLSTM (Bidirectional Long Short-Term Memory) com uma função CTC (Connectionist Temporal Classification)



Fonte: Adaptado de (Liu et al., 2023)

Essa combinação de tecnologias representa uma abordagem robusta e amplamente adotada em sistemas de OCR modernos, representando o estado da arte de aplicações dessa tecnologia.

3 TRABALHOS RELACIONADOS

Esse capítulo apresenta alguns trabalhos que se assemelham ao propósito desse projeto, visto que trabalhos de extração de texto em vídeos ou filmes, mais especificamente de legendas, são raros. Assim, os trabalhos relacionados foram adquiridos por meio da pesquisa de palavras chaves como "subtitle extraction ocr" e "text extraction ocr" no Google Scholar. Primeiramente, discorre-se sobre um projeto que utiliza técnicas de Deep Learning para o reconhecimento de legendas em vídeos, utilizando uma abordagem híbrida de técnicas. Em seguida, é visto um projeto em que são usadas técnicas de Deep Learning, PDI e OCR para extrair legendas de vídeos, tendo um resultado muito semelhante ao esperado para a proposta deste TCC. Por fim, é visto um trabalho que utiliza as mesmas técnicas para reconhecer placas de veículos e também um TCC que utiliza OCR e PDI para identificar propriedades de produtos a partir da leitura de rótulos.

3.1 MULTI-LANGUAGE VIDEO SUBTITLE RECOGNITION WITH CONVOLUTIONAL NEURAL NETWORK AND LONG SHORT-TERM MEMORY NETWORKS

Neste trabalho, Singkhornart e Surinta (2022) propuseram técnicas de Deep Learning para reconhecer legendas em vídeos. A motivação do trabalho veio da ampla variedade de legendas encontradas em vídeos presentes em grandes plataformas como Youtube e Facebook, que são adicionadas buscando auxiliar pessoas com deficiência auditiva e também aqueles que não entendem certos idiomas. Nesse sentido, o trabalho buscou experimentar o reconhecimento das legendas com consequente transformação delas em formato de texto, a partir de imagens. Para isso, utilizaram uma arquitetura híbrida que combina diferentes técnicas de *Deep Learning*.

O modelo proposto foi uma combinação das arquiteturas de Redes Neurais Convolucionais (CNNs) e Long Short-Term Memory (LSTM), mais conhecidas como CNN-LSTM para reconhecimento de textos. Para o CNN, essa foi baseada na arquitetura VGGNet, sendo integrada então com uma rede LSTM bidirecional (Bi-LSTM), para então serem transformadas por meio da função *softmax*, em uma distribuição de probabilidade. Assim, essa arquitetura aprende a partir das imagens das legendas e prediz a sequência de texto utilizando a Connectionist Temporal Classification (CTC) como função de perda, utilizada para mapear a sequência de caracteres sem a necessidade de segmentação explícita, sendo essencial para a extração de legendas contínuas.

Esse trabalho utilizou um dataset de legendas extraídas de 24 vídeos compartilhados no YouTube e Facebook. O conjunto de dados contém 4.224 imagens de legendas nos idiomas inglês, tailandês e numerais árabes, totalizando 157 caracteres diferentes. As imagens das legendas foram convertidas para resolução 1280×720 pixels e armazenadas no formato JPG. Para a validação, foi aplicada a técnica de *5-fold cross-validation*.

Para a aplicação do trabalho, foram treinadas quatro variações do modelo CNN-LSTM (Modelos A, B, C e D) baseadas em arquiteturas VGG16 e VGG19, combinadas com Bi-LSTM

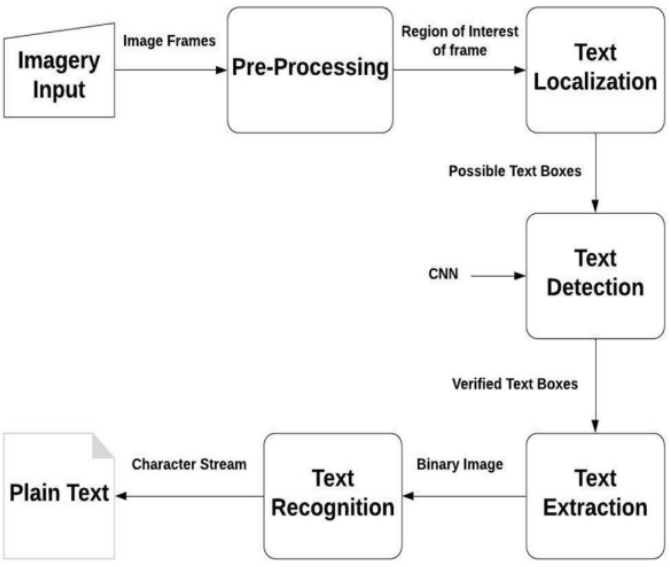
ou Bi-GRU. A avaliação foi feita utilizando a métrica Character Error Rate (CER), que mede a taxa de erro no reconhecimento de caracteres. Com isso, foi visto que o Modelo C, que foi combinado com a rede Bi-LSTM, providenciou uma taxa de erro de 9,36%, sendo o melhor valor dentre todos.

Por fim, esse estudo concluiu que a arquitetura CNN-LSTM é eficaz para o reconhecimento de legendas em vídeos, apresentando um erro reduzido na predição de caracteres. Porém, o alto número de camadas convolucionais tornou o treinamento mais demorado em comparação com modelos mais simples. Como trabalhos futuros, os autores sugerem o uso de arquiteturas híbridas de CNN e técnicas de *data augmentation* para melhorar ainda mais o desempenho do reconhecimento de legendas.

3.2 AN END-TO-END SYSTEM FOR SUBTITLE TEXT EXTRACTION FROM MOVIE VIDEOS

Neste trabalho, os autores Elshahaby e Rashwan (2022) propuseram um sistema completo e automatizado para extração de legendas em vídeos de filmes, utilizando uma abordagem híbrida que combina técnicas geométricas com redes neurais profundas. A motivação do projeto vem dos benefícios que a extração de legendas traz para uma análise de filmes mais rápida e eficiente, colaborando para pontos como a classificação de filmes e tradução automática, por exemplo. Sabendo-se disso, o projeto definiu como desafio principal a extração de texto de *backgrounds* complexos de filmes utilizando uma metodologia específica, baseada em técnicas de PDI, detecção e extração de texto, como mostra o diagrama da figura 12.

Figura 12 – Metodologia para detecção/reconhecimento de texto



Fonte: (Elshahaby; Rashwan, 2022)

Primeiramente, em relação a parte de localização do texto, os autores optaram por

fazer com que o sistema recortasse automaticamente a parte inferior da tela, onde as legendas geralmente estão localizadas. Em seguida, uma CNN determina se há ou não texto naquele determinado *frame*. Quando confirmado, a legenda é isolada comparando *frames* consecutivos com e sem texto, utilizando técnicas de rastreamento de movimento (KLT tracking) e mínimo erro bidirecional, o que permite estabilizar os caracteres da legenda mesmo com movimentos na cena. Para aprimorar a qualidade da imagem antes da etapa de OCR, o sistema realiza técnicas de pré-processamento de imagem, por meio da fusão de múltiplos *frames*, aumentando o contraste e a resolução da imagem (de 70 para 300 dpi).

A rede neural escolhida para a verificação de texto foi a HMVGG16, uma variação da conhecida arquitetura VGG16, composta por 16 camadas e uma função softmax para classificação binária. Essa arquitetura foi treinada com Stochastic Gradient Descent com momentum (SGDM) e teve uma ótima acurácia de 98% na tarefa de classificar *frames* como tendo ou não legendas. Além disso, os autores implementaram versões mais leves, como a rede HMLe8, que possui menos camadas e é mais rápida, porém com uma precisão inferior de 92,31%. O sistema também se beneficia de técnicas clássicas de processamento de imagem, como recorte de regiões de interesse, realce de contraste, detecção de contornos e interpolação para aumento de resolução, utilizando elas tanto para ajudar na localização dos textos como na preparação das imagens para obter melhores resultados para o OCR.

Quanto ao dataset, foi utilizado o chamado "FiVid", contendo vídeos com legendas de diferentes fontes, tamanhos e estilos. Os vídeos são fontes de filmes e séries e contém legendas de idiomas como árabe, inglês e chinês. O dataset foi utilizado para avaliar o sistema por completo, para treino e também testes. Com isso, os resultados experimentais usando esses vídeos foram o seguinte: na extração de legendas, o sistema alcançou acurácia de 96,15%, precisão de 93,97%, recall de 94,79% e F1-score de 94,38%. Na etapa de OCR, utilizando o Abbyy SDK sobre as legendas pós-processamento, o sistema obteve acurácia média de 97,75% no reconhecimento de caracteres.

Por fim, o trabalho ofereceu uma contribuição sólida para o campo de visão computacional, buscando especificamente a extração automatizada de legendas em vídeos, combinando redes neurais profundas, processamento de imagem e engenharia de dados. Além disso, os autores apontaram como trabalhos futuros para o projeto criar seu próprio OCR e diminuir o tempo de execução para o classificador de *frames*, para que assim pudessem rodar o sistema em tempo real.

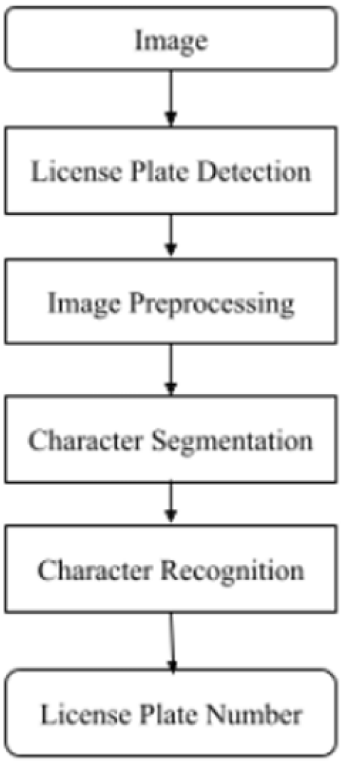
3.3 LICENSE PLATE RECOGNITION SYSTEM USING YOLOV5 AND CNN

Nesse trabalho, Raj, Gupta e Malhotra (2022) propuseram um sistema de reconhecimento automático de placas veiculares utilizando um modelo de detecção de objetos, técnicas de PDI e OCR. O projeto veio da problemática crescente com relação ao aumento da circulação de veículos em cidades grandes, aumentando, por consequência, as violações de trânsito. Nesse sentido, um sistema para monitorar os veículos nas ruas, mais especificamente as placas,

colaboraria para a penalização de tais violações e consequentemente para um melhor controle e gerenciamento do tráfego nas ruas .

A pipeline utilizada pelo sistema consiste em receber uma imagem como input, identificar a placa do carro com uma bounding box utilizando a tecnologia YOLO, aplicar técnicas de PDI para realçar os caracteres e por fim, extrai-los usando OCR para ter o número da placa em formato de texto. Primeiramente, foi utilizado um dataset de mais de 250 imagens para treinar o modelo YOLOv5, assim como outro dataset contendo caracteres de A-Z e 0-9 para treinar um modelo de CNN baseada na arquitetura LeNet para o OCR.

Figura 13 – Pipeline do sistema ANPR



Fonte: (Raj; Gupta; Malhotra, 2022)

Para o treinamento do modelo YOLOv5, foram usadas 100 *epochs* com um lote de tamanho 32. Para inicialização de peso, foram usados os pesos do YOLOv5. Para avaliação do modelo, foi obtido uma precisão mAP@0.5 de 96,5%. Quanto para o OCR, foram utilizadas técnicas de *data augmentation* para ampliar o dataset de imagens de treinamento, chegando a até 96015 imagens. Assim, o modelo CNN foi treinado com lotes de tamanho 32 e em 50 *epochs*. Por fim, a rede atingiu uma acurácia de 98,3%.

Para o pré-processamento de imagem, a região da placa, após ser recortada, é submetida a técnicas que envolvem filtros bilaterais, aumento de contraste, conversão para escala de cinza e limiarização adaptativa, visando destacar os caracteres da placa do fundo das imagens. Em seguida, os caracteres são segmentados por contornos, usando critérios morfológicos e de

proporção, por fim tendo cada caracter da placa uma bounding box. Dessa forma, cada caracter segmentado é alimentado ao modelo de OCR treinado, dando um resultado correspondente.

Unindo todas as técnicas citadas anteriormente, tem-se a solução final, que recebe uma imagem de um veículo com a placa à mostra e retorna em texto os caracteres da placa. Assim, para avaliar a performance do sistema de reconhecimento de placas, os autores utilizaram duas métricas principais: *similarity score* e *edit distance*. A primeira métrica mede o grau de semelhança entre o texto reconhecido e o texto real da placa. Foi calculada usando o método SequenceMatcher da biblioteca difflib do Python, que retorna um valor entre 0 e 1, sendo que, quanto mais próximo de 1, maior a semelhança. O valor final obtido foi 0.825, indicando um bom nível de correspondência textual, ainda que não perfeita. A segunda métrica representa o número de operações (inserções, remoções e substituições) necessárias para transformar o texto reconhecido no texto real. Para isso foi usada a biblioteca jellyfish do Python. O valor médio obtido foi 1.511, o que significa que, em média, é necessário pouco mais de uma operação para corrigir a saída do OCR. Essas métricas foram aplicadas em todos os exemplos do conjunto de teste, mostrando que, embora não tenham sido usados conjuntos de dados extensos, os resultados mostram que o sistema é funcional e promissor.

3.4 EVE: TOWARDS END-TO-END VIDEO SUBTITLE EXTRACTION WITH VISION-LANGUAGE MODELS

O trabalho de (Huang et al., 2025) propõe o EVE (End-to-end Video subtitle Extraction), um método end-to-end para extração de legendas de vídeos utilizando Large Vision-Language Models (LVLMs). A principal motivação do trabalho é superar as limitações dos métodos existentes para extração de legendas em vídeos. Segundo os autores, normalmente os métodos baseados em OCR utilizam múltiplos estágios sequenciais, incluindo detecção de texto, classificação de legendas e reconhecimento de texto, o que pode levar ao acúmulo de erros entre as etapas, influenciando no resultado final. Além disso, esses métodos processam cada *frame* independentemente, ignorando a informação temporal presente nos vídeos.

O EVE é projetado para extrair legendas em formato bilíngue (chinês e inglês), prevendo simultaneamente os *timestamps* e os textos das legendas. A arquitetura do modelo consiste em três módulos principais: um codificador de visão (*vision encoder*), um adaptador e um Large Language Model (LLM). O codificador de visão utiliza a arquitetura Swin-Transformer pré-treinada no Donut, escolhida por sua melhor inicialização para tarefas de OCR e maior eficiência temporal devido ao mecanismo de atenção baseado em janelas deslizantes. Este módulo é responsável por extrair características visuais (*features*) de cada frame do vídeo de entrada.

A principal contribuição técnica do trabalho é o adaptador InterleavedVT, um módulo inovador para compressão de tokens visuais que combina duas abordagens: *average pooling* e Q-Former. O InterleavedVT é composto por dois componentes complementares. O Visual Compressor aplica operações de *average pooling* para preservar informações visuais globais e posicionais das legendas. O Textual Region Compressor utiliza a arquitetura Q-Former com um

mecanismo de janela deslizante para extrair tokens relacionados especificamente ao texto das legendas. Os tokens gerados por ambos os compressores são intercalados (*interleaved*) para formar a sequência final de entrada. Uma característica importante é o uso de projetores separados para cada tipo de token, o que melhora significativamente o alinhamento das características com o modelo de linguagem.

O modelo de linguagem utilizado no EVE é o Qwen2, disponível nas versões de 7B e 0.5B, escolhido por seu suporte robusto à geração bilíngue em chinês e inglês. Este módulo recebe os tokens comprimidos do adaptador e gera as legendas no formato estruturado `[início][fim][texto]`, onde os índices de início e fim correspondem aos frames do vídeo.

O processo de treinamento do EVE é dividido em duas etapas distintas. Na etapa de pré-treinamento, o LLM permanece congelado enquanto o codificador de visão e o adaptador são otimizados com o objetivo de aprender a perceber todos os *tracklets* de texto presentes nos *frames* do vídeo. Na etapa de *fine-tuning* supervisionado, o codificador de visão é congelado enquanto o adaptador e o LLM são otimizados para aprender a discriminar quais textos detectados são mesmo legendas.

Para *benchmarking* da tarefa, os autores criaram o *dataset* ViSa, composto por vídeos coletados dos *datasets* CNVid e MovieChat-1K. O *dataset* é dividido em três conjuntos: pré-treinamento, *fine-tuning* e teste, este último contendo vídeos manualmente anotados divididos entre vídeos curtos do tipo *shorts* e vídeos de filmes. Os vídeos incluem amostras com legendas e sem legendas, permitindo avaliar a capacidade do modelo em distinguir entre diferentes tipos de texto.

Os resultados experimentais demonstram que o EVE-7B supera todos os métodos que foram usados para comparação, envolvendo tanto ferramentas *open-source* quanto serviços em nuvem. Duas métricas foram utilizadas para avaliação: NED (*Normalized Edit Distance*), que avalia a precisão dos textos das legendas extraídas, e SubER (*Subtitle Error Rate*), que avalia tanto a precisão dos textos quanto a acurácia dos *timestamps* preditos. Em vídeos do tipo *shorts*, o EVE-7B alcança 0,882 de NED e 15,16 de SubER, representando uma melhoria de 11,1% em NED sobre o melhor método concorrente (OpenVision). Em vídeos de filmes, o EVE-7B atinge 0,808 de NED e 1,64 de SubER, superando consistentemente os métodos tradicionais. Mesmo a versão mais leve EVE-0.5B, com apenas 0,5 bilhão de parâmetros, consegue superar a maioria dos métodos existentes, demonstrando a eficácia da arquitetura proposta. LVLMs de código aberto, como Qwen2-VL e MiniCPM-V, apresentam desempenho insatisfatório (NED de 0,123 e 0,043 respectivamente em vídeos curtos) pois não foram especificamente treinados para esta tarefa.

Por fim, estudos de ablação, que tem o intuito de avaliar a importância de componentes separadamente, confirmam a importância de cada componente do InterleavedVT. A remoção do Visual Compressor ou do Textual Region Compressor resulta em piora significativa do desempenho. Particularmente, o uso de projetores separados para os *tokens* visuais e textuais é essencial, proporcionando melhorias de até 5,03% em SubER quando comparado ao uso de um único projetor. Os autores também destacam limitações do método, incluindo dificuldades com

vídeos muito longos (duração superior a uma hora) devido ao limite de 120 *frames* de entrada, e a ausência de predição de coordenadas espaciais (*bounding boxes*) das legendas, focando apenas em *timestamps* e textos.

3.5 IDENTIFICAÇÃO DE PROPRIEDADES DE PRODUTOS A PARTIR DA LEITURA AUTOMÁTICA DE RÓTULOS

Nesta monografia, Vicenzi (2021) trouxe à tona o desafio de identificar automaticamente informações úteis presentes em rótulos de produtos, como valores nutricionais e restrições alimentares, visando beneficiar consumidores, produtores e órgãos reguladores, a fim de trazer ganhos em logística, segurança, acessibilidade e comodidade. Nesse sentido, o trabalho propôs uma aplicação capaz de realizar uma leitura automática de rótulos, utilizando técnicas de Visão Computacional, OCR e Aprendizado de Máquina, além de mecanismos de validação textual com regras pré-definidas.

Para a implementação, o autor mostrou como técnicas de PDI desempenharam um papel fundamental para garantir uma entrada mais limpa e estruturada para o OCR. O sistema utiliza uma pipeline composta por técnicas clássicas de processamento de imagem, como equalização de histograma, filtros de suavização, operações morfológicas, limiarização e segmentação com MSER, além do uso do OCR com Tesseract para extrair os textos dos rótulos. A validação dos dados extraídos é realizada com a distância de Levenshtein e regras semânticas.

Mais especificamente sobre a parte de PDI do pipeline, o pré-processamento de imagens iniciou-se com a equalização de histograma para realçar o contraste entre o texto e o fundo do rótulo. Em seguida, foi feita a remoção de ruídos com filtros, como o Gaussiano, bilateral e da mediana, eliminando ruídos visuais e preservando as bordas dos caracteres. Após o pré-processamento, o autor aplicou operações morfológicas, como dilatação, erosão, abertura e fechamento, que auxiliaram tanto na eliminação de pequenas imperfeições quanto na união de partes desconectadas dos caracteres. A limiarização adaptativa, com destaque para o método de Otsu, foi empregada para binarizar a imagem, buscando facilitar o reconhecimento de texto pelo Tesseract. Por fim, foi testado o uso da técnica de segmentação baseada em MSER (Maximally Stable Extremal Regions), para detectar automaticamente regiões de interesse textuais, reduzindo a dependência de cortes manuais.

Com relação aos resultados, o trabalho trouxe cinco abordagens diferentes para comparar os resultados. Essas abordagens variaram na combinação de técnicas de pré-processamento e segmentação. A primeira abordagem buscou aplicar o OCR sem qualquer passo anterior de processamento de imagem, tendo um resultado insatisfatório, principalmente em imagens com baixo contraste, iluminação ruim ou fundos com textura. Em seguida, os resultados foram melhorando gradualmente com a introdução de técnicas de PDI, até chegar no último e melhor experimento, onde o pipeline completo foi utilizado. Nesse experimento, foram usadas técnicas de equalização, suavização e limiarização, seguido da segmentação com MSER, atingindo assim resultados significativamente superiores.

Por fim, a recuperação de informações específicas, como termos nutricionais e restrições alimentares, foi avaliada com base em métricas como acurácia de extração e distância de Levenshtein. Como visto anteriormente, o melhor desempenho foi observado no quinto experimento, onde a integração de todas as técnicas produziu imagens mais nítidas e focadas, facilitando a correta leitura e validação das informações. Esses resultados reforçam a importância de um bom pré-processamento em sistemas de OCR aplicados a ambientes visuais desafiadores, como é o caso de rótulos e, por extensão, legendas embutidas em vídeos, contexto em que este trabalho se insere.

3.6 COMPARATIVO

A seguir é feita uma análise e comparação dos trabalhos quanto à aplicação das técnicas que são relevantes para esse trabalho, com o auxílio de uma tabela comparativa. Além disso, também é analisada qual a contribuição desses trabalhos para o projeto desse TCC. A Tabela 1 resume as principais características técnicas dos trabalhos analisados na seção anterior, considerando o uso de OCR, técnicas de PDI e Deep Learning. Com base nessa síntese, é possível identificar tanto as contribuições de cada trabalho para a fundamentação desse TCC quanto os pontos em que ainda há espaço para avanços.

Primeiramente, quanto ao uso de OCR, todos os trabalhos relacionados aplicam esse conceito em contextos diferentes, o que reforça a relevância dessa tecnologia para tarefas que envolvem extração de texto em imagens. Porém, é possível observar que alguns trabalhos, como o de Vicenzi (2021), fazem uso de OCR em sua forma tradicional, com Tesseract, enquanto outros, como Singkhornart e Surinta (2022) e Elshahaby e Rashwan (2022), buscam o estado da arte, utilizando arquiteturas neurais modernas, como CNN-LSTM. Esse contraste destaca o avanço que vem se tendo com abordagens baseadas em deep learning, que tendem a ser mais robustas diante de maiores desafios quanto à obstáculos visuais.

No que tange a área de PDI, os trabalhos de Vicenzi (2021) e Raj, Gupta e Malhotra (2022) evidenciam bem como o uso de técnicas clássicas para pré-processamento de imagem aumentam a eficácia do OCR. Técnicas como equalização de histograma, binarização adaptativa, filtros de suavização e segmentação baseada em contornos ou MSER são aplicadas para melhorar a qualidade dos textos antes da extração. Tais procedimentos servem de inspiração direta para a etapa de PDI a ser implementada no presente TCC, considerando que as legendas embutidas em vídeos podem ter variações de cor, contraste, entre outras coisas.

Em relação ao uso de Deep Learning, o uso de ferramentas como YOLOv5 para localização de placas de veículos no trabalho de Raj, Gupta e Malhotra (2022) mostrou o excelente desempenho de CNNs para tarefas que envolvem detecção de objetos em imagens, o que é altamente aplicável no caso de legendas. Por outro lado, alguns trabalhos carecem de datasets mais representativos ou não apresentam soluções para problemas como variações de fundo, distorções ou baixa resolução, desafios bastante comuns em vídeos reais.

Dessa forma, o projeto deste TCC se apoia nas metodologias mais eficazes observadas

nos trabalhos, buscando integrar o uso de CNNs, como a YOLO, para detecção moderna de texto, aliadas a técnicas clássicas de PDI e OCR. A proposta visa consolidar esses aprendizados em uma arquitetura simples, robusta e modular, aproveitando o melhor de cada tecnologia para oferecer uma solução eficiente e adaptável à variedade de contextos encontrados em vídeos.

Tabela 1 – Comparação entre os trabalhos relacionados quanto ao uso de técnicas de OCR, PDI e Deep Learning.
Legenda: A = OCR; B = PDI; C = Deep Learning; D = Treinamento de modelo; E = Quantidade de imagens/vídeos do dataset

Trabalho	**A**	**B**	**C**	**D**	**E**
Singkhornart & Surinta (2022)	×	—	×	×	4224 (imagens)
Elshahaby & Rashwan (2022)	×	×	×	×	20+ (vídeos)
Raj, Gupta e Malhotra (2022)	×	×	×	×	800+ (imagens)
Vicenzi (2021)	×	×	—	—	—
EVE (2025)	—	—	×	×	2.35M+ (vídeos)

Fonte: O autor.

Tabela 2 – Métricas de avaliação dos modelos treinados nos trabalhos relacionados.
Legenda: A = Acurácia; B = Precisão; C = Recall; D = F1-Score; E = CER; F = NED

Trabalho	**A (%)**	**B (%)**	**C (%)**	**D (%)**	**E (%)**	**F (%)**
Singkhornart & Surinta (2022)	—	—	—	—	9,36	—
Elshahaby & Rashwan (2022)	98,04	97,73	98,27	97,97	—	—
Raj et al. (2022) - Placas	—	96,60	96,50	—	—	—
Raj et al. (2022) - Caracteres	98,30	—	—	—	—	—
EVE (2025)	—	—	—	—	—	88,20

Fonte: O autor.

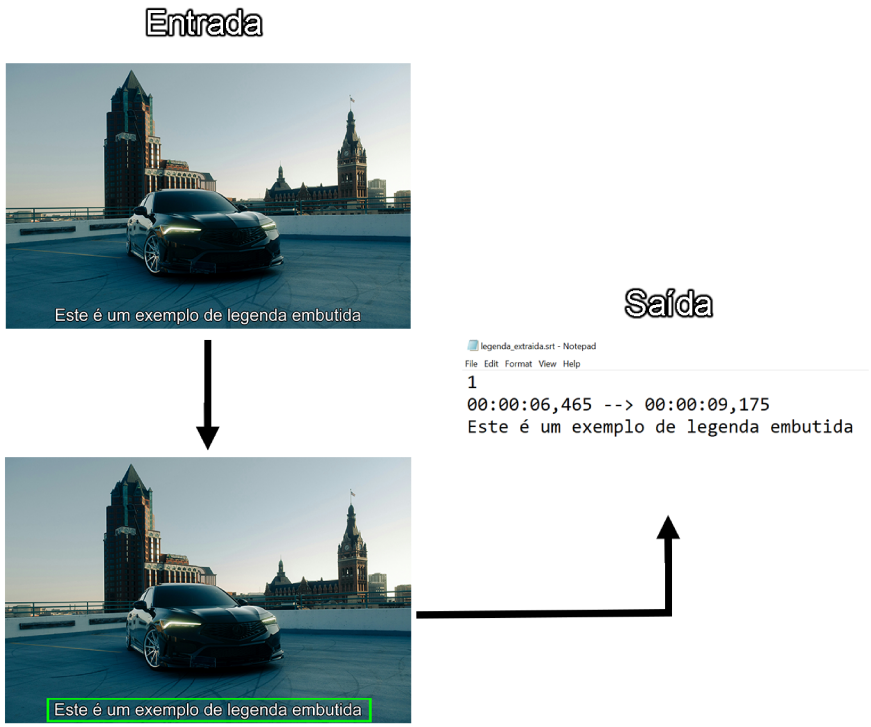
4 METODOLOGIA DE DESENVOLVIMENTO

Esta seção descreve os métodos e ferramentas que foram utilizadas para o desenvolvimento da solução proposta. São apresentadas as tecnologias envolvidas, a arquitetura geral do sistema, a base de dados adotada e as etapas de implementação utilizadas.

4.1 RESUMO DA PROPOSTA

De forma geral, o objetivo deste trabalho é desenvolver um software capaz de extrair legendas embutidas em vídeos (*burned-in subtitles*) e gerar um arquivo .srt com as legendas devidamente sincronizadas com seus *timestamps*. Para isso, foram empregadas tecnologias que envolvem redes neurais pré-treinadas para detecção e reconhecimento de texto (OCR), além de técnicas para pós-processamento textual. A figura 14 exemplifica, de forma simplificada, como é o processamento de um frame de um vídeo com legenda embutida.

Figura 14 – Exemplo de funcionamento do software. As imagens mostram, em sequência: 1. Entrada de um frame de um vídeo no software; 2. Processo de detecção e reconhecimento do texto no frame; 3. Extração e sincronização da legenda extraída, levando em conta o processamento de todos os frames necessários para tal.



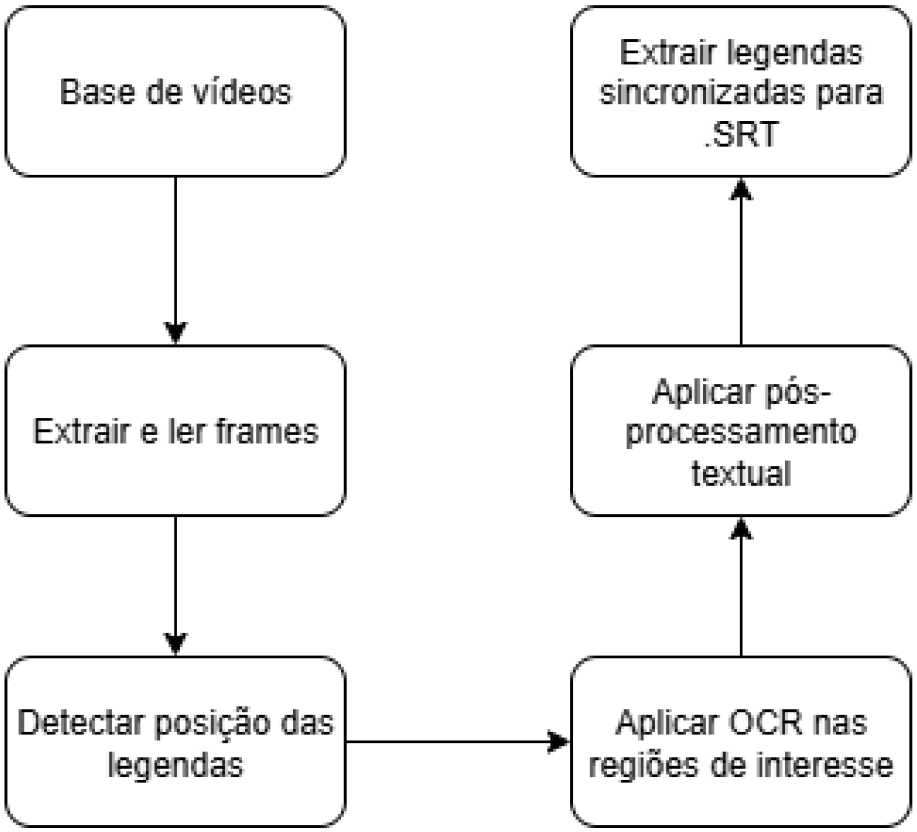
Fonte: O autor.

O sistema foi implementado utilizando a linguagem de programação Python e suas bibliotecas voltadas à visão computacional e processamento textual. O pipeline proposto abrange a detecção da presença de texto em *frames* individuais de vídeo por meio da aplicação do modelo DBNet, seguido do pré-processamento da região de interesse (ROI) e alimentação para um

modelo baseado em CRNN, extraindo-se o texto. Todo esse pipeline é suportado por meio do uso de uma biblioteca moderna de OCR. Além disso, a sincronização das legendas é inferida com base na presença ou ausência de texto nos frames consecutivos, mapeando os intervalos de tempo corretamente.

Para fins de implementação e testes, foram utilizados vídeos disponíveis na internet, contendo episódios de filmes, documentários e séries, dos quais foram extraídos trechos de até 1 minuto contendo legendas. Esses segmentos permitiram avaliar o desempenho do sistema com base na taxa de acertos e erros na extração das legendas. Para ilustrar melhor a solução, a figura 15 indica as etapas de funcionamento da arquitetura.

Figura 15 – Fluxograma da arquitetura



Fonte: O autor.

4.2 FERRAMENTAS E BIBLIOTECAS

Para o desenvolvimento da solução proposta, foi utilizada a linguagem de programação Python, amplamente adotada em projetos de ciência de dados, visão computacional e inteligência artificial. Visto isso, o software proposto utilizou uma combinação de Deep Learning e OCR. Para aplicá-las em conjunto, foram utilizadas as seguintes tecnologias: para detecção e reconhecimento de texto, foi utilizada a biblioteca PaddleOCR, que contém um modelo de detecção de texto baseado no DBNet e vários modelos para reconhecimento de texto, incluindo

um baseado em CRNN, que foi o utilizado para esse trabalho; para processamento de vídeo, foi utilizada a biblioteca de visão computacional OpenCV; por fim, para pós-processamento do OCR, foram utilizadas as bibliotecas pyspellchecker, wordninja e o módulo de regex nativo do Python.

4.2.1 DETECÇÃO E RECONHECIMENTO TEXTUAL

O PaddleOCR é uma biblioteca de código aberto desenvolvida pela comunidade PaddlePaddle (Baidu), oferecendo um pipeline completo para tarefas de OCR, incluindo módulos de detecção e reconhecimento de texto, que utilizam de diversos modelos pré-treinados (Contributors, 2025). Entre suas opções, encontram-se modelos de detecção baseados no DBNet e modelos de reconhecimento fundamentados em arquiteturas CRNN, que constituem soluções amplamente utilizadas e eficientes na extração de texto em imagens.

Entretanto, apesar de o PaddleOCR disponibilizar modelos para múltiplos idiomas, o modelo de reconhecimento adotado neste trabalho é otimizado especificamente para textos em inglês, chinês e japonês. Essa limitação linguística influenciou diretamente a seleção do conjunto de vídeos utilizado na etapa de avaliação. Ainda assim, a modularidade e eficiência dessa biblioteca a tornam uma solução prática para aplicações reais, motivo pelo qual foi adotado neste trabalho (Contributors, 2025).

4.2.2 PÓS-PROCESSAMENTO TEXTUAL

Após a extração inicial de textos por OCR, é comum que os resultados tenham inconsistências, como a concatenação indevida de palavras, erros de ortografia e espaçamento incorreto de pontuações. Para mitigar esses problemas e obter uma transcrição mais próxima das legendas reais exibidas no vídeo, é utilizada uma etapa de pós-processamento textual baseada em três ferramentas principais: `pyspellchecker`, `wordninja` e `regex`.

- **PySpellChecker**

  A biblioteca `pyspellchecker` é um módulo em Python voltado à verificação ortográfica, permitindo identificar a existência e a acurácia de palavras em relação a dicionários de referência. Essa ferramenta é particularmente útil em aplicações nas quais a precisão lexical é crítica, como em sistemas de gerenciamento de conteúdo, editores de texto e processamento de documentos. Nesse trabalho, sua utilização possibilitou diferenciar textos legíveis de fragmentos de ruído, auxiliando tanto na correção ortográfica quanto na filtragem de blocos de legendas inválidos (Barris, 2024).

- **WordNinja**

  Em paralelo, a biblioteca `wordninja` foi empregada para mitigar o problema da concatenação de palavras, frequente em saídas de OCR, ao segmentar cadeias alfanumé-

ricas longas em palavras individuais com base em estatísticas de frequência linguística. Dessa forma, casos como “LIFEISGREAT”, por exemplo, puderam ser transformados em “LIFE IS GREAT”, preservando a naturalidade e a legibilidade da legenda (Timminator, 2024).

- **Regex**

  Por fim, expressões regulares (`regex`) foram utilizadas como ferramenta complementar para tratar padrões específicos de pontuação e contrações. Essa abordagem visou corrigir separações incorretas de apóstrofos, vírgulas e outros sinais gráficos, evitando que o texto processado exibisse espaçamentos inadequados, como “IT ’S” em vez de “IT’S”, ou vírgulas isoladas com espaços indevidos.

O uso conjunto dessas três técnicas mostrou-se essencial para refinar os resultados do OCR, fornecendo legendas mais consistentes e fiéis ao conteúdo original.

4.2.3 IMAGEM E VÍDEO

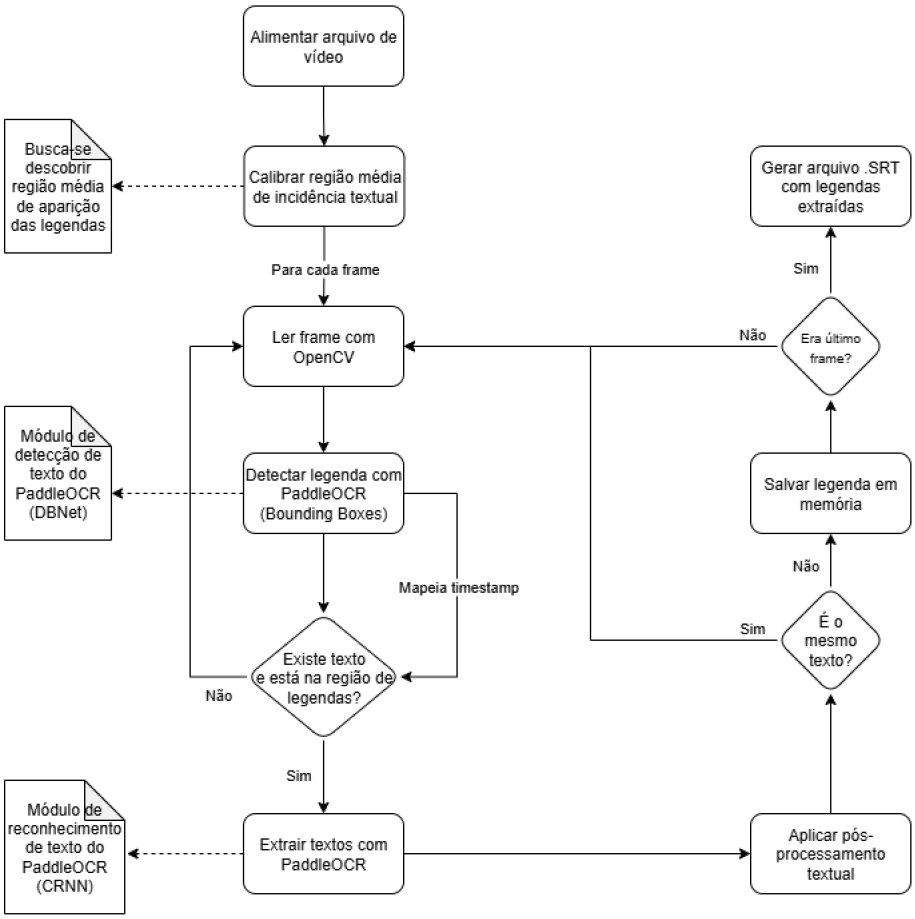
- **OPENCV**

  A Open Source Computer Vision Library (OpenCV) é uma biblioteca de visão computacional fundada em 2000, contendo milhares de algoritmos para tarefas como filtros de imagem, calibração de câmera, reconhecimento de objetos e análise estrutural. Essa biblioteca contém módulos de Processamento de Imagens, Video I/O, Estrutura de Dados, Álbebra Linear e uma Interface Gráfica do Usuário (GUI) básica com sistema de janelas independentes e controle de mouse e teclado. Assim, o que interessa para esse trabalho é o módulo de PDI, utilizada para tarefas como carregamento e manipulação de imagens, conversão para tons de cinza, binarização, redimensionamento e aplicação de filtros morfológicos, por exemplo (Wikipédia, 2024).

4.3 METODOLOGIA

Essa seção descreve detalhadamente cada etapa envolvida no pipeline de extração de legendas embutidas em vídeos. Cada etapa emprega alguma das ferramentas citadas anteriormente, de acordo com a funcionalidade necessária em questão. Assim, unindo as etapas de forma sequencial, espera-se alcançar a sincronização e transcrição das legendas em um arquivo `.srt`. A figura 16 mostra o fluxograma esperado dessas etapas.

Figura 16 – Metodologia passo a passo



Fonte: O autor.

1. **Extração de Frames com OpenCV:** O vídeo de entrada é convertido em uma sequência de *frames* utilizando o OpenCV. Os *frames* são extraídos com uma taxa configurável (1-10 FPS) e servem como insumo para as etapas seguintes.

2. **Detecção de Texto:** Cada *frame* é alimentado ao PaddleOCR, que internamente aplica técnicas de pré-processamento de imagem como normalização e redimensionamento, buscando identificar textos com o modelo baseado em DBNet. Quando detectados, os textos são marcados com uma *bounding box* e preparados para o reconhecimento de texto na etapa seguinte.

   OBS (Posicionamento das Legendas): Utilizando-se dessa lógica de detecção de texto, é feito um processo de calibração no começo do pipeline buscando identificar a posição média das legendas no vídeo, para eliminar detecções de texto aleatórias. Isso é feito por meio da coleta de todas as detecções textuais durante uma parte do vídeo, calculando a posição média dessas detecções em busca de definir a posição média das legendas do vídeo.

3. **Verificação da Posição do Texto:** Após detectado, o texto tem as coordenadas de sua *bounding box* verificadas com a posição média detectada para legendas no vídeo, bus-

cando identificar se a detecção trata-se de uma possível legenda ou de um texto perdido. Se a verificação for positiva, significa que o texto está dentro da região média das legendas, seguindo no pipeline.

4. **Reconhecimento de Texto:** Passando pela verificação de posicionamento, a região textual detectada é alimentada a um modelo baseado em CRNN do PaddleOCR, obtendo-se o conteúdo textual da imagem. Todo esse processo, assim como o de detecção, ocorre intermanente no PaddleOCR.

5. **Pós-Processamento Textual:** Após um texto ser extraído, é feito um pós-processamento textual buscando corrigir eventuais falhas de OCR. Esse processamento é feito por meio de um algoritmo que quebra o texto em *tokens* e faz algumas verificações, sendo elas: se a língua for inglesa, existe o suporte para a verificação e correção de palavras concatenadas (erro comum de OCR), por meio do uso de uma biblioteca que contém o dicionário inglês; para quaisquer línguas, é feita uma correção de espaçamento para caracteres como pontos de interrogação, exclamação e aspas, sendo erros comuns encontrados após OCR.

   OBS (Mapeamento Temporal): Ao mesmo tempo que todo o pipeline ocorre, é aplicado um controle temporal para sincronizar as legendas com seus *timestamps*. Isso é feito da seguinte forma: quando uma nova legenda é detectada pelo modelo, inicia-se uma lógica para detectar o início e fim dessa legenda, baseando-se na verificação de semelhância textual entre as detecções textuais dos frames consecutivos. Juntamente com isso, é feito um cálculo do timestamp com base na quantidade de frames e o FPS do vídeo, para registrar a duração da legenda.

6. **Geração do Arquivo .SRT:** Com todas as legendas extraídas e devidamente sincronizadas utilizando seus *start_time* e *end_time*, um arquivo `.srt` é gerado. Cada bloco de legenda é numerado e contém:

   - Um índice sequencial;
   - O tempo de exibição no formato `hh:mm:ss,ms`;
   - O texto extraído da legenda.

Essa organização torna a solução compatível com ferramentas de edição, tradução e exibição de legendas amplamente utilizadas.

4.4 BASE DE VÍDEOS

Para a realização dos testes e avaliação da solução proposta, foi utilizada uma base própria composta por 20 vídeos, incluindo trechos de documentários, filmes e séries. Esses vídeos foram selecionados de forma a representar diferentes estilos visuais e condições de exibição das legendas, englobando variações na fonte, cor, tamanho e posição dos textos no vídeo. Foi utilizada a plataforma Youtube para a coleta dos vídeos.

Dos vídeos, foram extraídos trechos de até 1 minuto cada, escolhidos manualmente de forma a buscar cenas com quantidade significativa de legendas. Assim, adquiriu-se um conjunto heterogêneo de amostras legendadas. Os trechos foram processados individualmente pelo pipeline da aplicação, e os resultados foram comparados com as legendas reais dos vídeos, previamente transcritas manualmente ou adquiridas prontas, para fins de comparação.

4.5 MÉTRICAS DE AVALIAÇÃO

A avaliação da extração das legendas foi feita por meio das métricas Character Error Rate (CER) e Word Error Rate (WER), amplamente utilizadas na literatura de OCR e reconhecimento de sequência de texto. A métrica CER mede a proporção de caracteres inseridos (I), deletados (D) ou substituídos (S) em relação ao total de caracteres da transcrição correta (N), utilizando como base a distância de Levenshtein. A fórmula do CER é dada por:

$$CER = \frac{S + D + I}{N}$$

Em que:

- S = número de substituições;
- D = número de deleções;
- I = número de inserções;
- N = número total de caracteres na transcrição de referência.

De maneira análoga, a métrica WER calcula a taxa de erro considerando palavras inteiras como unidade de análise, em vez de caracteres individuais. A fórmula do WER é dada por:

$$WER = \frac{Sw + Dw + Iw}{Nw}$$

Em que:

- Sw = número de substituições de palavras;
- Dw = número de deleções de palavras;
- Iw = número de inserções de palavras;
- Nw = número total de palavras na transcrição de referência.

Enquanto o CER é mais sensível a erros de grafia e pequenas variações no reconhecimento de caracteres, o WER avalia a legibilidade em termos de palavras completas, refletindo melhor o impacto do erro na compreensão da legenda. Para o cálculo dessas métricas, será utilizada uma transcrição manual ou já pronta das legendas originais, para serem usadas como

referência. A comparação entre os textos extraídos pelo sistema e os textos corretos permitirá uma análise quantitativa do desempenho, tanto em termos de taxa de erros por caractere quanto de erros por palavra. A implementação foi feita com a biblioteca `python-levenshtein`, que oferece métodos eficientes para o cálculo da distância de Levenshtein entre sequências de caracteres ou palavras.

Por fim, apesar de não ser utilizada efetivamente nesse trabalho, é importante citar a métrica Normalized Edit Distance (NED), que é uma medida utilizada para avaliar a similaridade entre dois textos, normalizando a distância de edição (Levenshtein) pelo comprimento do texto de referência. A NED varia entre 0 e 1, em que valores próximos de 1 indicam alta similaridade. Seja $d(\text{ref}, \text{hyp})$ a distância de edição entre o texto de referência e o texto predito, e $|\text{ref}|$ o comprimento do texto de referência, a NED é definida por:

$$\text{NED} = 1 - \frac{d(\text{ref}, \text{hyp})}{|\text{ref}|}.$$

Essa normalização permite comparar resultados entre sequências de diferentes tamanhos, sendo especialmente útil na avaliação de sistemas de extração e reconhecimento de texto. Embora o NED, o CER e o WER sejam baseados na distância de Levenshtein, cada métrica avalia aspectos distintos da qualidade do texto reconhecido. O CER mede a proporção de erros por caractere, enquanto o WER quantifica erros por palavra, tornando ambas métricas orientadas à detecção de falhas pontuais no nível de caracteres ou palavras. Já o NED avalia a similaridade global entre a sequência predita e a sequência de referência, normalizando a distância de edição pelo tamanho do texto e fornecendo um valor entre 0 e 1, em que valores mais altos indicam maior correspondência. Assim, enquanto CER e WER são métricas de erro sensíveis a pequenas variações textuais, o NED captura a coerência geral da transcrição, sendo mais adequado para comparar sistemas de extração ou modelos end-to-end que operam em escala de sequência completa.

5 DESENVOLVIMENTO

Este capítulo apresenta o processo de desenvolvimento do software proposto, detalhando as decisões relacionadas aos requisitos funcionais e não funcionais, a arquitetura adotada e a organização do projeto. Também são descritas as escolhas tecnológicas, o desenho dos módulos e as heurísticas empregadas na implementação do pipeline de extração de legendas, de modo a evidenciar como a solução foi construída e estruturada para atingir os objetivos definidos.

5.1 ANÁLISE DE REQUISITOS

A definição dos requisitos do software foi orientada pela necessidade de desenvolver uma ferramenta prática e acessível, capaz de extrair e estruturar legendas a partir de vídeos de forma automática. O objetivo central do software é gerar arquivos no formato `.srt`, a partir do processamento de um vídeo fornecido como entrada.

Buscou-se garantir que o software fosse simples de utilizar, então, para isso, foram considerados aspectos como clareza na interface de linha de comando (CLI), parametrização mínima para execução e mensagens informativas durante o processo de extração.

Além da facilidade de uso, os requisitos foram definidos de modo a assegurar a precisão dos resultados, a modularidade da arquitetura e a viabilidade de futuras extensões, como a integração com diferentes modelos de OCR ou a tradução automática das legendas. A Tabela 3 apresenta os principais requisitos funcionais e não funcionais que guiaram o desenvolvimento do sistema.

Tabela 3 – Requisitos funcionais e não funcionais do sistema

Tipo de Requisito	**Descrição**
Funcionais	RF01 - O sistema deve processar vídeos e extrair legendas por meio de OCR.
	RF02 - O sistema deve gerar arquivos de saída no formato `.srt`, contendo os intervalos temporais e o texto correspondente.
	RF03 - O sistema deve permitir parametrização de execução, incluindo opções para permitir a seleção do idioma e a taxa de frames a usar para o processamento.
Não funcionais	RNF01 - O sistema deve ser implementado em linguagem Python, devido ao seu ecossistema consolidado de bibliotecas voltadas à visão computacional.
	RNF02 - O sistema deve priorizar eficiência de execução, realizando o processamento por amostragem de *frames* para reduzir o tempo total de processamento.
	RNF03 - O sistema deve garantir portabilidade para execução tanto para GPU quanto para CPU, sem exigir reconfigurações significativas.

5.2 TECNOLOGIAS E DEPENDÊNCIAS

Para o desenvolvimento da solução, foram utilizadas as seguintes bibliotecas e ferramentas:

- **PaddleOCR**: Utilizada para a detecção e reconhecimento de texto, com base em modelos pré-treinados para DBNet (detecção) e CRNN (reconhecimento);
- **OpenCV**: Responsável pela leitura e processamento dos *frames* dos vídeos;
- **wordninja-enhanced**, **pyspellchecker** e **regex**: Empregados no pós-processamento textual;
- **python-Levenshtein**: Utilizada para o cálculo das métricas de avaliação, como CER e WER.

O programa foi desenvolvido em Python 3.12, aproveitando-se de seu extenso ecossistema voltado a aplicações de visão computacional e aprendizado de máquina.

5.3 AMBIENTE DE EXECUÇÃO E REPRODUTIBILIDADE

Para garantir a reprodutibilidade dos resultados, o projeto inclui um arquivo `requirements.txt` contendo todas as dependências e versões utilizadas. A seguir, descrevem-se os passos recomendados para configurar o ambiente de execução do projeto:

1. Criar e ativar um ambiente virtual Python (venv):

```
python -m venv venv
source venv/bin/activate        # Linux / macOS
venv\Scripts\activate           # Windows
```

2. Instalar as dependências listadas no arquivo `requirements.txt`:

```
pip install -r requirements.txt
```

3. Instalar o framework PaddlePaddle, necessário para o funcionamento da biblioteca PaddleOCR. A versão a ser instalada depende do ambiente de execução:

   - Para execução em **CPU**:

```
python -m pip install paddlepaddle==3.0.0 \
-i https://www.paddlepaddle.org.cn/packages/stable/cpu/
```

- Para execução em **GPU** (por exemplo, CUDA 11.8):

```
python -m pip install paddlepaddle-gpu==3.0.0 \
-i https://www.paddlepaddle.org.cn/packages/stable/cu118/
```

4. Após a instalação do PaddlePaddle, instalar a biblioteca PaddleOCR:

```
python -m pip install paddleocr
```

Com isso, o ambiente de execução estará pronto para executar o projeto com as ferramentas e dependências necessárias.

5.4 ARQUITETURA E ORGANIZAÇÃO DO CÓDIGO

A arquitetura do projeto foi mantida propositadamente simples e modular, de modo a facilitar a execução, manutenção e reprodutibilidade dos experimentos. O software principal encontra-se implementado no arquivo `main.py`, localizado na raiz do projeto, responsável pela execução do pipeline completo de processamento e extração de legendas. Na pasta `/videos` estão organizados os vídeos de teste, as legendas de referência e o script `evaluation.py`, utilizado para a avaliação automática dos resultados. Essa estrutura busca tornar o projeto autoexplicativo e de fácil replicação, permitindo que o ambiente seja configurado e executado com o mínimo de dependências externas.

5.5 CLASSE CENTRAL E MÓDULOS

A implementação principal foi encapsulada na classe `SubtitleExtractor`, localizada no arquivo `main.py`. Essa classe concentra os atributos e métodos centrais do sistema, incluindo variáveis estáticas utilizadas como parâmetros globais de execução e instâncias das principais ferramentas empregadas no processamento, como PaddleOCR, SpellChecker e WordNinja. Dessa forma, a classe atua como núcleo funcional do programa, integrando as etapas de detecção, reconhecimento e pós-processamento textual em um fluxo unificado.

5.6 ALGORITMOS E HEURÍSTICAS

Nesta seção são descritos os algoritmos e heurísticas implementados que compõem o funcionamento do sistema, bem como as decisões de projeto que orientaram sua construção. São detalhadas as etapas de processamento, desde a extração de frames até a detecção textual, OCR e o pós-processamento textual, destacando-se as estratégias adotadas para otimizar a precisão e o desempenho do software.

5.6.1 Definição de Atributos e Variáveis Estáticas

No início da classe, são definidas variáveis estáticas que atuam como parâmetros globais para os métodos de detecção textual, calibração do posicionamento de legendas e filtragem textual. Esses valores foram ajustados empiricamente durante o desenvolvimento, de modo a equilibrar desempenho e precisão. As funções de cada variável são descritas a seguir:

- **WHITESPACE**: Representa o caractere de espaço, utilizado em operações de concatenação e formatação textual nas etapas de reconstrução das legendas.

```
WHITESPACE = " "
```

- **CALIBRATION_FRAMES**: Define o número de frames utilizados para calibrar a posição média das legendas no vídeo a partir de um certo ponto de partida. Essa calibração permite identificar de forma estável a região onde o texto costuma aparecer, reduzindo falsos positivos. O número escolhido foi encontrado empiricamente e mostrou-se bom o suficiente para capturar uma quantia considerável de legendas para todos os vídeos testados. Dessa forma, essa variável foi suficiente para determinar a posição média das legendas para todos os casos de teste.

```
CALIBRATION_FRAMES = 600
```

- **TEXT_LENGTH_THRESHOLD**: Estabelece o número mínimo de caracteres que um texto deve possuir para ser considerado uma legenda válida. Essa variável é utilizada no processo de calibração, assim como o `CALIBRATION_FRAMES`, e serve como uma restrição que evita que fragmentos curtos e ruídos visuais sejam processados como legendas no processo de calibração.

```
TEXT_LENGTH_THRESHOLD = 20
```

- **POSITION_TOLERANCE**: Determina a margem de tolerância, em *pixels*, para considerar uma detecção textual como estando dentro da posição média de legendas definida no processo de calibração. A escolha para o valor dessa variável foi motivado por experimentos que mostraram que um pequeno valor era suficiente para as variações de posicionamento das legendas.

```
POSITION_TOLERANCE = 50
```

- **TOKENIZER**: Define a expressão regular responsável pela separação das legendas em tokens, reconhecendo palavras, números e sinais de pontuação (incluindo caracteres acentuados). Essa etapa serve como base para as fases de pós-processamento textual.

```
TOKENIZER = re.compile(r"[A-Za-zÀ-ÿ0-9']+|[.,!?;:\"“”‘’]")
```

O construtor da classe (`__init__`) recebe os parâmetros principais `video_path`, `language` e `target_fps`, e instancia os componentes necessários para o processamento. O trecho de código a seguir ilustra sua estrutura:

```
def __init__(self, video_path, language, target_fps = 10):
    self.video_path = video_path
    self.language = language
    self.target_fps = target_fps
    self.ocr = PaddleOCR()
    self.spell_en = SpellChecker(language="en")
    self.spell_pt = SpellChecker(language="pt")
    self.wordninja_lm = wordninja_enhanced
```

A instância do PaddleOCR contêm os módulos de detecção e reconhecimento textual (DBNet e CRNN), enquanto SpellChecker (em inglês e português) é utilizado para validação das palavras no pós-processamento textual. A ferramenta WordNinja, por sua vez, é empregada para segmentar palavras concatenadas. Esses componentes, configurados na inicialização, compõem a base funcional para os métodos subsequentes da classe.

5.6.2 Métodos Auxiliares

A seguir, são descritos os métodos auxiliares implementados na classe principal. Essas funções desempenham papel fundamental no pré e pós-processamento textual, garantindo que as legendas extraídas passem por etapas de padronização e correção antes de sua consolidação.

- **simple_tokenize(text)**: Este método tem como finalidade realizar a tokenização simples de uma sequência textual, retornando uma lista de palavras e sinais de pontuação. Ele utiliza o padrão definido pelo regex `TOKENIZER`, capaz de reconhecer caracteres alfanuméricos, acentuação e símbolos de pontuação. Essa etapa permite que os textos extraídos sejam manipulados de forma estruturada.

```
def simple_tokenize(text)
    tokens ← TOKENIZER.findall(text)
    return tokens
```

- **fix_special_cases(text)**: Este método aplica uma série de substituições baseadas em regex para corrigir erros recorrentes de espaçamento e pontuação nas saídas de OCR. As correções incluem, por exemplo, remoção de espaços indevidos em contrações (“it ’s” → “it’s”). Esse método auxiliar colabora significativamente para a legibilidade e coerência das legendas processadas.

```
def fix_special_cases(text)
    text ← replace("\b(\w+)\s+'(\w+)", "\1'\2", text)
    text ← replace("\s+([,!.?;:])", "\1", text)
    return text
```

- **split_concatenated_words(text)**: Este é o método principal usado no pós-processamento textual, unindo os outros métodos auxiliares já citados. Ele recebe como entrada um texto puro do OCR e realiza uma sequência de passos, sendo eles:

  1. Tokenização do texto de entrada

     Para cada token extraído do passo 1, roda o laço:

  2. Verifica, por regex, se token é combinação de letras e pontuações (palavra) ou se é somente pontuação
  3. Se token é uma pontuação, adiciona diretamente à lista final
  4. Se token é uma palavra, segue para o resto da função, a seguir
  5. Se token identificado como palavra está presente no dicionário do spellchecker, adiciona à lista final, se não, segue para a verificação final
  6. Se token identificado como palavra não está presente no dicionário, tenta separar token com wordninja para ver se há concatenação de palavras e, se bem sucedido, adiciona as palavras separadas para a lista final, terminando o laço
  7. Por fim, roda a função `fix_special_cases()` no texto final já concatenado com `WHITESPACE`, buscando corrigir possíveis problemas de espaçamento e pontuações. Assim, retorna o resultado final da função

  Esse método tem a função principal de separar palavras concatenadas e preservar palavras e pontuações legítimas vindos do OCR.

- **format_time_srt(seconds)**:

  Este método tem como objetivo converter um valor de tempo em segundos (representado como número decimal) para o formato utilizado em arquivos de legenda, que segue o modelo `HH:MM:SS,mmm`, onde `HH` representa as horas, `MM` os minutos, `SS` os segundos e `mmm` os milissegundos. Essa função é utilizada no processo de geração das legendas

extraídas, permitindo que cada bloco de texto reconhecido seja associado a um intervalo de tempo preciso. Por exemplo, ao criar um novo segmento de legenda, o programa define os campos `start_time` e `end_time` chamando a função `format_time_srt()` sobre os *frames* de início e fim, convertidos em segundos.

- **normalize_text(text)**:

  Este método é responsável por padronizar o texto antes de comparações. Ele realiza a normalização de caracteres (removendo acentuação), converte todas as letras para minúsculas e elimina pontuações e espaços excessivos. Esse processo garante consistência na análise de similaridade entre legendas, independentemente de diferenças de formatação ou caracteres especiais.

- **jaccard_similarity(a, b)**:

  Este método calcula o índice de similaridade de Jaccard entre dois textos, considerando o conjunto de palavras que eles contêm. O resultado é um valor entre 0 e 1 que representa o grau de sobreposição entre os textos, sendo 1 quando ambos são idênticos. Esse método é utilizado como uma métrica auxiliar para determinar se duas legendas correspondem ao mesmo conteúdo textual.

- **check_if_same_text(current_text, last_text, lev_threshold, jaccard_threshold)**:

  Este método combina as diferentes abordagens citadas anteriormente para verificar se dois textos podem ser considerados equivalentes. Primeiro, realiza-se a normalização de ambos os textos e, em seguida, aplicam-se duas métricas de similaridade: a razão de Levenshtein e o índice de Jaccard. Se qualquer uma das métricas ultrapassar os limiares definidos, os textos são considerados iguais. Essa verificação é utilizada para evitar duplicação de legendas de frames consecutivos que possam apresentar pequenas variações.

- **is_text_legible(text, threshold)**:

  Este método tem como objetivo verificar se um texto extraído é composto majoritariamente por palavras válidas, distinguindo legendas reais de ruídos ou falhas de leitura. O método utiliza o dicionário correspondente da biblioteca `pyspellchecker` para validar cada palavra, calculando a proporção de termos reconhecidos em relação ao total de palavras detectadas. Caso essa proporção seja igual ou superior a um *threshold* de 0,75, o texto é considerado legível. O valor de limiar adotado foi determinado empiricamente a partir de experimentos, observando que valores inferiores permitiam falsos positivos e valores superiores descartavam legendas legítimas.

- **is_noise(subtitle, min_duration)**:

  Este método avalia se um bloco de legenda detectado corresponde a ruído ou não, usando como base a duração e a legibilidade do conteúdo, com auxílio da função `is_text_legible()`. Algumas premissas para o funcionamento desse método são:

- Sabe-se que esse método recebe um objeto que contêm o conteúdo textual da legenda e sua duração, com `start_time` e `end_time`.
- Além disso, sabe-se que o método utiliza dois valores de *threshold* para seu funcionamento, sendo eles o `min_duration` e `legibility_threshold`. O valor adotado para o `min_duration`, de 0,3 segundos, foi definido empiricamente após testes, nos quais observou-se que todos os fragmentos de texto com duração inferior a esse valor correspondiam a ruídos visuais ou falhas de detecção.
- A função também ajusta o limiar de legibilidade (`legibility_threshold`) conforme a duração da legenda. O valor inicial definido é de 0,75 mas pode aumentar caso a duração da legenda seja menor que o threshold de duração (`min_duration`). Assim, esse limiar é usado como parâmetro para a execução da função `is_text_legible()` com o *threshold* de legibilidade.

Tendo isso em mente, o método segue um passo a passo para a definição de um bloco de texto como ruído ou não:

1. A função inicia calculando a duração da legenda por meio da subtração do `end_time` pelo `start_time`;
2. Tendo a duração da legenda, é feito uma verificação com o `min_duration`. Se a duração for menor ou igual ao limiar de 0,3 segundos, a verificação continua, caso contrário, descarta-se a possibilidade de ser ruído e considera-se o texto como potencialmente legítimo, finalizando;
3. Caso continue o fluxo de execução, é feita uma verificação que condicionalmente aumenta o valor do `legibility_threshold` para 0,95 caso a duração da legenda seja menor que 0,25 segundos, por ser abaixo do limiar de `min_duration`. Isso tem como objetivo tornar a verificação mais rígida para blocos de legendas de duração extremamente curtas, por muito provavelmente serem ruídos;
4. Por fim, o método finaliza retornando o resultado booleano da função `is_text_legible(text, threshold)`, fornecendo o texto da legenda e o limiar `legibility_threshold` como parâmetros para execução dessa função.

Esses métodos auxiliares têm grande impacto na qualidade final das legendas geradas, por servirem principalmente para a etapa de pós-processamento textual. Além disso, também auxiliam no funcionamento dos métodos principais, que serão detalhados em seguida.

5.6.3 Métodos Principais

Nessa seção, são descritos os principais métodos que compõem o núcleo funcional do sistema, responsáveis por operacionalizar o processo de extração e geração das legendas. São descritos três componentes fundamentais: o método de calibração da posição das legendas, que

determina a região média do vídeo onde os textos aparecem; o método de extração de texto dos *frames*, responsável por aplicar o OCR e pós-processar os resultados; e o pipeline principal, que integra todas as etapas do processo, desde a leitura dos frames e extração das legendas até a formatação e exportação final do arquivo `.srt`. Esses métodos, em conjunto, definem o comportamento central do software e garantem a automação do fluxo de reconhecimento e reconstrução textual.

- **calibrate_subtitle_position(video, num_frames)**:

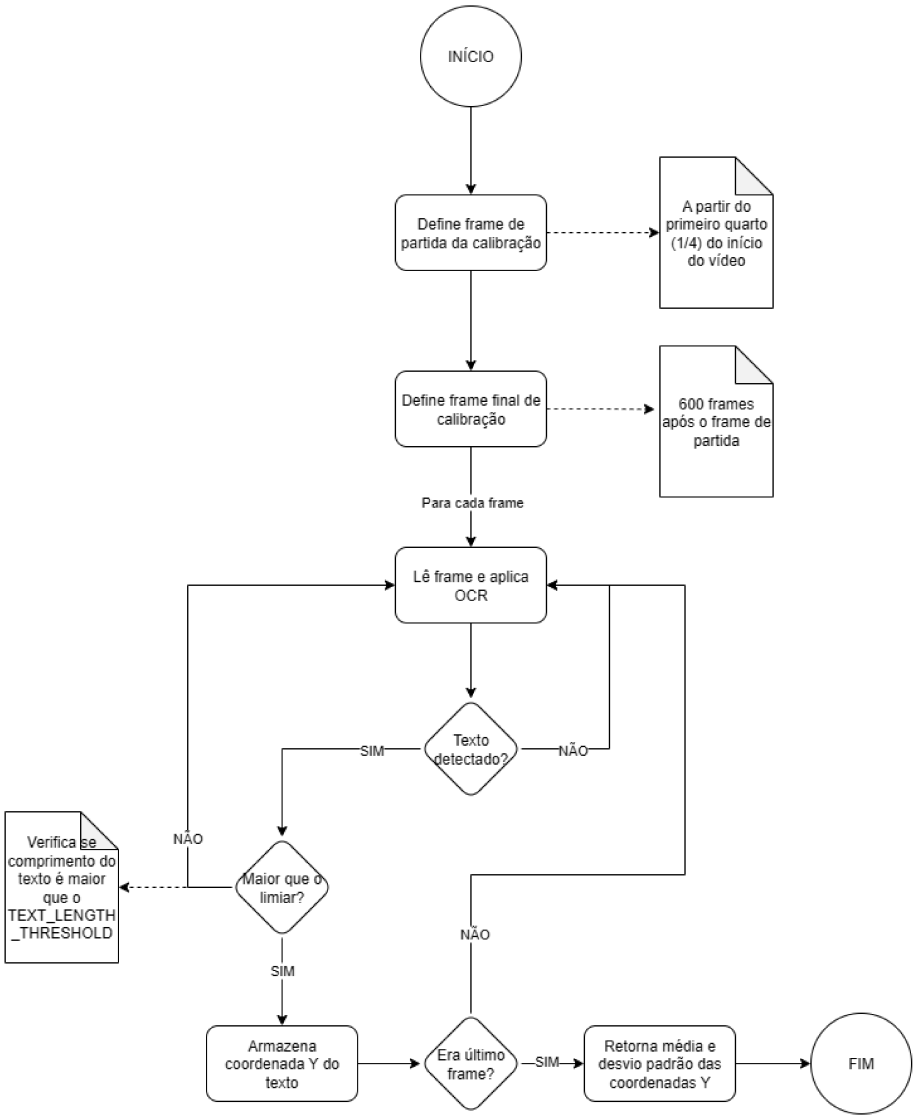
  Este método tem como objetivo estimar a posição média em que as legendas aparecem no vídeo, de forma automática, antes do início do processamento principal. Essa etapa de calibração é fundamental para que o sistema identifique, de maneira eficiente, a região mais provável de ocorrência das legendas, reduzindo o número de falsos positivos e otimizando o tempo de processamento.

  O método realiza uma varredura em um subconjunto de *frames*, utilizando, por padrão, o valor de 600 *frames* definido pela variável estática `CALIBRATION_FRAMES`. Assim, a calibração toma como partida o primeiro quarto do vídeo e avança de acordo com o número de *frames* definidos, sobre os quais o OCR é executado para identificar textos e suas posições na tela.

  A partir dos resultados de detecção, o algoritmo coleta as coordenadas verticais (y) dos blocos de texto que possuem comprimento superior a um limiar mínimo, definido pela variável estática `TEXT_LENGTH_THRESHOLD`, com o intuito de evitar que ruídos ou textos do ambiente do vídeo influenciem na calibração. Com base nesses valores, calcula-se a média e o desvio padrão da posição vertical das detecções, retornando esses dois indicadores para uso posterior na filtragem dos textos detectados no processamento principal.

  Em termos conceituais, o algoritmo pode ser descrito conforme o fluxograma da figura 17:

Figura 17 – Fluxograma de calibração do posicionamento das legendas



Fonte: O autor.

Ao final, são retornadas duas medidas, a média e o desvio padrão da posição vertical dos textos detectados, que servem como base para filtrar as detecções textuais no processamento principal, considerando uma tolerância em pixels definida pela variável estática `POSITION_TOLERANCE`.

- **extract_text_from_frame(image, avg_y_pos, std_y_pos, k):**

  Este método é responsável por realizar a extração textual presente em um *frame* do vídeo, utilizando os módulos de detecção e reconhecimento de texto do PaddleOCR. Essa etapa é essencial para identificar e recuperar as legendas embutidas no vídeo e realizar o pós-processamento textual em seguida.

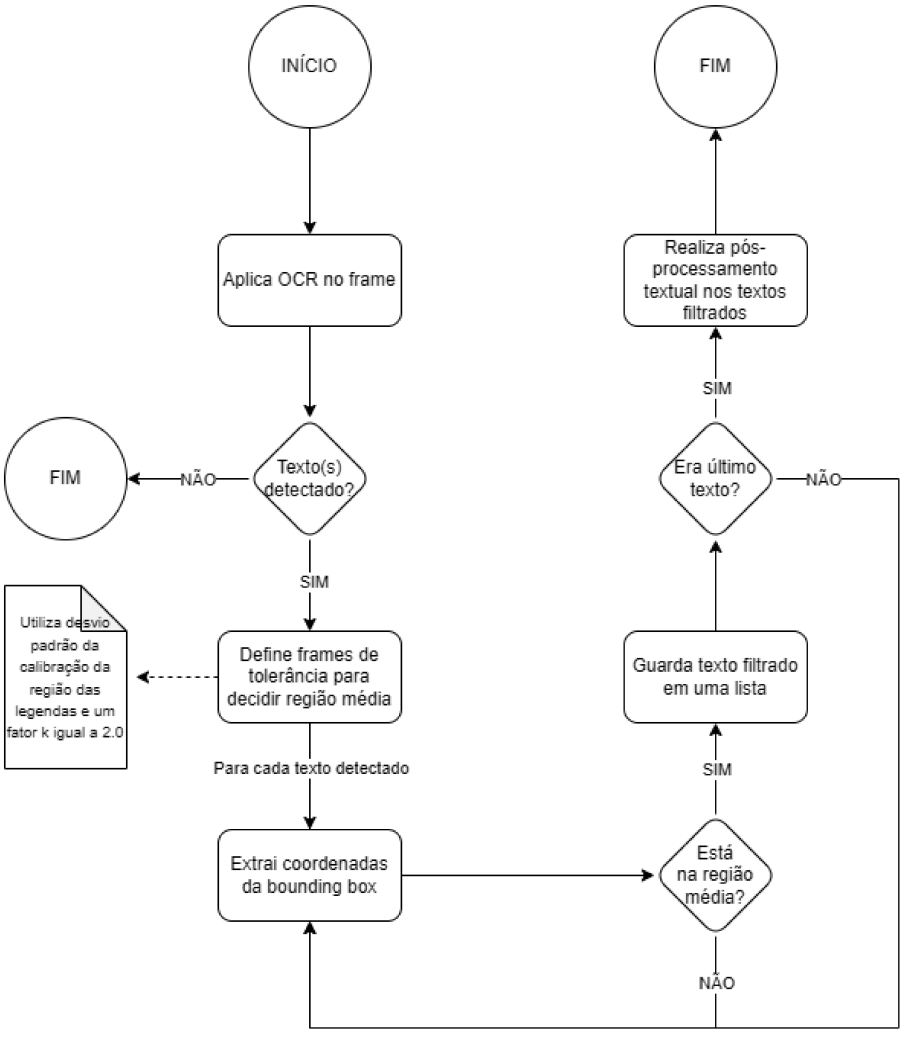
  O processo inicia com a execução do método `predict()` da biblioteca, que executa duas tarefas principais: a detecção das regiões que contêm texto e o reconhecimento dos carac-

teres nelas presentes. O retorno dessa etapa consiste em uma lista de textos reconhecidos e suas respectivas coordenadas na imagem.

Em seguida, o método utiliza a posição média das legendas (obtida no processo de calibração anterior) para filtrar textos que realmente pertencem à região das legendas. Essa filtragem é feita com base em uma tolerância ajustável, determinada pelo desvio padrão da posição média, multiplicado por um fator k. Assim, textos fora da faixa esperada, como logotipos, marcas d'água ou textos em outras partes do vídeo, são descartados.

Após essa filtragem, o método aplica a etapa de pós-processamento textual por meio do método auxiliar `split_concatenated_words()`, que tem o papel de corrigir palavras que eventualmente foram reconhecidas como um único bloco concatenado. Por fim, os textos resultantes são unidos e retornados como uma string final, representando a legenda detectada naquele *frame*. Em termos conceituais, o algoritmo pode ser descrito conforme o fluxograma da figura 18:

Figura 18 – Fluxograma de extração textual de frame



Fonte: O autor.

Este método garante um equilíbrio entre precisão e desempenho, ao combinar o OCR com

heurísticas espaciais e linguísticas, assegurando que apenas textos relevantes e legíveis sejam mantidos.

- **run():**

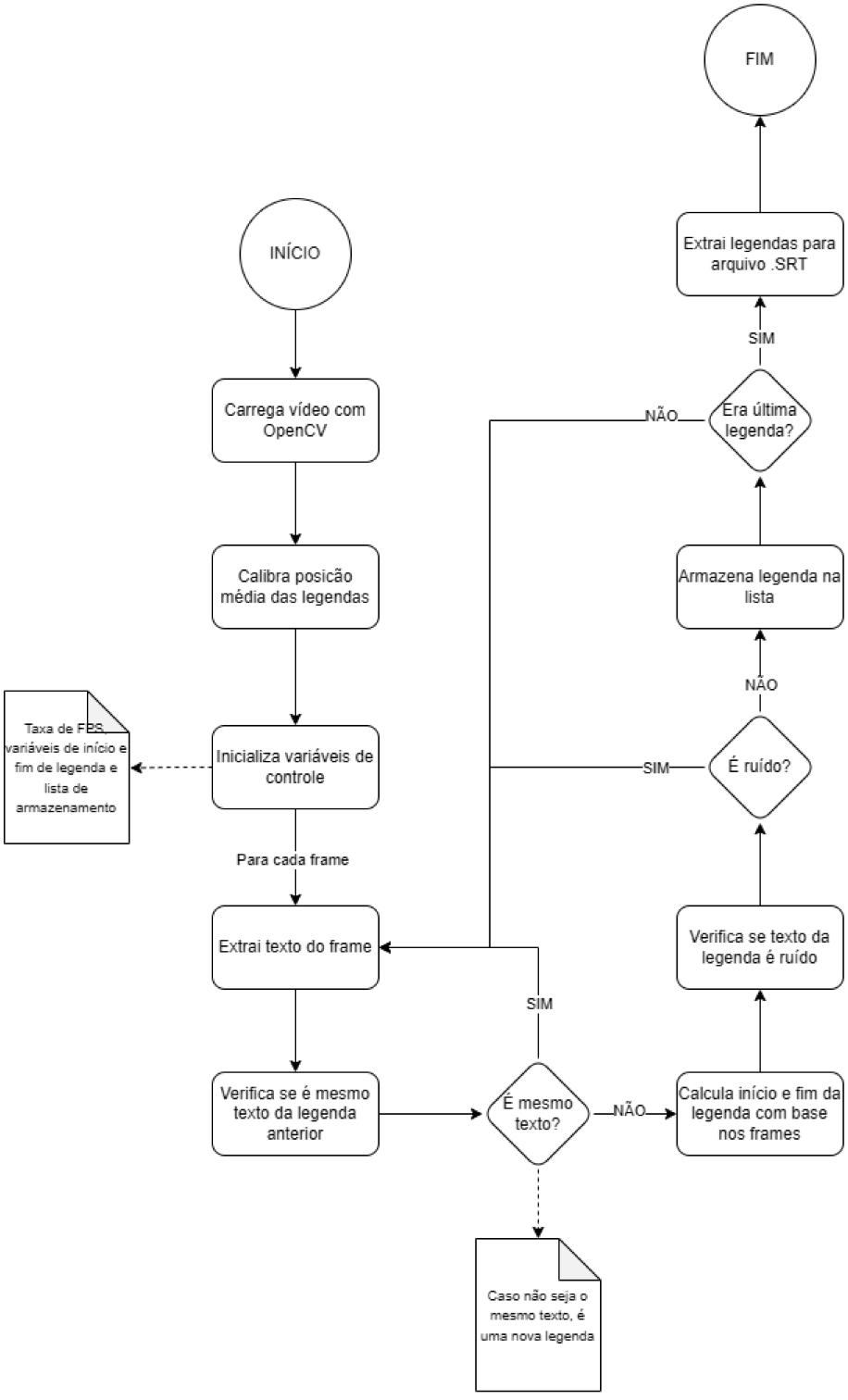
  Este método representa o núcleo do sistema, sendo responsável por integrar todas as etapas do pipeline de extração de legendas, desde a leitura do vídeo até a geração final do arquivo `.srt`. Ele coordena os processos de calibração, detecção, filtragem e armazenamento das legendas, garantindo o fluxo lógico e temporal entre os *frames* analisados. A seguir, apresenta-se uma visão geral passo a passo de seu funcionamento:

  1. **Carregamento do vídeo:** Inicialmente, o arquivo de vídeo a ser processado é carregado utilizando o OpenCV. Caso o vídeo não possa ser carregado corretamente, o programa é encerrado com uma mensagem de erro.
  2. **Calibração da posição das legendas:** É realizada a calibração automática para determinar a posição média das legendas ao longo do vídeo, através do método `calibrate_subtitle_position()`. Caso essa calibração falhe, o programa desativa o filtro posicional e passa a analisar os *frames* completos por textos.
  3. **Inicialização das variáveis de controle:** São inicializadas variáveis para controlar fatores de funcionamento do programa como:
     a) O intervalo de frames a processar conforme a taxa de amostragem definida pelo `target_fps` (Passado como argumento na execução por command-line);
     b) *Frames* de início e fim de um bloco de legenda (Para calcular duração);
     c) Armazenamento das legendas extraídas na variável `subtitle_data`, para posterior extração para arquivo `.srt`.
  4. **Loop principal de processamento:** O vídeo é percorrido conforme a taxa de FPS configurada e segue o fluxo:
     a) O *frame* atual é lido e alimentado ao método `extract_text_from_frame()`, que utiliza o PaddleOCR para detectar e reconhecer textos;
     b) Caso o texto detectado seja semelhante ao anterior, avaliado por meio do método `check_if_same_text()`, o programa considera que a legenda ainda está ativa;
     c) Quando o texto muda, o programa define o fim da legenda anterior, convertendo os tempos de início e fim utilizando o método `format_time_srt()`;
     d) Além disso, é feito uma verificação para ver se a legenda em questão é ruído ou não, por meio do método `is_noise()`;
     e) Se o texto da legenda for válido, o bloco é salvo na lista criada, em `subtitle_data`.
  5. **Geração do arquivo `.srt`:** Por fim, todas as legendas armazenadas são escritas em um arquivo `subtitles.srt`, seguindo o formato padrão de numeração, tempo e texto.

6. **Liberação de recursos:** O objeto de vídeo é liberado da memória e o programa exibe uma mensagem de sucesso indicando que a extração foi concluída.

Em termos conceituais, o algoritmo pode ser descrito conforme o fluxograma da figura 19:

Figura 19 – Fluxograma do processamento principal



Fonte: O autor.

Com a execução do método `run()`, conclui-se o fluxo principal do sistema, que integra todas as etapas de detecção, reconhecimento, filtragem e reconstrução das legendas. Esse método representa a síntese dos componentes desenvolvidos, unindo todos os módulos, desde o carregamento do vídeo até a geração do arquivo final no formato `.srt`. Assim, é através dele que o objetivo central do software é alcançado: a extração automática e estruturada de legendas embutidas em vídeos, de forma reprodutível e independente de intervenção manual.

5.7 INTERFACE DE LINHA DE COMANDO (CLI)

O programa foi projetado para ser executado diretamente pelo terminal, utilizando uma interface de linha de comando (CLI) desenvolvida com o módulo `argparse` da linguagem Python. Essa interface permite que o usuário informe os parâmetros necessários para a execução do processo de extração das legendas de forma simples e estruturada.

O método `parse_args()` é responsável por criar e configurar o objeto de parser, que interpreta os argumentos fornecidos na execução do programa. Os parâmetros esperados são:

- **video_path**: Caminho do vídeo que será processado. É um argumento obrigatório, e deve apontar para um arquivo de vídeo válido no sistema de arquivos.
- **language**: Idioma principal das legendas embutidas no vídeo. O programa aceita duas opções: `"en"` (inglês) ou `"pt"` (português), ajustando automaticamente o dicionário de palavras (`pyspellchecker`) conforme o idioma selecionado.
- **target_fps**: Taxa de FPS utilizada na análise. Este parâmetro é opcional, possuindo o valor padrão de 10. Ele define a frequência com que os *frames* são amostrados do vídeo, influenciando diretamente o desempenho e a precisão do processo.

Uma vez interpretados os argumentos, o programa instancia a classe `SubtitleExtractor`, passando os parâmetros recebidos, e em seguida invoca o método principal `run()`. Esse método inicia todo o pipeline de processamento descrito nas seções anteriores.

Caso o usuário execute o programa sem fornecer os argumentos necessários, o módulo `argparse` exibe automaticamente uma mensagem de ajuda no terminal, explicando o formato correto de uso e os parâmetros disponíveis. O bloco de código a seguir ilustra um exemplo de execução da ferramenta via linha de comando:

```
$ python main.py video.mp4 en
```

Nesse exemplo, o programa processa o arquivo `video.mp4`, assumindo o idioma inglês e utilizando a taxa de FPS padrão definida.

6 RESULTADOS

Neste capítulo, são apresentados e analisados os resultados obtidos a partir da execução do software em um conjunto de vídeos de teste. Para avaliar a eficácia do sistema, foram utilizados vídeos em língua inglesa, contendo legendas com diferentes estilos, contrastes e padrões visuais.

A escolha por vídeos em inglês se justifica por dois fatores principais: primeiramente, o modelo de reconhecimento textual utilizado no PaddleOCR é otimizado especificamente para idiomas como inglês, chinês e japonês, o que torna o inglês a opção mais adequada para garantir compatibilidade e precisão durante a avaliação; além disso, o inglês é amplamente reconhecido como língua universal e concentra grande parte do conteúdo audiovisual disponível em plataformas digitais. Essa ampla oferta facilita a seleção de vídeos representativos e diversificados, permitindo testar o sistema em cenários realistas.

Dessa forma, a precisão dos textos extraídos foi mensurada por meio das métricas CER e WER, amplamente utilizadas em tarefas de OCR. Essas métricas quantificam o nível de erro em relação às legendas originais, oferecendo uma visão objetiva da qualidade dos resultados produzidos pelo software.

6.1 EXEMPLO DE APLICAÇÃO

Antes de apresentar os resultados, é importante visualizar o funcionamento prático do sistema. Assim, esta seção apresenta um exemplo de aplicação do programa em *frames* isolados e em um vídeo fictício legendado criado pelo autor. O objetivo é ilustrar o comportamento do pipeline completo, desde a leitura do vídeo e detecção das legendas até a geração do arquivo `.srt` final, evidenciando a capacidade do software em identificar, processar e reconstruir legendas embutidas de forma autônoma.

6.1.1 Execução em frames com diferentes estilos de legenda

Uma das qualidades da aplicação é a capacidade de extrair legendas independente de estilo, tamanho ou cor, como mostram os exemplos das figuras 20 a 23. As figuras em questão mostram as imagens alvo e resultado, pós execução do PaddleOCR, respectivamente:

1. Exemplo de legenda com coloração branca e caixa baixa:

Figura 20 – Exemplo de aplicação do PaddleOCR para legendas brancas



Fonte: O autor.

Figura 21 – Resultado de aplicação do PaddleOCR para legendas brancas



Fonte: O autor.

2. Exemplo de legenda com coloração amarela e caixa alta:

Figura 22 – Exemplo de aplicação do PaddleOCR para legendas amarelas



Fonte: O autor.

Figura 23 – Resultado de aplicação do PaddleOCR para legendas amarelas



Fonte: O autor.

6.1.2 Execução completa para vídeo fictício

Para demonstrar o funcionamento da execução completa do programa, foi criado um vídeo com legendas fictícias. Dessa forma, sabendo-se que o vídeo tem cerca de 30 segundos e cerca de 5 blocos de legendas, o resultado final foi um arquivo `.srt` gerado com as legendas sincronizadas:

```
1
```

```
00:00:00,000 --> 00:00:01,568
In the ' video industry, burned in subtitles

2
00:00:01,601 --> 00:00:04,237
refer to text embedded per manet ly into a video file

3
00:00:04,270 --> 00:00:06,906
Unlike traditional subtitles, burned in subtitles are not separate files.

4
00:00:06,940 --> 00:00:09,576
They are part of the video file itself.

5
00:00:09,609 --> 00:00:09,642
Your captions are visible and always cannot be turned off removed. or

6
00:00:09,676 --> 00:00:12,979
Your captions are always visible and cannot be turned off removed. or
```

Tendo em vista o resultado da extração, é possível notar que, apesar de compreensíveis, as legendas extraídas possuem algumas imperfeições, como ausência de caracteres, posicionamento errado de palavras e blocos de legendas a mais. Tais problemas expõem alguns dos desafios enfrentados e limitações encontradas durante a implementação e testes da solução desenvolvida.

6.2 DESAFIOS E LIMITAÇÕES

Durante o processo de implementação e testes da ferramenta, diversos desafios foram identificados, principalmente relacionados à natureza dos vídeos analisados e às limitações inerentes à solução desenvolvida. Alguns desses desafios impactam diretamente na precisão da extração das legendas, sendo os principais: interferências visuais provenientes do próprio conteúdo do vídeo e a dificuldade de segmentação em cenários com baixo contraste entre a legenda e o fundo.

6.2.1 Textos Ambientais na Região das Legendas

Um dos desafios mais significativos ocorreu em vídeos que apresentavam textos inseridos no próprio cenário ou fundo, estando localizados na mesma região média onde normalmente aparecem as legendas. Exemplos comuns incluem placas, letreiros ou interfaces gráficas sobrepostas ao vídeo. Esses textos induziam falsos positivos durante as etapas de calibração e extração, uma vez que o algoritmo os interpretava como possíveis legendas embutidas.

Para mitigar o problema, a combinação dos mecanismos de verificação de ruído e de validação da posição vertical dos textos mostrou-se suficiente na maioria dos casos. Em situações excepcionais, no entanto, ainda ocorria a detecção e extração de textos do cenário como legendas. Apesar dessa limitação, tais ocorrências não comprometiam a compreensão do conteúdo principal, já que as legendas legítimas continuavam sendo corretamente extraídas.

6.2.2 Baixo Contraste entre Legenda e Fundo

Outro obstáculo recorrente foi a presença de legendas com cores que se confundiam com o fundo do vídeo, especialmente em cenas muito claras ou muito escuras. Nessas situações, o OCR tinha dificuldade em distinguir os contornos das letras, resultando em perdas parciais ou erros na detecção. Esse desafio evidenciou a importância do pré-processamento de imagem, como ajustes de brilho, contraste e binarização, para melhorar a separabilidade entre o texto e o plano de fundo. Embora algumas estratégias sejam automaticamente aplicadas pelo PaddleOCR, o desempenho do OCR ainda variou de acordo com as condições visuais específicas de cada vídeo.

6.3 TESTES E VALIDAÇÃO DO SISTEMA

A etapa de testes e validação teve como objetivo avaliar o desempenho do sistema desenvolvido na extração de legendas embutidas em diferentes tipos de vídeos. Foram realizados experimentos com um conjunto de 20 vídeos em inglês, contendo variações de resolução, contraste e estilos de legenda, a fim de verificar a robustez e generalização do algoritmo. Para mensurar a precisão da extração textual, foram utilizadas as métricas CER e WER, amplamente empregadas em tarefas de OCR e transcrição automática. Os resultados obtidos permitiram quantificar a qualidade das legendas extraídas e identificar padrões de erro relevantes para futuras melhorias do sistema.

6.3.1 Ambiente de Execução e Reprodutibilidade

O sistema foi avaliado em uma máquina com as seguintes especificações:

- **Processador**: AMD Ryzen 7 5700X (8 núcleos);

- **Memória RAM**: 32 GB;
- **GPU**: NVIDIA GeForce GTX 1080;
- **Sistema Operacional**: Windows 10;
- **Linguagem e Versão**: Python 3.12.

6.3.2 Métricas CER/WER

Para avaliar a precisão da extração das legendas embutidas, foram utilizadas as métricas Character Error Rate (CER) e Word Error Rate (WER). Essas métricas mensuram, respectivamente, a proporção de caracteres e palavras incorretamente reconhecidos em relação ao texto de referência. Além disso, elas foram aplicadas sob dois critérios distintos de rigor: *relaxed* e *strict*.

A diferença entre os modos *relaxed* e *strict* está no grau de tolerância aos desvios entre o texto extraído e o texto esperado. No modo *relaxed*, pequenas variações, como diferenças em pontuação, espaçamento, acentuação ou capitalização, são desconsideradas, oferecendo uma visão mais flexível do desempenho do sistema. Já no modo *strict*, toda divergência textual, mesmo um simples ponto final ausente, uma vírgula trocada ou uma letra maiúscula indevida, é contabilizada como erro.

Essa distinção é relevante, pois no contexto de legendas, variações sutis podem não comprometer a compreensão humana, mas penalizam fortemente a métrica *strict*. Por exemplo, se o texto de referência for “I’m fine.” e o sistema reconhecer “Im fine”, o modo *relaxed* pode ignorar a ausência do apóstrofo e do ponto final, considerando o resultado quase perfeito, enquanto o modo *strict* registraria duas falhas distintas. Assim, a análise comparativa entre as duas abordagens permite avaliar tanto a robustez do sistema em termos práticos, quanto sua precisão literal em termos de fidelidade textual.

- **Resultados Antes do Pós-Processamento**

  Para mensurar o impacto direto das heurísticas desenvolvidas para o pós-processamento textual, o sistema foi avaliado primeiramente em sua forma bruta, isto é, utilizando apenas a detecção e o reconhecimento óptico, sem qualquer etapa adicional de correção ou filtragem. Nesse cenário, foram obtidas taxas médias de erro significativamente superiores: **CER = 0,0538** (5,38%) e **WER = 0,2890** (28,9%) no modo *relaxed*, além de **CER = 0,0586** (5,86%) e **WER = 0,3084** (30,84%) no modo *strict*. Esses valores refletem o desempenho cru dos modelos utilizados ao lidar com legendas de diferentes características visuais, com variação de contraste ou com segmentação irregular.

  A tabela 4 apresenta os valores individuais por vídeo, permitindo, posteriormente, comparar com os resultados finais. Observa-se que, em vários casos, especialmente nos vídeos com maior ruído visual ou legendas estilizadas, o WER ultrapassa 50% sem pós-

processamento, indicando que mais da metade das palavras extraídas sofrem algum tipo de divergência em relação ao texto de referência.

Tabela 4 – Métricas CER e WER sem pós-processamento (modos Relaxed e Strict)

Vídeo	CER (Relaxed)	WER (Relaxed)	CER (Strict)	WER (Strict)
1	0,0712	0,5138	0,0732	0,5315
2	0,0686	0,4696	0,0659	0,4696
3	0,0070	0,0649	0,0408	0,1595
4	0,0383	0,1881	0,0375	0,1980
5	0,0437	0,3415	0,0450	0,3735
6	0,0176	0,1333	0,0271	0,1600
7	0,0758	0,4835	0,0724	0,4835
8	0,1440	0,7965	0,1493	0,8052
9	0,0239	0,2087	0,0221	0,2087
10	0,0369	0,3205	0,0386	0,3333
11	0,1064	0,2135	0,1196	0,2660
12	0,0832	0,4054	0,0874	0,4234
13	0,0539	0,0897	0,0612	0,1310
14	0,0074	0,0769	0,0072	0,0769
15	0,0378	0,1281	0,0382	0,1281
16	0,1033	0,7094	0,0977	0,7350
17	0,0029	0,0199	0,0027	0,0199
18	0,0096	0,0710	0,0092	0,0710
19	0,1195	0,5771	0,1288	0,6175
20	0,0361	0,2025	0,0402	0,2025
Média	**0,0538 (5,38%)**	**0,2890 (28,9%)**	**0,0586 (5,86%)**	**0,3084 (30,84%)**

Fonte: O autor.

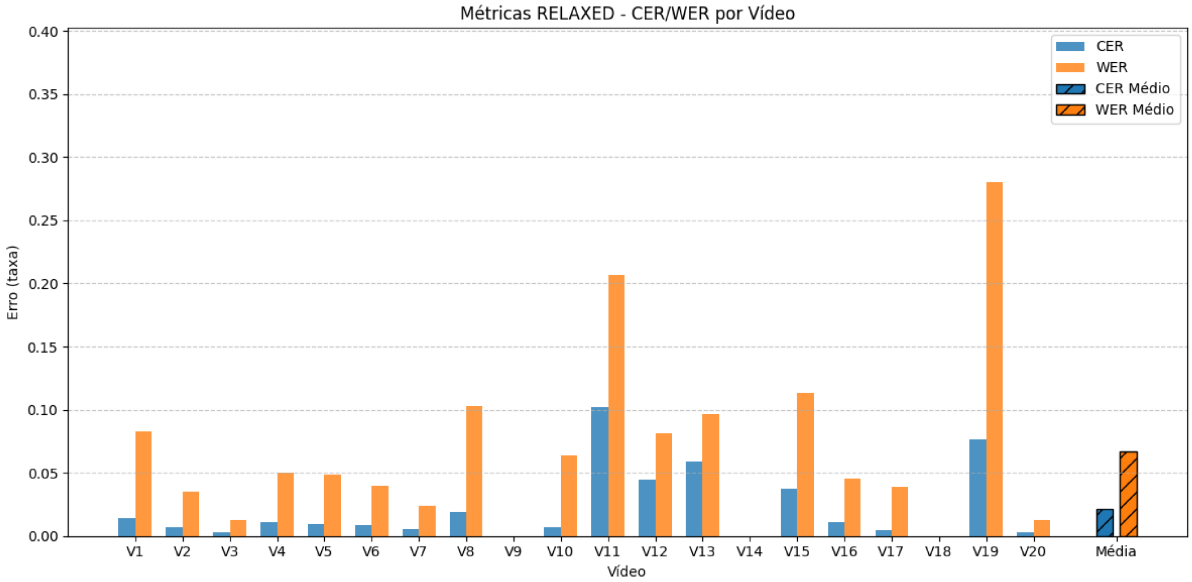
- **Resultados Depois do Pós-Processamento (Finais)**

  Em contrapartida, foram mensurados os resultados da avaliação das métricas com a aplicação do módulo de pós-processamento textual do programa, representando os resultados finais. Em comparação direta com as métricas antes dessa etapa, evidencia-se uma melhora substancial: tanto o CER quanto o WER foram significativamente reduzidos nos modos *relaxed* e *strict*.

  - **CER/WER Relaxed**

    O gráfico da figura 24 apresenta os resultados obtidos para as métricas sob o critério *relaxed*. Observa-se que as taxas médias de erro foram **CER = 0,0211** (2,1%) e **WER = 0,0667** (6,7%).

Figura 24 – Resultado métricas de avaliação - Relaxed

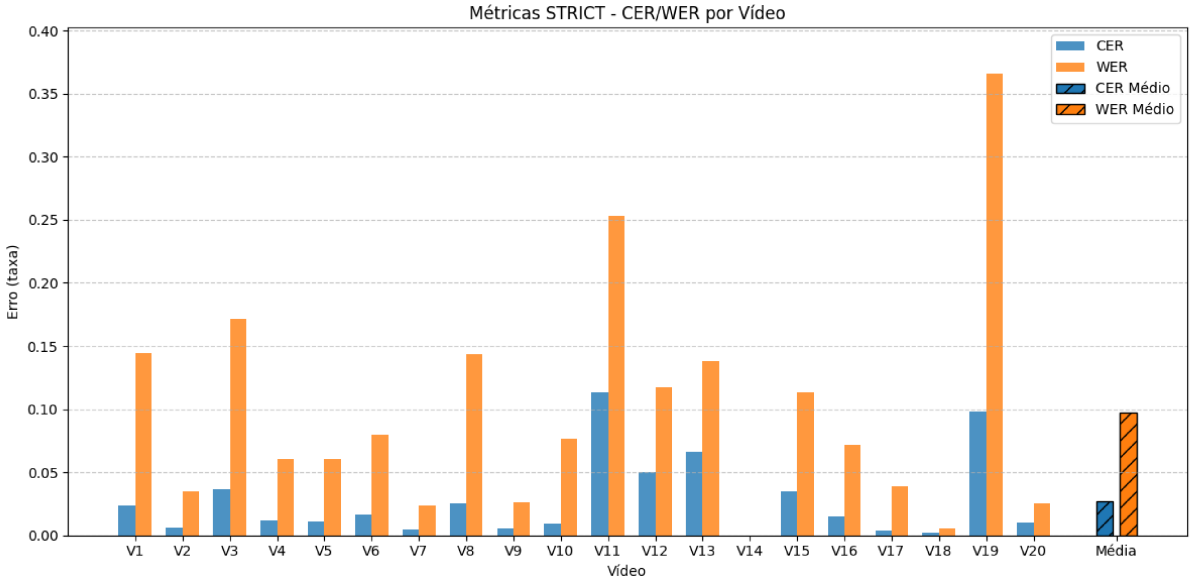


Fonte: O autor.

– **CER/WER Strict**

A figura 25 apresenta as mesmas métricas aplicadas sob o critério *strict*, no qual qualquer divergência textual é penalizada integralmente. Nesse cenário mais rigoroso, as taxas médias de erro foram **CER = 0,0273** (2,7%) e **WER = 0,0976** (9,8%).

Figura 25 – Resultado métricas de avaliação - Strict



Fonte: O autor.

– Tabela Comparativa das Métricas

Tabela 5 – Comparativo das métricas CER e WER nos modos Relaxed e Strict

Vídeo	CER (Relaxed)	WER (Relaxed)	CER (Strict)	WER (Strict)
1	0,0142	0,0825	0,0239	0,1441
2	0,0068	0,0347	0,0065	0,0347
3	0,0028	0,0129	0,0368	0,1717
4	0,0108	0,0505	0,0120	0,0606
5	0,0097	0,0487	0,0112	0,0602
6	0,0088	0,0400	0,0162	0,0800
7	0,0051	0,0240	0,0049	0,0240
8	0,0186	0,1026	0,0254	0,1436
9	0,0000	0,0000	0,0050	0,0260
10	0,0069	0,0641	0,0090	0,0769
11	0,1018	0,2064	0,1136	0,2530
12	0,0444	0,0810	0,0501	0,1171
13	0,0591	0,0965	0,0661	0,1379
14	0,0000	0,0000	0,0000	0,0000
15	0,0377	0,1133	0,0353	0,1133
16	0,0112	0,0450	0,0152	0,0720
17	0,0042	0,0387	0,0039	0,0387
18	0,0000	0,0000	0,0022	0,0054
19	0,0764	0,2800	0,0983	0,3661
20	0,0027	0,0126	0,0100	0,0253
Média	**0,0211 (2,1%)**	**0,0667 (6,7%)**	**0,0273 (2,7%)**	**0,0976 (9,8%)**

Fonte: O autor.

A comparação direta entre os resultados brutos e os resultados finais evidencia de forma clara o impacto positivo das técnicas de pós-processamento implementadas. As heurísticas de normalização ortográfica e reconstrução textual minimizaram erros comuns em legendas contendo estilos variados ou símbolos recorrentes, e corrigiram automaticamente padrões previsíveis de falhas, como hífens e apóstrofos mal posicionados e concatenação de palavras. Como consequência, o WER caiu de quase 30% para menos de 7% no modo *relaxed* e o CER foi reduzido a menos de metade do valor original, demonstrando a efetividade do módulo criado na melhoria da qualidade textual final.

6.3.3 Tempo de Execução

Além da análise da precisão das legendas extraídas, também foi avaliado o tempo de execução do software para cada um dos vídeos utilizados na etapa de testes, levando em conta execução com GPU (GTX 1080). Esse levantamento permite compreender o custo computacional da solução proposta, bem como a viabilidade prática de sua aplicação em diferentes cenários. Todos os experimentos foram executados no mesmo ambiente computacional, garantindo consistência na comparação entre os tempos medidos.

A tabela 6 apresenta a duração original de cada vídeo e o tempo total requerido pelo software para processá-lo. Observa-se que, embora vídeos curtos sejam processados rapidamente, vídeos com legendas mais densas ou com maior quantidade de *frames* relevantes tendem a demandar mais tempo, especialmente quando há múltiplas ocorrências de texto por segundo. A média dos tempos registrados demonstra que o sistema levou, em média, 296,9 s para processar vídeos com duração média de 85,3 s.

Tabela 6 – Comparativo entre duração dos vídeos e tempo de execução do programa

Vídeo	**Duração (s)**	**Tempo de Execução (s)**
1	54	90
2	107	103
3	204	189
4	46	78
5	120	134
6	76	540
7	74	517
8	100	87
9	151	101
10	100	49
11	104	527
12	77	556
13	75	545
14	60	83
15	67	502.8
16	60	64.5
17	69	526
18	92	117
19	180	171
20	100	104
Média	**85,3 s**	**296,9 s**

Fonte: O autor.

6.3.4 Interpretação dos Resultados

A avaliação de sistemas de OCR deve considerar o contexto e o objetivo da aplicação, uma vez que diferentes cenários impõem desafios específicos à leitura textual. Enquanto sistemas voltados a documentos digitalizados operam em ambientes controlados e com imagens de alta nitidez, aplicações como a do presente trabalho enfrentam variações constantes de iluminação, cor, contraste, tipografia, movimento e sobreposição de elementos visuais. Dessa forma, a acurácia de um OCR não deve ser analisada de forma absoluta, mas sim em relação à complexidade do domínio em que é aplicado.

De acordo com a definição apresentada pela plataforma aiOla, sistemas OCR de alta qualidade apresentam normalmente um CER inferior a 5%, sendo que valores entre 5% e 10%

ainda são considerados adequados para aplicações práticas, enquanto taxas superiores a 10% indicam necessidade de ajustes ou retrabalho no pipeline. No contexto deste trabalho, o sistema proposto alcançou um **CER de 2,1%** e **WER de 6,7%** no modo *relaxed*, e **CER de 2,7%**, **WER de 9,8%** no modo *strict* de avaliação. Esses resultados situam-se, portanto, na faixa satisfatória segundo os parâmetros de referência, demonstrando que o sistema atinge precisão excelente para o CER e valores aceitáveis para o WER, tendo em vista as complexidades visuais no contexto da aplicação (aiOla, 2024). A tabela 5 evidencia essa comparação:

Tabela 7 – Comparação entre faixas de desempenho de OCR (aiOla) e resultados obtidos no sistema proposto

Categoria / Métrica	**Faixa de Referência (aiOla)**	**Resultado Obtido**
Excellent (CER)	1–2%	**2,1% (Relaxed) | 2,7% (Strict)**
Very Strong (CER)	3–5%	—
Acceptable (CER)	5–10%	—
Needs Optimization (CER)	>10%	—
WER (informativo)	—	**6,7% (Relaxed) | 9,8% (Strict)**

Fonte: Adaptado de aiOla (2024) e resultados do autor.

Quando comparado a abordagens de ponta como o modelo EVE (End-to-End Video Subtitle Extraction) de Huang et al. (2025), observa-se que o presente sistema, embora mais leve e totalmente offline, apresenta desempenho notavelmente competitivo. O EVE utiliza a métrica Normalized Edit Distance (NED), que mede a similaridade entre o texto predito e o texto de referência, variando de 0 (sem correspondência) a 1 (correspondência perfeita). Essa métrica é conceitualmente próxima ao CER e ao WER, porém difere por considerar a sequência de caracteres de forma global, como visto no estudo de métricas de avaliação desse trabalho.

Conforme reportado no estudo de trabalhos relacionados, os resultados do EVE levam em consideração dois modelos, o EVE-0.5B e o EVE-7B, compostos por três módulos principais: um codificador de visão, um módulo adaptador e um modelo de linguagem de grande porte. Para o primeiro modelo, considerando vídeos do tipo *shorts* e filmes, os valores de NED foram de 75,9% e 68,5%, respectivamente. Já para o EVE-7B, os resultados alcançaram 88,2% e 80,8%.

Em comparação, o sistema desenvolvido neste trabalho obteve uma média de **NED = 99,15%**, ao aplicar esta métrica nos vídeos de avaliação utilizados nos resultados do presente projeto. Assim, evidencia-se um desempenho competitivo em termos numéricos, especialmente ao se considerar que o sistema é inteiramente local e não depende de redes neurais de larga escala ou de serviços de IA em nuvem. Cabe ressaltar, contudo, que as bases de vídeos utilizadas diferem significativamente entre os experimentos, o que naturalmente impacta as métricas obtidas. Ainda assim, o elevado índice de similaridade alcançado reforça a robustez e a eficácia da abordagem proposta.

Esses resultados demonstram que, mesmo com recursos computacionais modestos e arquitetura simplificada, o sistema apresenta desempenho comparável a soluções de última ge-

ração, confirmando seu potencial para aplicações em pesquisa, preservação e análise de acervos audiovisuais.

7 CONCLUSÃO

7.1 CONSIDERAÇÕES FINAIS

O presente trabalho teve como objetivo o desenvolvimento de um sistema capaz de extrair legendas embutidas (*burned-in subtitles*) de vídeos, empregando técnicas de Machine Learning (ML), Processamento de Imagem (PDI) e Reconhecimento Óptico de Caracteres (OCR). A proposta surgiu da necessidade de criar uma ferramenta automatizada que permitisse recuperar legendas visuais em formato editável, possibilitando aplicações em tradução, sumarização de conteúdo e indexação de acervos audiovisuais.

Ao longo do projeto, foi implementado um pipeline que integra etapas de detecção e reconhecimento textual, pós-processamento e filtragem espacial, resultando em uma solução funcional e eficiente. A avaliação foi conduzida sobre um conjunto de 20 vídeos em inglês com diferentes estilos de legenda e condições visuais. As métricas Character Error Rate (CER) e Word Error Rate (WER) indicaram resultados satisfatórios dentro do contexto em que foi analisado, especialmente no modo *relaxed*, demonstrando que o sistema cumpre seus objetivos e é capaz de extrair texto com boa precisão mesmo em cenários desafiadores.

Além disso, o sistema desenvolvido opera de forma totalmente offline, apoiado exclusivamente em modelos pré-treinados disponíveis em bibliotecas abertas. Essa característica garante maior privacidade, reprodutibilidade e independência de serviços externos, tornando a solução adequada para diferentes ambientes de uso.

Por fim, destaca-se que a extração de legendas embutidas possui aplicações relevantes em diversos contextos, como processos de tradução e localização, nos quais produtores, pesquisadores ou instituições podem recuperar legendas de vídeos para adaptação linguística, e também em estudos acadêmicos e análise de mídia, que frequentemente demandam o texto das falas para investigações sobre coisas como narrativa, estilo ou discurso, por exemplo. Essas possibilidades reforçam o potencial do sistema como ferramenta de apoio em tarefas de processamento, pesquisa e análise de conteúdo audiovisual.

7.2 TRABALHOS FUTUROS

Apesar dos resultados alcançados, há espaço para aprimoramentos que podem elevar ainda mais a robustez e a precisão do sistema. Entre as possibilidades de trabalhos futuros, destacam-se:

- **Aprimoramento do pós-processamento:** Incluir técnicas de reconstrução de palavras incorretas, além do tratamento já existente para separação de textos concatenados. Isso teria como objetivo reduzir o impacto de erros comuns em legendas com espaçamento irregular ou com sobreposição de caracteres.

- **Detecção de idioma e tradução automática:** Integrar um módulo de detecção automática de idioma e tradução assistida, permitindo gerar legendas traduzidas diretamente a partir do texto extraído, ampliando as aplicações do sistema para contextos multilíngues.
- **Ajuste fino da DBNet:** Realizar um *fine-tuning* da rede DBNet com um conjunto de dados específico de legendas em vídeos, de modo a melhorar a detecção textual sem depender exclusivamente da inferência genérica do modelo pré-treinado.
- **Integração com IAs multimodais:** Explorar a integração do sistema com IAs multimodais que vem revolucionando tarefas como o OCR. Essa melhoria teria como objetivo reduzir o índice de erro do programa e aprimorar a consistência semântica das legendas extraídas.

Essas melhorias visam aumentar a precisão global do sistema, reduzir a taxa de erro e ampliar o escopo de aplicação da ferramenta em diferentes domínios audiovisuais, consolidando sua utilidade prática e científica.

REFERÊNCIAS

aiOla. **Character Error Rate (CER) – Definition and Usage in OCR Accuracy**. 2024. https://aiola.ai/glossary/character-error-rate/. Acessado em 01 nov. 2025.

ALMEIDA, J. J. H.; LOPES, P. B.; AKAMINE, C. ATSC 3.0 Constellation Analysis using Computer Vision Combined with AI Decision Tree. In: **2023 IEEE International Conference on Consumer Electronics (ICCE)**. Las Vegas, NV, USA: IEEE, 2023. Disponível em: https://ieeexplore.ieee.org/document/10034169.

APARíCIO, M. et al. **Summarization of Films and Documentaries Based on Subtitles and Scripts**. 2015. https://arxiv.org/pdf/1506.01273. Acesso em: 14 de Março de 2025.

ARUN, M. et al. Lung cancer detection using resnet-50 cnn architecture. In: **2024 Second International Conference on Advances in Information Technology (ICAIT)**. IEEE, 2024. p. 1–7. Disponível em: https://ieeexplore.ieee.org/document/10690499.

BARRIS, B. **pyspellchecker: Pure Python Spell Checking Module**. 2024. Acessado em 27 set. 2025. Disponível em: https://pypi.org/project/pyspellchecker/.

BENCHAMARDIMATH, B.; HEGADI, R. S. A study on the importance of image processing and its applications. **International Journal of Research in Engineering and Technology (IJRET)**, v. 3, n. 1, p. 1–7, 2014. Acesso em: 18 de abril de 2025. Disponível em: https://www.researchgate.net/publication/340827009.

BRISINELLO, M. et al. Optical character recognition on images with colorful background. In: **2018 IEEE 8th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)**. IEEE, 2018. p. 1–6. Acesso em: 05 de Maio de 2025. Disponível em: https://ieeexplore.ieee.org/document/8576202.

CONTRIBUTORS, P. **PaddleOCR: An Open-Source Optical Character Recognition Toolkit**. 2025. Acessado em 08 out. 2025. Disponível em: https://www.paddleocr.ai/main/en/index.html.

EIKVIL, L. **Optical Character Recognition**. [S.l.], 1993. Disponível em: https://www.nr.no/ eikvil/OCR.pdf.

ELSHAHABY, H.; RASHWAN, M. An end-to-end system for subtitle text extraction from movie videos. **Journal of Ambient Intelligence and Humanized Computing**, v. 13, p. 1853–1865, 2022. Acesso em: 27 de Março de 2025. Disponível em: https://link.springer.com/article/10.1007/s12652-021-02951-1.

GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing**. 4. ed. Boston: Pearson, 2018. 43–45 p.

GOUD, D. S. et al. Text localization and recognition from natural scene images using ai. In: **2022 International Conference on Automation, Computing and Renewable Systems (ICACRS)**. IEEE, 2022. p. 1153–1158. Acesso em: 08 de Maio de 2025. Disponível em: https://ieeexplore.ieee.org/document/10029220.

GUPTA, A. Current research opportunities for image processing and computer vision. **Library of Science**, p. 1–4, 2019. Acesso em: 14 de Março de 2025. Disponível em: https://bibliotekanauki.pl/articles/305319.pdf.

HAYKIN, S. **Neural Networks and Learning Machines**. 3. ed. New York, NY, USA: Pearson Education, 2009. Acesso em: 23 de junho de 2025. Disponível em: https://lps.ufrj.br/ caloba/Livros/Haykin2009.pdf.

HE, K. et al. Deep residual learning for image recognition. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2016. p. 770–778.

HUANG, Y. et al. Eve: Towards end-to-end video subtitle extraction with vision-language models. **arXiv preprint arXiv:2503.04058**, 2025. Disponível em: https://arxiv.org/html/2503.04058v1.

JAYALAKSHMI, M. et al. Sign language to text conversion using random forest. In: **Proceedings of the 6th International Conference on Mobile Computing and Sustainable Informatics (ICMCSI-2025)**. Anand Nagar, Tamilnadu, India: IEEE, 2025. ISBN 979-8-3315-2266-7. Acesso em: 15 de Maio de 2025. Disponível em: https://ieeexplore.ieee.org/document/10883074.

KHAN, A. A.; LAGHARI, A. A.; AWAN, S. A. Machine learning in computer vision: A review. **ResearchGate Preprint**, p. 1–4, 2021. Acesso em: 14 de Março de 2025. Disponível em: https://www.researchgate.net/publication/351603970_Machine_Learning_in_Computer_Vision_A_Review.

KHATTAR, S.; SHEENAM. Enhancement computer vision using deep learning optimization techniques. In: **Proceedings of the Second International Conference on Automation, Computing and Renewable Systems (ICACRS-2023)**. Mohali, Punjab, India: IEEE, 2023. ISBN 979-8-3503-4023-5. Disponível em: https://ieeexplore.ieee.org/document/10405055.

KORTING, T. S. **Um paradigma para re-segmentação de imagens de alta resolução**. Dissertação (Dissertação de Mestrado) — Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, SP, Brasil, dezembro 2007. Disponível em: http://urlib.net/sid.inpe.br/mtc-m17@80/2008/02.12.12.07.

KUMAR, S. et al. Improve ocr accuracy with advanced image preprocessing using machine learning with python. **Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)**, 2020. Acesso em: 14 de Março de 2025. Disponível em: https://www.academia.edu/download/91232941/G5745059720.pdf.

LIAO, M. et al. Real-time scene text detection with differentiable binarization. In: AAAI. **Proceedings of the AAAI Conference on Artificial Intelligence**. [S.l.], 2020. v. 34, n. 07, p. 11474–11481.

LIU, J. et al. The character recognition method based on ocr. In: **Proceedings of the 26th ACIS International Winter Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD-Winter)**. Taiyuan, China: IEEE, 2023. p. 92–95. Acesso em: 18 de Abril de 2025. Disponível em: https://ieeexplore.ieee.org/document/10223979.

MAHADEVKAR, S. V. et al. A review on machine learning styles in computer vision—techniques and future directions. **IEEE Access**, IEEE, v. 10, p. 101518–101545, 2022. Disponível em: https://doi.org/10.1109/ACCESS.2022.3209825.

MANOCHA, S. K.; TEWARI, P. Comparative study of deep learning models for devanagari ocr. In: IEEE. **2021 International Conference on Smart Generation Computing, Communication and Networking (SMART GENCON)**. [S.l.], 2021. p. 1–6.

MISHRA, V.; KAUR, D. Sequence-to-sequence learning using deep learning for optical character recognition (ocr). In: **2018 International Conference on Computational Science and Computational Intelligence (CSCI)**. IEEE, 2018. p. 324–329. Acesso em: 08 de maio de 2025. Disponível em: https://doi.org/10.1109/CSCI46756.2018.00069.

MORRIS, K. K. et al. Closed captioning matters: Examining the value of closed captions for all students. **Journal of Postsecondary Education and Disability**, v. 29, n. 3, p. 231–240, 2016. Acesso em: 06 de abril de 2025. Disponível em: https://www.ahead-archive.org/uploads/publications/JPED/jped_29_3/JPED%2029_3_Final%20Doc.pdf.

NIMMA, D. et al. Revolutionizing image analysis: The integration of mlp and gru architectures in advanced computer vision. In: IEEE. **2024 1st International Conference on Green Industrial Electronics and Sustainable Technologies (GIEST)**. [S.l.], 2024. p. 1–8.

POLLION. **Global Accessibility: The Role of Subtitles and Captions in Media**. 2022. https://pollion.net/subtitles-and-captions. Acesso em: 06 de abril de 2025.

RAJ, S.; GUPTA, Y.; MALHOTRA, R. License plate recognition system using yolov5 and cnn. In: **2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS)**. IEEE, 2022. p. 1–6. Acesso em: 10 de Abril de 2025. Disponível em: https://ieeexplore.ieee.org/document/9785014.

RATHORE, D. K.; MANNEPALLI, P. K. A review of machine learning techniques and applications for health care. In: **Proceedings of the International Conference on Advances in Technology, Management & Education (ICATME-2021)**. IEEE, 2021. ISBN 978-1-7281-8586-6. Acesso em: 1 de Maio de 2025. Disponível em: https://ieeexplore.ieee.org/document/9732761.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 3. ed. New Jersey, NJ, USA: Prentice Hall, 2009. Acesso em: 23 de junho de 2025. Disponível em: https://people.engr.tamu.edu/guni/csce625/slides/AI.pdf.

SHARMA, R.; KAUSHIK, B.; GONDHI, N. Character recognition using machine learning and deep learning - a survey. In: **2020 International Conference on Emerging Smart Computing and Informatics (ESCI)**. [S.l.]: IEEE, 2020. p. 341–345.

SILVA, A. G. **Ambiente de suporte ao ensino de processamento de imagens usando a linguagem Python**. Dissertação (Dissertação de Mestrado) — Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação, Campinas, SP, Brasil, 2003. Orientador: Prof. Dr. Roberto de Alencar Lotufo.

SINGH, A.; BACCHUWAR, K.; BHASIN, A. A survey of ocr applications. **International Journal of Machine Learning and Computing**, IJMLC, v. 2, n. 3, p. 314–318, 2012. Disponível em: https://www.researchgate.net/publication/269395249.

SINGH, S.; SEHGAL, V. K. A comprehensive study: Image forensic analysis traditional to cognitive image processing. In: **2022 8th International Conference on Signal Processing and Communication (ICSC)**. IEEE, 2022. p. 236–243. Acesso em: 20 de maio de 2025. Disponível em: https://ieeexplore.ieee.org/document/10009322.

SINGKHORNART, T.; SURINTA, O. Multi-language video subtitle recognition with convolutional neural network and long short-term memory networks. **ICIC Express Letters**, v. 16, n. 6, p. 647–655, jun 2022. Acesso em: 27 de Março de 2025. Disponível em: https://www.researchgate.net/publication/360259761.

SMITH, R. An overview of the tesseract ocr engine. In: **Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR)**. [S.l.]: IEEE, 2007. p. 629–633.

SRIVASTAVA, S.; VERMA, A.; SHARMA, S. Optical character recognition techniques: A review. In: **2022 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS)**. [S.l.: s.n.], 2022.

TIMMINATOR. **wordninja-enhanced: Probabilistic word segmentation in Python**. 2024. Acessado em 27 set. 2025. Disponível em: https://github.com/timminator/wordninja-enhanced.

VARELA, M. E. Finetuning vision-language models as ocr systems for low-resource languages: A case study of manchu. **arXiv preprint arXiv:2404.XXXX**, 2024. Acesso em: 26 de agosto de 2025. Disponível em: https://arxiv.org/pdf/2507.06761.

VARGAS, A. C. G.; PAES, A.; VASCONCELOS, C. N. Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres. In: **Anais do Congresso Brasileiro de Computação**. Niterói, Brasil: [s.n.], 2016. Acesso em: 7 de Maio de 2025. Disponível em: http://sibgrapi.sid.inpe.br/col/sid.inpe.br/sibgrapi/2016/09.12.15.44/doc/um-estudo-sobre.pdf.

VERBOVSKIY, A. **Comparing OCR and VLM Techniques in Processing Tabular Data**. Dissertação (Master's Thesis) — Faculty of Information Technology and Electrical Engineering, University of Oulu, June 2025.

VICENZI, F.

Identificação de propriedades de produtos a partir da leitura automática de rótulos — Universidade Federal de Santa Catarina, Araranguá, Brasil, 2021. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação). Acesso em: 03 de Abril de 2025. Disponível em: https://repositorio.ufsc.br/handle/123456789/226045.

WANG, S.; CAO, J. Ai and deep learning for urban computing. In: SHI, W. et al. (Ed.). **Urban Informatics**. Singapore: Springer, 2021, (The Urban Book Series). p. 815–844. Disponível em: https://link.springer.com/chapter/10.1007/978-981-15-8983-6_43.

WANJARI, K.; VERMA, P. A review on the applications of machine learning and deep learning algorithms for image recognition. In: **Proceedings of the Fourth International Conference on Sentiment Analysis and Deep Learning (ICSADL-2025)**. IEEE, 2025. p. 1707–1711. Acesso em: 15 de Maio de 2025. Disponível em: https://ieeexplore.ieee.org/document/10933346.

WIBAWA, A. et al. Limb detection and classification using computer vision and machine learning. In: **2024 International Conference on Artificial Intelligence, Blockchain, Cloud Computing, and Data Analytics (ICoABCD)**. IEEE, 2024. Acesso em: 15 de Maio de 2025. Disponível em: https://ieeexplore.ieee.org/document/10704535.

Wikipédia. **OpenCV – Wikipédia, a enciclopédia livre**. 2024. Acesso em: 15 de Maio de 2025. Disponível em: https://pt.wikipedia.org/wiki/OpenCV.

ZHANG, C. C. et al. Unleashing the potential of machine learning: An exploration of state-of-the-art algorithms and real-world applications in computer vision. In: **2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE)**. IEEE, 2023. Acesso em: 23 de Abril de 2025. Disponível em: https://ieeexplore.ieee.org/document/10223979.

ZHANG, Y. et al. Consensus entropy: Harnessing multi-vlm agreement for self-verifying and self-improving ocr. **arXiv preprint arXiv:2403.XXXX**, 2024. Acesso em: 26 de agosto de 2025. Disponível em: https://arxiv.org/pdf/2504.11101.

APÊNDICE A – CÓDIGO-FONTE

O código fonte desenvolvido neste projeto está disponível no respositório do autor no GitHub, no link: https://github.com/JohnJohnNB/Burned-In-Subtitles-Extractor. Lá está o código-fonte na íntegra, assim como os arquivos e scripts utilizados para computar as métricas de avaliação. Para fins de completude do documento, este apêndice apresentará o código-fonte da aplicação desenvolvida.

```
1
2  import argparse
3  import cv2
4  import datetime
5  import numpy as np
6  import wordninja_enhanced
7  import re
8  import unicodedata
9  import Levenshtein as lev
10 from spellchecker import SpellChecker
11 from paddleocr import PaddleOCR
12
13
14 class SubtitleExtractor:
15     # Static variables
16     WHITESPACE = " "
17     CALIBRATION_FRAMES = 600  # Number of frames to use for subtitle position
       ↪  calibration
18     TEXT_LENGTH_THRESHOLD = 20  # Minimum length to consider a text as subtitle
19     POSITION_TOLERANCE = 50    # Pixels tolerance to consider a text position as
       ↪  a subtitle one
20
21     TOKENIZER = re.compile(r"[A-Za-zÀ-ÿ0-9']+|[.,!?;:\"“”‘’]")
22
23     def __init__(self, video_path: str, language: str, target_fps: int = 10):
24         self.video_path = video_path
25         self.language = language
26         self.target_fps = target_fps
27
28         # Instantiates PaddleOCR
29         self.ocr = PaddleOCR(
30             use_doc_orientation_classify=False,
31             use_doc_unwarping=False,
32             use_textline_orientation=False
```

```
        )

        # SpellCheckers - Checks for existent/non-existent words in language dictionary
        self.spell_en = SpellChecker(language="en")
        self.spell_pt = SpellChecker(language="pt")

        # WordNinja - Splits concatenated words
        self.wordninja_lm = wordninja_enhanced.LanguageModel(language=language)

    # ----------------------
    # Auxiliary functions
    # ----------------------

    # Function to retrieve all tokens within a subtitle
    def simple_tokenize(self, text):
        return self.TOKENIZER.findall(text)

    # Function to fix special character errors on subtitles (fixes wrong spacing on "!?,.", etc.)
    def fix_special_cases(self, text):
        text = re.sub(r"\b(\w+)\s+'(\w+)", r"\1'\2", text)
        text = re.sub(r"\b(\w)\s+'(\w+)", r"\1'\2", text)
        text = re.sub(r"\s+([,!.?;:])", r"\1", text)
        return text

    # Main function for OCR post-processing (fix concatenated words, special characters wrong spacing, etc.)
    def split_concatenated_words(self, text):
        spell = self.spell_en if self.language == "en" else self.spell_pt
        use_wordninja = self.language in ["en", "pt"]

        tokens = self.simple_tokenize(text)
        corrected = []

        for token in tokens:
            # Classifies as word if made of letters + special chars (apostrophes, hyphen)
            if re.match(r"^[A-Za-zÀ-ÿ0-9'-]+$", token):
                if token.lower() in spell:
                    corrected.append(token)
                else:
```

```
                    if use_wordninja:
                        split_tokens = self.wordninja_lm.split(token)
                        if len(split_tokens) > 1:
                            # Inserts split words
                            corrected.extend(split_tokens)
                        else:
                            corrected.append(token)
                    else:
                        corrected.append(token)
            else:
                corrected.append(token)

        return self.fix_special_cases(" ".join(corrected))

    # Function to convert timestamp to srt format
    def format_time_srt(self, seconds):
        td = datetime.timedelta(seconds=seconds)
        total_milliseconds = int(td.total_seconds() * 1000)
        hours = total_milliseconds // 3600000
        minutes = (total_milliseconds % 3600000) // 60000
        seconds = (total_milliseconds % 60000) // 1000
        milliseconds = total_milliseconds % 1000
        return f"{hours:02d}:{minutes:02d}:{seconds:02d},{milliseconds:03d}"

    # Function to normalize text (for text similarity check)
    def normalize_text(self, text):
        text = unicodedata.normalize("NFKD", text)
        text = "".join([c for c in text if not unicodedata.combining(c)])
        text = text.lower()
        text = re.sub(r"[^\w\s]", "", text)
        return re.sub(r"\s+", " ", text).strip()

    # Function to check for jaccard similarity (for text similarity check)
    def jaccard_similarity(self, a, b):
        set_a = set(self.normalize_text(a).split())
        set_b = set(self.normalize_text(b).split())
        return len(set_a & set_b) / max(1, len(set_a | set_b))

    # Main function to check for text similarity (to check if subtitles are the
    ↪  same)
    def check_if_same_text(self, current_text, last_text, lev_threshold=0.85,
    ↪  jaccard_threshold=0.75):
```

```
        # Normalizes text
        current_norm = self.normalize_text(current_text)
        last_norm = self.normalize_text(last_text)

        # If identical after check, returns as same text
        if current_norm == last_norm:
            return True

        # Levenshtein ratio
        lev_ratio = lev.ratio(current_norm, last_norm)
        if lev_ratio >= lev_threshold:
            return True

        # Jaccard similarity as fallback
        jaccard = self.jaccard_similarity(current_text, last_text)
        if jaccard >= jaccard_threshold:
            return True

        return False

    # Function to check if subtitle is mostly made of legitimate words and not noise
    def is_text_legible(self, text, threshold=0.8):
        normalized_text = self.normalize_text(text)

        words = [w for w in re.findall(r"[A-Za-zÀ-ÿ']+", normalized_text)]
        if not words:
            return False

        valid = 0
        for word in words:
            # check = w.replace("'", "").replace("-", "")
            if len(word) >= 2 and word.lower() in self.spell_en:
                valid += 1

        ratio = valid / len(words)
        return ratio >= threshold

    # Function to determine if a subtitle is noise text or legit text
    def is_noise(self, sub, min_duration=0.3):
        duration = sub['end_time'] - sub['start_time']
        legibility_threshold = 0.75
```

```
152
153         # If subtitle's duration is within normal range, skips noise test
154         if duration >= min_duration:
155             return False
156
157         if duration <= 0.25:
158             legibility_threshold = 0.95
159
160         # If subtitle's duration is within abnormal range, proceeds with noise
            ↪  test
161         return not self.is_text_legible(sub['text'],
            ↪  threshold=legibility_threshold)
162
163     # ----------------------
164     # Main functions
165     # ----------------------
166
167     # Function to determine subtitle average position
168     def calibrate_subtitle_position(self, video, num_frames=CALIBRATION_FRAMES):
169         total_frames = int(video.get(cv2.CAP_PROP_FRAME_COUNT))
170         quarter_frame = total_frames // 4
171
172         # Defines starting point to calculate
173         start_frame = quarter_frame
174         end_frame = min(quarter_frame + num_frames, total_frames)
175
176         # Defines video starting point for calibration
177         video.set(cv2.CAP_PROP_POS_FRAMES, start_frame)
178
179         y_positions = []
180         frame_count = start_frame
181
182         while frame_count < end_frame:
183             ret, frame = video.read()
184
185             if not ret:
186                 break
187
188             # Runs OCR
189             result = self.ocr.predict(frame)
190             detected_texts = result[0]['rec_texts']
191             detected_bounding_boxes = result[0]['rec_polys']
```

```
192
193             # Iterates over detected texts from OCR and gets subtitles
                ↪  coordinates
194             if detected_texts:
195                 for i, text in enumerate(detected_texts):
196                     if len(text) > self.TEXT_LENGTH_THRESHOLD:
197                         points = np.array(detected_bounding_boxes[i],
                            ↪  dtype=np.int32)
198                         x, y, w, h = cv2.boundingRect(points)
199                         y_positions.append(y)
200
201             frame_count += 1
202
203         # Resets video to start
204         video.set(cv2.CAP_PROP_POS_FRAMES, 0)
205
206         if y_positions:
207             avg_y = int(np.mean(y_positions))
208             std_y = int(np.std(y_positions))
209             return avg_y, std_y  # Returns average and standard deviation
210
211         return None, None
212
213     # Function that uses PaddleOCR Text Detection and Recognition modules for
214     # detecting and recognizing text from image. Additionally, uses wordsegment
215     # NLP library for post-processing final output (for english subtitles)
216     def extract_text_from_frame(self, image, avg_y_pos, std_y_pos=None, k=2.0):
217         if image is None:
218             return ""
219
220         # Runs OCR engine to detect and recognize text from an image
221         result = self.ocr.predict(image)
222         detected_texts = result[0]['rec_texts']
223         detected_bounding_boxes = result[0]['rec_polys']
224
225         if not detected_texts:
226             return ""
227
228         # Defines tolerance to determine if text position is within subtitle
            ↪  average position
229         if std_y_pos is not None and std_y_pos > 0:
230             tolerance = k * std_y_pos
```

```
231         else:
232             tolerance = self.POSITION_TOLERANCE  # fallback para valor fixo
233 
234         # Filter texts based on bounding box position (or not if calibration
            ↪ failed)
235         filtered_texts = []
236 
237         if avg_y_pos == float('inf'):
238             filtered_texts = detected_texts
239         else:
240             for i, text in enumerate(detected_texts):
241                 points = np.array(detected_bounding_boxes[i], dtype=np.int32)
242                 x, y, w, h = cv2.boundingRect(points)
243 
244                 if abs(y - avg_y_pos) <= tolerance:
245                     filtered_texts.append(detected_texts[i])
246 
247         if not filtered_texts:
248             return ""
249 
250         enhanced_texts_list = [
251             self.split_concatenated_words(text.strip())
252             for text in filtered_texts
253         ]
254 
255         return self.WHITESPACE.join(enhanced_texts_list)
256 
257     # Main function with whole pipeline
258     def run(self):
259         # Loads video
260         video = cv2.VideoCapture(self.video_path)
261         wordninja_lm = wordninja_enhanced.LanguageModel(language=self.language)
262 
263         if not video.isOpened():
264             print(f"ERROR: Couldn't load video on {self.video_path}")
265             exit()
266 
267         # Calculates subtitle average position to prevent false positives
268         print("1. Calculating average subtitles position")
269         avg_y, std_y = self.calibrate_subtitle_position(video)
270 
271         if avg_y is None:
```

```
272             print("ERROR: Subtitle position average position calculus failed. No
                ↪  subtitles detected.")
273             # If calibration fails, disables subtitle position feature
274             avg_y, std_y = -1, None # Value to disable subtitle position feature
275         else:
276             print(f"SUCCESS: Calculating average subtitles position succeeded:
                ↪  {avg_y} pixels | Standard Deviation: {std_y}")
277 
278         # Calculates ideal FPS for better performance
279         fps = video.get(cv2.CAP_PROP_FPS)
280         total_frames = int(video.get(cv2.CAP_PROP_FRAME_COUNT))
281         duration = total_frames / fps
282         print(f"Real FPS: {fps}, Frames: {total_frames}, Duration:
                ↪  {duration:.2f}s")
283 
284         # Variables required for main loop
285         subtitle_data = []
286         last_subtitle_text = ""
287         frame_interval = int(fps / self.target_fps) if self.target_fps < fps
                ↪  else 1
288         frame_index = 0
289         start_frame = 0
290 
291         # Start subtitle extraction main process
292         print("2. Extracting subtitles")
293 
294         while True:
295             video.set(cv2.CAP_PROP_POS_FRAMES, frame_index)
296             ret, frame = video.read()
297             if not ret:
298                 break
299 
300             # Extract texts and verifies if its in subtitles average position
301             if avg_y != -1:
302                 current_subtitle_text = self.extract_text_from_frame(frame,
                    ↪  avg_y, std_y, 3.0)
303             else:
304                 # In case subtitle position calibration fails, extract text from
                    ↪  any position
305                 current_subtitle_text = self.extract_text_from_frame(frame,
                    ↪  float('inf'))
306 
```

```
            # Checks if subtitles are the same between different frames
            is_same_text = False

            if last_subtitle_text != "":
                is_same_text = self.check_if_same_text(current_subtitle_text,
                ↪  last_subtitle_text)

            # Logic to handle subtitle start and end
            if current_subtitle_text != last_subtitle_text and not is_same_text:
                if not last_subtitle_text:
                    # A new subtitle has appeared
                    start_frame = frame_index
                else:
                    # The text has changed, save the previous subtitle
                    end_frame = frame_index - 1
                    candidate = {
                        'text': last_subtitle_text,
                        'start_time': self.format_time_srt(start_frame / fps),
                        'end_time': self.format_time_srt(end_frame / fps),
                    }

                    # Noise subtitle test
                    if not self.is_noise({
                        'text': candidate['text'],
                        'start_time': start_frame / fps,
                        'end_time': end_frame / fps,
                    }):
                        subtitle_data.append(candidate)
                        print(last_subtitle_text)

                    # Start a new subtitle
                    start_frame = frame_index

            # Increments frame index variable and updates last subtitle
            frame_index += frame_interval
            last_subtitle_text = current_subtitle_text

        # Saves the last subtitle after the loop ends
        if last_subtitle_text:
            end_frame = frame_index - 1
            candidate = {
                'text': last_subtitle_text,
```

```
                'start_time': self.format_time_srt(start_frame / fps),
                'end_time': self.format_time_srt(end_frame / fps),
            }

            if not self.is_noise({
                'text': candidate['text'],
                'start_time': start_frame / fps,
                'end_time': end_frame / fps,
            }):
                subtitle_data.append(candidate)

        # Releases video object
        video.release()

        # Saves subtitles to .srt file
        with open("subtitles.srt", "w", encoding="utf-8") as srt_file:
            for i, sub in enumerate(subtitle_data):
                # Number sequence
                srt_file.write(f"{i + 1}\n")
                # Start / End - Timestamp
                srt_file.write(f"{sub['start_time']} --> {sub['end_time']}\n")
                # Subtitle text
                srt_file.write(f"{sub['text']}\n\n")

        print("\nSubtitles succesfully saved on subtitles.srt!")


def parse_args():
    parser = argparse.ArgumentParser(description="Burned-in Subtitles
    ↪ Extractor")
    parser.add_argument("video_path", type=str, help="Video path to be
    ↪ extracted")
    parser.add_argument("language", type=str, choices=["en", "pt"],
    ↪ help="Language (en or pt)")
    parser.add_argument("target_fps", type=int, nargs="?", default=10,
    ↪ help="Target FPS (default=10)")
    return parser.parse_args()


if __name__ == "__main__":
    args = parse_args()
```

```
385     extractor = SubtitleExtractor(args.video_path, args.language,
        ↪  args.target_fps)
386     extractor.run()
387
```

APÊNDICE B – ARTIGO SBC

Neste apêndice, o presente projeto será apresentado como artigo no formato SBC.

Implementação de um software para extração automática de legendas embutidas em vídeos utilizando OCR e Processamento de Imagem

João Victor Nery B. de Oliveira[1], **Alexandre Gonçalves Silva**[2]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)

Abstract. *Image Processing, Optical Character Recognition (OCR), and Machine Learning (ML) play a key role in automating the extraction and interpretation of visual information. This work presents the development of a software for extracting burned-in subtitles from videos and converting them into readable, synchronizable text in* `.srt` *format. The system integrates text detection and recognition techniques, operating fully offline using pre-trained models. Tests with various video styles and contrast levels achieved average error rates of 2.1% (CER) and 6.7% (WER) in relaxed mode, demonstrating the efficiency of the proposed approach. The project contributes to applications in accessibility, translation, and audiovisual content analysis.*

Resumo. *O Processamento de Imagem (PDI), o Reconhecimento Óptico de Caracteres (OCR) e Machine Learning (ML) têm desempenhado papel essencial na extração e interpretação automática de informações visuais. Este trabalho apresenta o desenvolvimento de um software para extração de legendas embutidas (burned-in subtitles) em vídeos, convertendo-as em texto legível e sincronizável em formato* `.srt`*. O sistema integra técnicas de detecção e reconhecimento de texto, operando de forma totalmente offline a partir de modelos pré-treinados. Foram realizados testes com vídeos de diferentes estilos e contrastes, alcançando taxas médias de erro de 2,1% (CER) e 6,7% (WER) no modo de avaliação relaxado (relaxed), o que demonstra a eficácia da solução proposta. O projeto contribui para aplicações em acessibilidade, tradução e análise de conteúdo audiovisual.*

1. Introdução

O avanço das tecnologias de Processamento Digital de Imagens (PDI), Reconhecimento Óptico de Caracteres (OCR) e Machine Learning (ML) tem proporcionado soluções inovadoras para transformar informações visuais em dados processáveis, facilitando a vida humana em diversos contextos. Contudo, existe um desafio na extração de legendas embutidas em vídeos. Diferentemente das legendas convencionais, essas legendas, denominadas *burned-in subtitles*, são fixadas diretamente no conteúdo visual, tornando a separação do texto e a sua utilização em outros contextos um processo complexo e pouco explorado.

A extração de legendas embutidas possui utilidade em diferentes contextos. Entre eles, destacam-se: a tradução de conteúdos, permitindo que materiais originalmente produzidos em um idioma cheguem a públicos mais amplos; o arquivamento de texto relacionado a vídeos, possibilitando a indexação textual para busca eficiente por termos;

e a sumarização de conteúdos, fornecendo resumos automáticos de vídeos. Além disso, esses pontos colaboram para a acessibilidade, pois viabilizam a manipulação e conversão de legendas para diversos públicos, como pessoas surdas.

De forma mais abrangente, as técnicas empregadas neste trabalho podem ser estendidas para outras aplicações práticas. Por exemplo, em sistemas de navegação assistida para pessoas cegas, a extração de informações visuais pode auxiliar no deslocamento seguro ao identificar placas de sinalização ou informações contextuais em ruas e avenidas. Em veículos autônomos, mecanismos semelhantes permitem reconhecer sinais como placas de "STOP" ou instruções de trânsito essenciais para a condução segura.

Como mencionado anteriormente, a necessidade deste trabalho surge também das limitações enfrentadas por pessoas que dependem de recursos como tradução e audiodescrição para compreender conteúdos audiovisuais. Como aponta [Pollion 2022], indivíduos surdos ou com perda auditiva recente relataram sentir-se mais incluídos e conectados à sociedade ao assistirem conteúdos com legendas *closed caption*, que oferecem flexibilidade ao serem facilmente removíveis ou editáveis. No entanto, a realidade das legendas embutidas dificulta a criação de versões acessíveis ou adaptadas, pois não podem ser facilmente removidas ou manipuladas. Além disso, [Morris et al. 2016] destacam que o uso de *closed captions* pode melhorar a compreensão de conteúdos audiovisuais para estudantes, promovendo inclusão e acessibilidade.

Isso demonstra a relevância de ferramentas que automatizam a extração de legendas embutidas, facilitando a tradução e a preservação de conteúdos. Em vista disso, [Aparício et al. 2015] ressaltam que a extração de informações a partir de legendas e roteiros pode gerar resumos automáticos de qualidade, cuja avaliação pode ser realizada por meio de métricas como ROUGE, amplamente utilizadas por sua forte correlação com julgamentos humanos.

Nesse cenário, é notável a importância de desenvolver um software capaz de identificar e capturar automaticamente essas legendas. Desse modo, o PDI desempenha um papel crucial em diversas aplicações, abrangendo desde diagnósticos médicos até monitoramento ambiental e reconhecimento automático. Segundo [Gupta 2019], a evolução contínua dessa área se deve a avanços em técnicas de imagem, aumento do poder computacional e crescente necessidade de aplicações inovadoras. Algumas técnicas conhecidas são a binarização adaptativa, muito utilizada para destacar caracteres em fundos complexos, a transformada de Hough, empregada na detecção de linhas e formas geométricas, e os filtros de convolução, que possibilitam a detecção de bordas e redução de ruídos.

Além disso, a área de ML tem sido fundamental para a detecção de objetos em imagens, permitindo uma análise visual mais precisa. Segundo [Khan et al. 2021], técnicas supervisionadas e não supervisionadas são amplamente utilizadas para esse fim. O aprendizado não supervisionado permite a extração de características por meio de técnicas como segmentação por histogramas e *clustering*, a exemplo do algoritmo Fuzzy C-means. Já o aprendizado supervisionado utiliza dados rotulados para melhorar a precisão na identificação de objetos, a exemplo de técnicas como Support Vector Machines (SVM) e Convolutional Neural Networks (CNNs), tornando possível a implementação de sistemas robustos para análise visual em áreas como o reconhecimento de padrões.

Por último, a tecnologia denominada OCR é amplamente utilizada para a con-

versão eletrônica de texto impresso ou digitado em texto legível por máquina. Segundo [Kumar et al. 2020], o processo de OCR envolve três etapas principais: digitalização do documento, identificação do texto e armazenamento do resultado em um formato editável e pesquisável. Esse método é amplamente aplicado para a digitalização de documentos oficiais, recibos, e-mails empresariais e extratos bancários, tornando a informação mais acessível e estruturada. Dessa forma, espera-se que, unindo as tecnologias citadas juntamente ao processamento de OCR, seja possível extrair legendas embutidas com alta precisão e convertê-las em um formato legível, como a extensão .srt, amplamente utilizada para sincronização de legendas, contribuindo para acessibilidade, preservação e tradução de conteúdos.

2. Fundamentação Teórica

2.1. Processamento Digital de Imagens

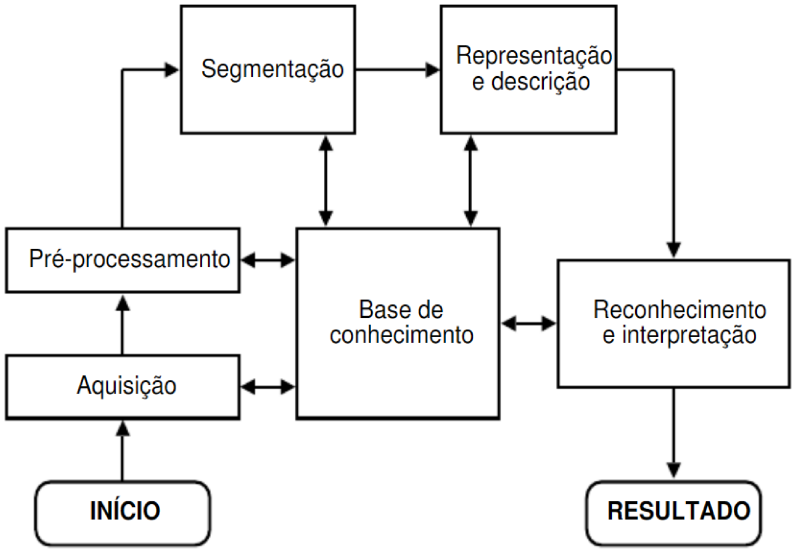
O Processamento Digital de Imagens (PDI) é o campo da ciência da computação dedicado à análise e manipulação de imagens digitais, visando a extração de informações relevantes ou a melhoria da percepção visual. Suas aplicações abrangem áreas como visão computacional, reconhecimento facial, realidade aumentada e reconhecimento óptico de caracteres (OCR) [Benchamardimath and Hegadi 2014].

Uma imagem digital pode ser entendida como uma função bidimensional $f(x, y)$, onde cada valor corresponde à intensidade luminosa de um pixel. Em imagens de 8 bits, esses valores variam entre 0 e 255, e no caso de imagens coloridas, cada pixel é composto por três canais: Red, Green e Blue (RGB) [Singh and Sehgal 2022].

As técnicas de PDI permitem manipular essas matrizes para melhorar a visualização ou permitir que sistemas computacionais interpretem o conteúdo. Operações como limiarização, suavização, detecção de bordas e análise morfológica são amplamente utilizadas para destacar regiões de interesse e reduzir ruídos visuais [Gonzalez and Woods 2018].

Conforme [Gonzalez and Woods 2018], o fluxo típico do PDI envolve etapas como aquisição, pré-processamento, melhoria, restauração, segmentação e reconhecimento de padrões. Em sistemas de OCR, por exemplo, o reconhecimento de padrões é responsável por identificar caracteres após a segmentação da imagem. Essas etapas são ajustadas conforme o contexto e guiadas por uma base de conhecimento que orienta a escolha de técnicas e parâmetros.

Figure 1. Etapas fundamentais do PDI [Korting 2007]



2.2. Machine Learning

O Machine Learning (ML) é um subcampo da Inteligência Artificial que permite que sistemas aprendam a partir de dados, identifiquem padrões e tomem decisões com mínima intervenção humana [Rathore and Mannepalli 2021]. Sua integração com a visão computacional impulsionou grandes avanços em tarefas como detecção de objetos, reconhecimento facial e OCR [Zhang et al. 2023].

Os métodos de ML podem ser classificados em aprendizado supervisionado e não supervisionado. No primeiro, o modelo é treinado com dados rotulados para realizar tarefas de classificação e regressão; no segundo, busca-se descobrir padrões e estruturas em dados não rotulados [Mahadevkar et al. 2022].

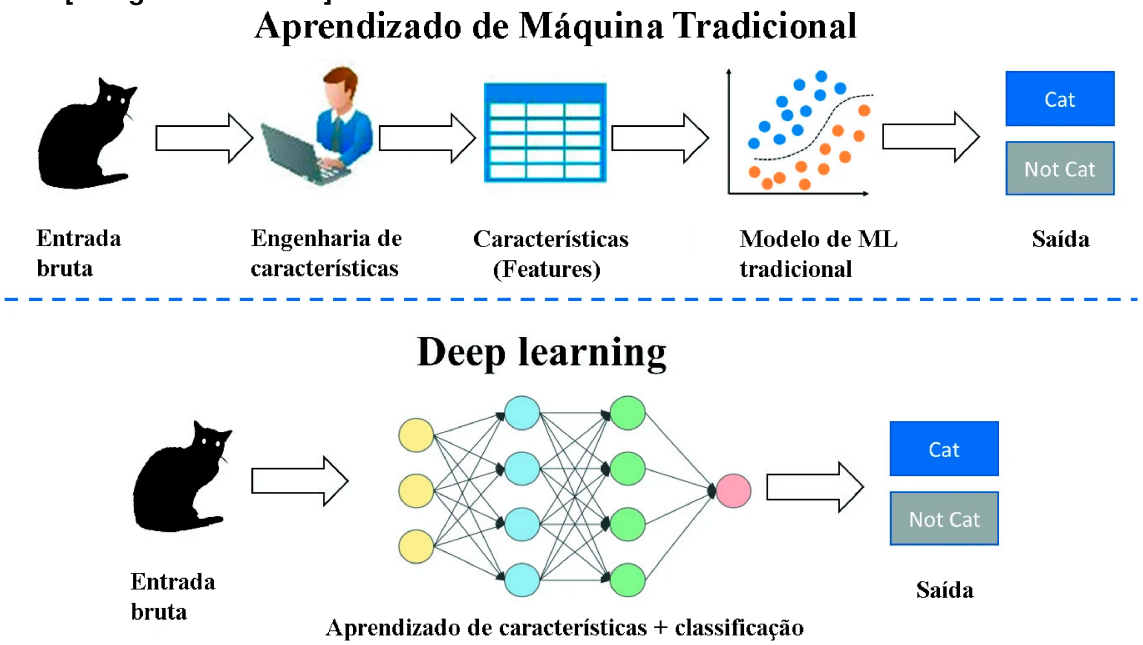
Antes do avanço das redes neurais profundas (Deep Learning), a visão computacional era amplamente dominada por técnicas clássicas de ML, ainda relevantes quando há restrições de dados ou recursos computacionais. Entre elas, destacam-se:

- **K-Nearest Neighbors (KNN):** classifica novos exemplos com base na similaridade com os "k" vizinhos mais próximos no espaço de atributos [Zhang et al. 2023].
- **Support Vector Machines (SVM):** encontra um hiperplano ótimo que separa classes no espaço de *features*, utilizando funções *kernel* para tratar dados não lineares [Sharma et al. 2020].
- **Árvores de Decisão e Random Forest:** constroem modelos hierárquicos baseados em regras de decisão e ganho de informação, sendo robustos e eficientes para classificação e análise de variáveis [Almeida et al. 2023, Wibawa et al. 2024].
- **K-Means:** método não supervisionado que agrupa dados em "k" clusters, comumente usado para segmentação de imagens e separação de regiões textuais [Brisinello et al. 2018].

Entre as abordagens inspiradas no funcionamento cerebral humano, destacam-se as Redes Neurais Artificiais (RNAs) e, em especial, o Perceptron Multicamadas (MLP),

capaz de aprender representações não lineares por meio do algoritmo de retropropagação [Haykin 2009]. Essas redes serviram de base para o surgimento das redes neurais profundas (Deep Learning), amplamente utilizadas em sistemas modernos de OCR. A figura 2 ilustra bem a diferença em relação ao processo de aprendizado clássico e moderno quanto à uma imagem, por exemplo.

Figure 2. Aprendizado de Máquina Tradicional vs Deep Learning, Adaptado de [Wang and Cao 2021]



2.2.1. Deep Learning

O Deep Learning (DL) é uma subárea do Machine Learning que utiliza redes neurais com múltiplas camadas para representar e aprender padrões complexos de dados. Essas arquiteturas são amplamente aplicadas em tarefas como reconhecimento de texto, classificação e visão computacional, e diferenciam-se das abordagens tradicionais pela capacidade de extração automática de características, reduzindo a necessidade de pré-processamentos manuais [Khattar and Sheenam 2023, Sharma et al. 2020]. Dessa forma, essa área do conhecimento trouxe avanços significativos em tarefas como classificação de imagens, segmentação semântica e detecção de objetos, por exemplo [Sharma et al. 2020].

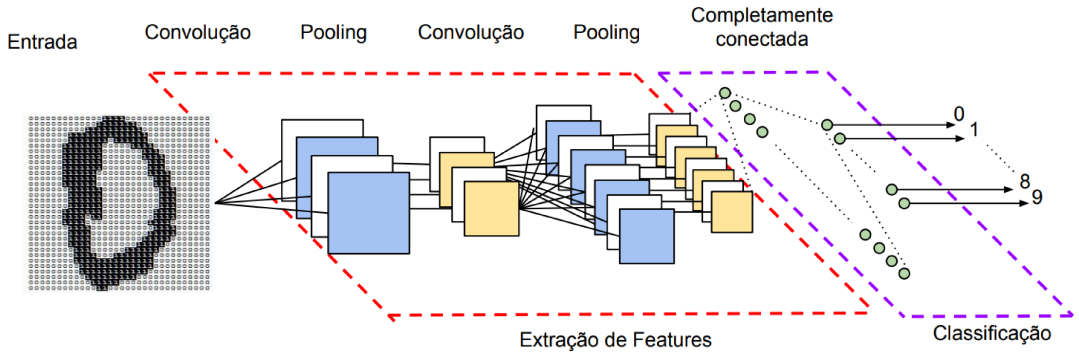
Nas aplicações de visão computacional, os modelos empregam blocos estruturais específicos como camadas convolucionais, operações de *pooling* e camadas de normalização, que exploram a natureza espacial dos dados de imagem e reduzem drasticamente o número de parâmetros aprendidos. Além disso, componentes como funções de perda e funções de ativação não lineares auxiliam algoritmos a extrair padrões de dados e produzir previsões para dados novos [Khattar and Sheenam 2023].

Entre as principais arquiteturas de Deep Learning utilizadas em visão computa-

cional, destacam-se as Redes Neurais Convolucionais (CNNs), as Redes Neurais Recorrentes (RNNs) e as arquiteturas híbridas CRNNs, além de modelos avançados como ResNet-50, DBNet e os emergentes Vision-Language Models (VLMs).

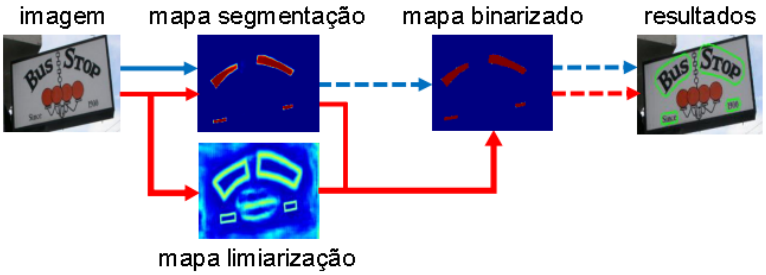
- **Convolutional Neural Networks (CNNs)**
  As Redes Neurais Convolucionais são amplamente utilizadas em visão computacional e reconhecimento de padrões visuais. Suas tarefas incluem detecção de objetos, classificação de imagens, reconhecimento de caracteres, processamento de linguagem natural, entre outras. Elas empregam camadas convolucionais, de *pooling* e totalmente conectadas para extrair automaticamente características hierárquicas de imagens, eliminando a necessidade de extração manual de *features* [Sharma et al. 2020]. Essa estrutura permite detectar bordas, texturas e formas complexas, sendo eficaz em tarefas como classificação e detecção de objetos. Modelos como LeNet, AlexNet e ResNet demonstram sua relevância para tarefas visuais em larga escala [Nimma et al. 2024].

Figure 3. Exemplo de uma CNN e suas diferentes camadas [Vargas et al. 2016]



Para detecção de texto em imagens, existem modelos como a DBNet (Differentiable Binarization Network), que é uma arquitetura de detecção de texto baseada em segmentação projetada para lidar com a complexidade de detectar textos em cenas reais. Esse modelo incorpora o processo de binarização diretamente à rede, permitindo aprendizado adaptativo de limiares durante o treinamento. Com isso, arquiteturas como essa mostram-se promissoras para tarefas como a detecção de legendas embutidas, onde a velocidade e a precisão são igualmente relevantes. Dessa forma, essa abordagem reduz o custo de pós-processamento e atinge equilíbrio entre precisão e velocidade, tornando-a ideal para aplicações de OCR em cenários reais [Liao et al. 2020].

Figure 4. Pipeline tradicional vs Pipeline DBNet. As linhas azuis representam o pipeline tradicional e as vermelhas o do DBNet; As setas pontilhadas representam os operadores de inferência e as setas preenchidas representam operadores diferenciáveis tanto no treinamento quanto na inferência. Adaptado de [Liao et al. 2020]



- **Recurrent Neural Networks (RNN) e Convolutional Recurrent Neural Networks (CRNN)**
  Em tarefas de reconhecimento de texto sequencial, as RNNs são amplamente utilizadas por sua capacidade de modelar dependências temporais. Quando combinadas com CNNs, formam as Convolutional Recurrent Neural Networks (CRNNs), que integram a extração visual de características (*features*) com o aprendizado sequencial. Treinadas com a função de perda Connectionist Temporal Classification (CTC), as CRNNs permitem o aprendizado *end-to-end* sem necessidade de alinhamento explícito entre caracteres e imagens [Goud et al. 2022].

- **Vision-Language Models (VLMs)**
  Os Vision-Language Models (VLMs) representam uma nova geração de modelos multimodais que integram visão computacional e processamento de linguagem natural. Baseados em arquiteturas transformers, eles são capazes de compreender e gerar informações visuais e textuais de forma conjunta. Estudos recentes mostram que VLMs refinados para OCR atingem resultados competitivos em métricas como CER e WER, evidenciando seu potencial como evolução natural para sistemas integrados de reconhecimento de texto [Zhang et al. 2024, Escobar Varela 2024].

2.3. Reconhecimento Óptico de Caracteres (OCR)

O Optical Character Recognition (OCR) é uma tecnologia de visão computacional voltada à conversão de texto presente em imagens, documentos digitalizados e vídeos em texto pesquisável e processável por máquina [Singh et al. 2012]. Essa tecnologia é amplamente empregada em áreas como análise documental, automação de registros médicos e leitura automática de placas veiculares (*Automatic Number Plate Recognition* – ANPR) [Raj et al. 2022].

2.3.1. Sistemas Tradicionais de OCR

Sistemas tradicionais de OCR seguem um pipeline composto pelas etapas de aquisição, pré-processamento, segmentação, extração de *features* (características) e classificação

[Srivastava et al. 2022]. Inicialmente, a imagem é refinada por técnicas como binarização e remoção de ruído, por exemplo. Em seguida, ocorre a segmentação e extração de *features*, utilizando transformadas de Fourier ou Wavelet. Por fim, algoritmos de classificação como KNN, SVM ou redes neurais associam os vetores extraídos aos caracteres correspondentes. Ferramentas clássicas como o Tesseract seguem essa abordagem em versões mais antigas [Smith 2007].

2.3.2. OCR Baseado em Deep Learning

Com o avanço do Deep Learning, o OCR passou a adotar redes neurais profundas capazes de realizar extração automática de *features*, superando limitações dos métodos tradicionais [Manocha and Tewari 2021]. As CNNs se destacam pela robustez em reconhecer caracteres sob ruído, distorções e variações de fonte, com arquiteturas como LeNet, AlexNet e ResNet amplamente exploradas [Sharma et al. 2020].

Modelos recentes integram arquiteturas especializadas, como a combinação de DBNet, CRNN e a função de perda CTC, proposta por [Liu et al. 2023]. Nesse pipeline, a DBNet realiza a detecção das regiões textuais, enquanto a CRNN, composta por camadas convolucionais e recorrentes, executa o reconhecimento sequencial de caracteres. A função CTC, por sua vez, permite o treinamento *end-to-end*, dispensando a necessidade de alinhamento explícito entre entrada e saída. Essa combinação tornou-se o estado da arte em OCR moderno, sendo amplamente utilizada em aplicações práticas e acadêmicas.

3. Desenvolvimento

De forma geral, o objetivo do software proposto é extrair legendas embutidas em vídeos (*burned-in subtitles*) e gerar um arquivo `.srt` com as legendas devidamente sincronizadas com seus *timestamps*. Para isso, serão empregadas tecnologias que envolvem redes neurais pré-treinadas para detecção e reconhecimento de texto (OCR), além de técnicas para pós-processamento textual. A figura 5 exemplifica, de forma simplificada, como será o processamento de um *frame* de um vídeo com legenda embutida.

Figure 5. Exemplo de funcionamento do software. As imagens mostram, em sequência: 1. Entrada de um *frame* de um vídeo no software; 2. Processo de detecção e reconhecimento do texto no *frame*; 3. Extração e sincronização da legenda extraída, levando em conta o processamento de todos os *frames* necessários para tal. Fonte: O autor.



3.1. Análise de Requisitos

O sistema foi projetado para oferecer uma ferramenta prática e modular de extração de legendas, com ênfase na precisão, portabilidade e facilidade de uso. Seu objetivo principal é processar vídeos e gerar arquivos `.srt` contendo os blocos de legendas devidamente sincronizados com seus intervalos temporais.

Os requisitos funcionais envolvem a capacidade de processar vídeos, aplicar OCR e extrair legendas temporizadas para um arquivo `.srt`, com possibilidade de customização da execução por meio de uma Interface de Linha de Comando (CLI). Por outro lado, os requisitos não funcionais enfatizam desempenho e reprodutibilidade (Para permitir execução tanto em ambientes que executam o software com CPU ou GPU). A linguagem Python foi escolhida devido ao seu ecossistema maduro em visão computacional.

3.2. Tecnologias Utilizadas

O sistema foi desenvolvido em Python 3.12, utilizando as seguintes bibliotecas principais:

- **PaddleOCR**: Responsável pela detecção e reconhecimento de texto, utilizando modelos como DBNet e CRNN;
- **OpenCV**: Empregado para leitura de *frames* dos vídeos;
- **wordninja-enhanced** e **pyspellchecker**: Utilizados para pós-processamento textual;
- **python-Levenshtein**: Utilizado para cálculo de similaridade textual e métricas de avaliação.

Essas ferramentas foram integradas em um pipeline que combina detecção espacial e filtragem linguística para aprimorar a precisão das legendas extraídas.

3.3. Arquitetura do Sistema

O projeto adota uma arquitetura modular, composta por um núcleo central e módulos auxiliares. A classe principal, denominada `SubtitleExtractor`, coordena todo o fluxo de execução, integrando os componentes de OCR, detecção e pós-processamento textual. A estrutura de diretórios foi mantida simples, contendo o script principal (`main.py`), vídeos e legendas de teste (`/videos`) e um módulo de avaliação (`evaluation.py`).

Essa organização visa garantir clareza, reprodutibilidade e facilidade de manutenção, permitindo ajustes ou substituição de módulos sem impacto estrutural.

3.4. Pipeline de Extração de Legendas

O pipeline implementado segue três estágios principais: **calibração espacial das legendas**, **extração textual** e **reconstrução das legendas**.

1. **Calibração Espacial das Legendas:** Antes do processamento principal, o sistema analisa um conjunto de *frames* para estimar automaticamente a região média onde as legendas costumam aparecer. Essa etapa reduz falsos positivos e otimiza o tempo de execução.
2. **Extração Textual:** Cada *frame* é processado pelo programa utilizando o `OpenCV`, onde o `PaddleOCR`, em seguida, identifica e reconhece blocos de texto. As detecções são filtradas de acordo com a posição média obtida na calibração, eliminando elementos irrelevantes como textos no cenário do vídeo. Em seguida, aplica-se o pós-processamento textual com `pyspellchecker` e `wordninja-enhanced`, que envolve a segmentação de palavras concatenadas, problema recorrente de aplicações de OCR.
3. **Reconstrução e Geração do Arquivo `.srt`:** Os textos reconhecidos são agrupados temporalmente, considerando a continuidade e similaridade entre *frames* consecutivos. Para cada bloco de legenda consolidado, são atribuídos intervalos temporais (*timestamps*) no formato `HH:MM:SS,mmm`, resultando no arquivo final.

3.5. Heurísticas e Estratégias de Filtragem

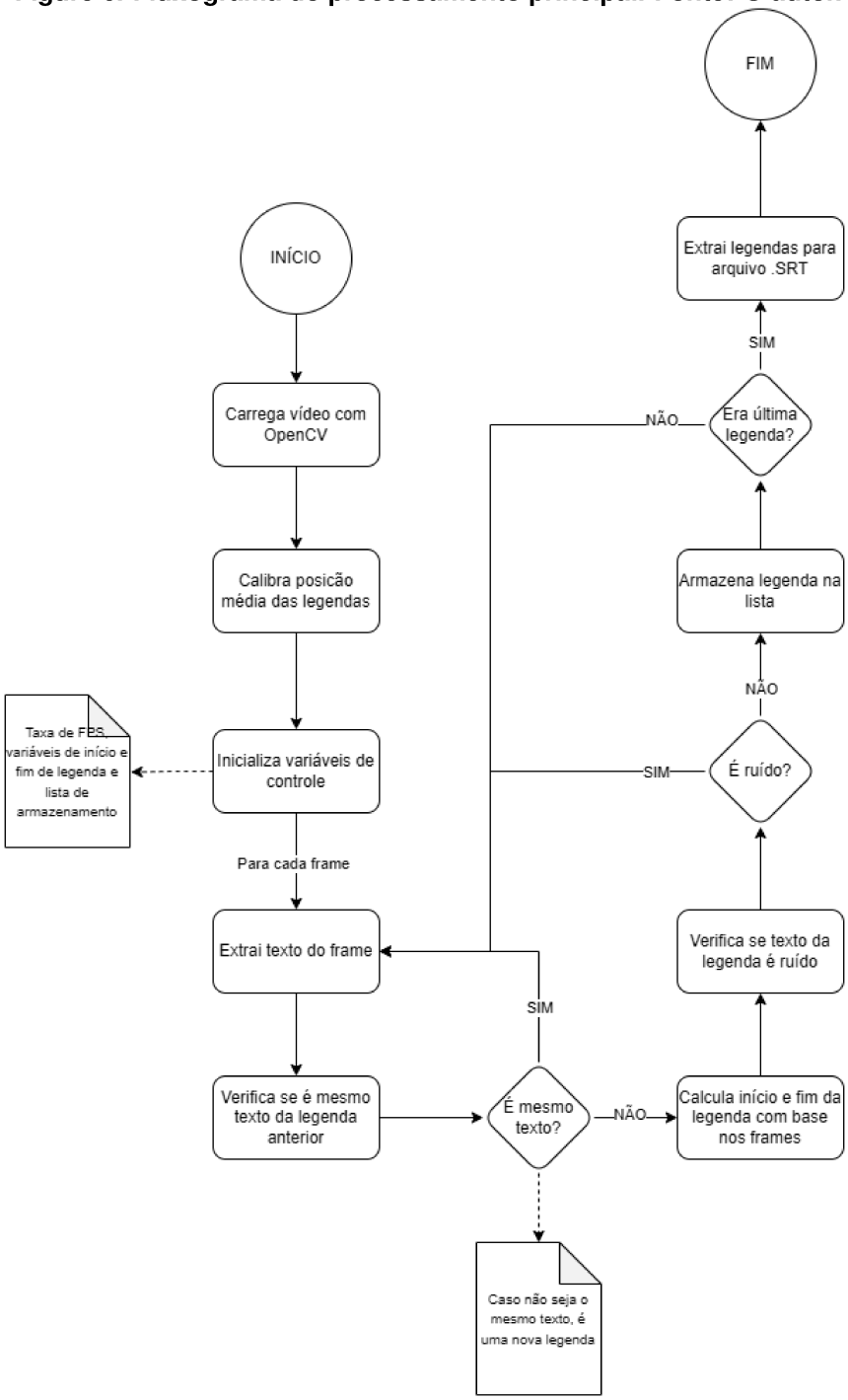
Durante o desenvolvimento, foram aplicadas diversas heurísticas para aprimorar a precisão do sistema:

- **Filtragem espacial:** Apenas textos dentro da região média de legendas são processados;

- **Filtragem temporal e linguística:** Um bloco de texto extraído no processamento principal passa por algumas validações antes de ser considerado uma legenda legítima, sendo: Blocos de texto com duração muito curtas (menor que 0.3s) são descartadas e consideradas ruído. Caso contrário, o texto em questão é validado com base em proporções mínimas de palavras reconhecidas no dicionário da língua em questão, utilizando-se o `pyspellchecker`;
- **Medição de similaridade:** Textos de *frames* consecutivos são comparados com base em métricas de Levenshtein e Jaccard, evitando duplicações e transições incorretas.

Essas heurísticas foram definidas empiricamente com base em testes sobre diferentes vídeos, mostrando-se eficazes para equilibrar desempenho e precisão. Sabendo-se disso, o fluxograma geral do pipeline é ilustrado na figura 6.

Figure 6. Fluxograma do processamento principal. Fonte: O autor.



4. Resultados

Os resultados obtidos com a avaliação do sistema proposto foi executado sobre um conjunto de 20 vídeos em língua inglesa, contendo diferentes resoluções, contrastes e estilos de legenda. A avaliação concentrou-se na mensuração da precisão do OCR por meio das métricas Character Error Rate (CER) e Word Error Rate (WER), amplamente empregadas para quantificar o desempenho de sistemas de reconhecimento de texto.

4.1. Métricas CER e WER

As métricas CER e WER foram aplicadas sob dois modos de avaliação: *relaxed* e *strict*. No modo *relaxed*, pequenas variações de pontuação, espaçamento e capitalização são ignoradas, enquanto no modo *strict* qualquer divergência literal é contabilizada como erro. Essa distinção permite avaliar tanto a robustez prática do sistema quanto sua precisão textual literal.

Os resultados médios obtidos foram **CER = 0,0211 (2,1%)** e **WER = 0,0667 (6,7%)** no modo *relaxed*, e **CER = 0,0273 (2,7%)** e **WER = 0,0976 (9,8%)** no modo *strict*, indicando bom grau de fidelidade em ambas as métricas.

Table 1. Comparativo das métricas CER e WER nos modos Relaxed e Strict. Fonte: O autor.

Vídeo	CER (Relaxed)	WER (Relaxed)	CER (Strict)	WER (Strict)
1	0,0142	0,0825	0,0239	0,1441
2	0,0068	0,0347	0,0065	0,0347
3	0,0028	0,0129	0,0368	0,1717
4	0,0108	0,0505	0,0120	0,0606
5	0,0097	0,0487	0,0112	0,0602
6	0,0088	0,0400	0,0162	0,0800
7	0,0051	0,0240	0,0049	0,0240
8	0,0186	0,1026	0,0254	0,1436
9	0,0000	0,0000	0,0050	0,0260
10	0,0069	0,0641	0,0090	0,0769
11	0,1018	0,2064	0,1136	0,2530
12	0,0444	0,0810	0,0501	0,1171
13	0,0591	0,0965	0,0661	0,1379
14	0,0000	0,0000	0,0000	0,0000
15	0,0377	0,1133	0,0353	0,1133
16	0,0112	0,0450	0,0152	0,0720
17	0,0042	0,0387	0,0039	0,0387
18	0,0000	0,0000	0,0022	0,0054
19	0,0764	0,2800	0,0983	0,3661
20	0,0027	0,0126	0,0100	0,0253
Média	**0,0211 (2,1%)**	**0,0667 (6,7%)**	**0,0273 (2,7%)**	**0,0976 (9,8%)**

4.2. Interpretação e Discussão dos Resultados

Segundo a classificação proposta pela plataforma aiOla [aiOla 2024], sistemas OCR de alta qualidade apresentam valores de CER inferiores a 5%, sendo taxas entre 5% e 10% consideradas aceitáveis. Assim, os resultados obtidos situam-se na faixa de **desempenho excelente** para o CER e **adequado** para o WER, considerando o contexto de desafios visuais diversos presentes na tarefa de extração de legendas embutidas.

A Tabela 2 resume a comparação entre os intervalos de referência e os resultados obtidos:

Table 2. Comparação entre faixas de desempenho de OCR e resultados obtidos

Categoria (aiOla)	Faixa CER	Resultado Obtido
Excellent	1–2%	2,1% (Relaxed) / 2,7% (Strict)
Acceptable	5–10%	—
Needs Optimization	>10%	—
WER (informativo)	—	6,7% (Relaxed) / 9,8% (Strict)

O modelo EVE (End-to-End Video Subtitle Extraction) [Huang et al. 2025], com objetivo semelhante ao deste trabalho, avaliou a precisão de extração de legendas utilizando a métrica Normalized Edit Distance (NED). Os resultados reportados indicam que o modelo EVE-0.5B alcançou, para vídeos do tipo *shorts* e filmes, valores de NED de 75,9% e 68,5%, respectivamente. Já para o EVE-7B, os resultados alcançaram 88,2% e 80,8%. Em comparação, o sistema proposto neste trabalho apresentou uma média de **NED = 99,15%**, evidenciando desempenho competitivo, mesmo sem recorrer a modelos de larga escala ou processamento em nuvem.

Cabe destacar que as bases de vídeos utilizadas diferem significativamente entre os experimentos, o que influencia diretamente os resultados. Ainda assim, o elevado índice de similaridade obtido reforça a robustez e a eficácia da abordagem desenvolvida, demonstrando que soluções locais e leves podem atingir resultados comparáveis a sistemas de última geração.

4.3. Limitações Observadas

Apesar dos resultados positivos, algumas limitações foram identificadas:

- **Interferências visuais:** Textos presentes no cenário (placas, letreiros) dos vídeos que não sejam legendas, caso posicionados na região de ocorrência média, podem ser confundidos como legendas legítimas, gerando falsos positivos;
- **Baixo contraste:** Legendas que contrastam com o fundo do vídeo reduzem a acurácia do OCR e podem causar duplicação de blocos de legenda legítimos na transcrição do `.srt` final;

Ainda assim, tais limitações não comprometeram a compreensão das legendas legítimas extraídas, e representam oportunidades de aprimoramento futuro do pipeline de extração.

5. Conclusão

O presente trabalho teve como objetivo o desenvolvimento de um sistema para extração automática de legendas embutidas (*burned-in subtitles*) de vídeos, utilizando técnicas de Machine Learning (ML), Processamento Digital de Imagens (PDI) e Reconhecimento Óptico de Caracteres (OCR). A proposta surgiu da necessidade de criar uma ferramenta automatizada capaz de converter legendas visuais em texto editável, viabilizando aplicações em tradução, sumarização e indexação de acervos audiovisuais.

O sistema implementado integra etapas de detecção e reconhecimento textual, pós-processamento textual e filtragem espacial, resultando em uma solução funcional e eficiente. A avaliação, conduzida sobre um conjunto de 20 vídeos com diferentes estilos e contrastes, demonstrou resultados satisfatórios. As métricas Character Error Rate (CER)

e Word Error Rate (WER) apresentaram baixos índices de erro, especialmente no modo *relaxed*, confirmando a precisão do método mesmo em contextos visuais complexos.

Uma característica relevante do sistema é sua independência de serviços de IA externos, operando de forma totalmente offline e utilizando apenas modelos pré-treinados para detecção e reconhecimento de texto. Essa abordagem garante privacidade, reprodutibilidade e viabilidade de uso em ambientes sem conexão à internet, além de reduzir custos e dependências externas.

De modo geral, os resultados obtidos comprovam a viabilidade técnica da abordagem proposta e reforçam o potencial do sistema como ferramenta de apoio em tarefas de processamento e análise de conteúdo audiovisual.

5.1. Trabalhos Futuros

Apesar dos resultados positivos, há espaço para aprimoramentos que podem elevar ainda mais a robustez e a precisão do sistema. Entre as principais possibilidades de trabalhos futuros, destacam-se:

- **Aprimoramento do pós-processamento:** Implementar técnicas de reconstrução de palavras incorretas e separação de textos sobrepostos, reduzindo o impacto de erros causados por espaçamento irregular ou baixa resolução.
- **Detecção de idioma e tradução automática:** Adicionar módulos de detecção automática de idioma e tradução assistida, permitindo gerar legendas traduzidas diretamente a partir do texto extraído, ampliando as aplicações em contextos multilíngues.
- **Ajuste fino da DBNet:** Realizar *fine-tuning* da rede DBNet com um conjunto de dados específico de legendas, de modo a aprimorar a detecção textual e reduzir falsos positivos.
- **Integração com IAs multimodais:** Explorar o uso de modelos de linguagem de grande porte (LLMs) com capacidades visuais, visando reduzir o índice de erro e aprimorar a consistência semântica das legendas extraídas.

Essas melhorias visam aumentar a precisão global do sistema, reduzir a taxa de erro e ampliar o escopo de aplicação da ferramenta em diferentes domínios audiovisuais, consolidando sua utilidade prática e científica.

References

aiOla (2024). Character error rate (cer) – definition and usage in ocr accuracy. `https://aiola.ai/glossary/character-error-rate/`. Acessado em 01 nov. 2025.

Almeida, J. J. H., Lopes, P. B., and Akamine, C. (2023). ATSC 3.0 Constellation Analysis using Computer Vision Combined with AI Decision Tree. In *2023 IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, NV, USA. IEEE.

Aparício, M., Figueiredo, P., Raposo, F., Martins de Matos, D., Ribeiro, R., and Marujo, L. (2015). Summarization of films and documentaries based on subtitles and scripts. `https://arxiv.org/pdf/1506.01273`. Acesso em: 14 de Março de 2025.

Benchamardimath, B. and Hegadi, R. S. (2014). A study on the importance of image processing and its applications. *International Journal of Research in Engineering and Technology (IJRET)*, 3(1):1–7. Acesso em: 18 de abril de 2025.

Brisinello, M., Grbić, R., Stefanović, D., and Pečkai-Kovač, R. (2018). Optical character recognition on images with colorful background. In *2018 IEEE 8th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, pages 1–6. IEEE. Acesso em: 05 de Maio de 2025.

Escobar Varela, M. (2024). Finetuning vision-language models as ocr systems for low-resource languages: A case study of manchu. *arXiv preprint arXiv:2404.XXXX*. Acesso em: 26 de agosto de 2025.

Gonzalez, R. C. and Woods, R. E. (2018). *Digital Image Processing*. Pearson, Boston, 4 edition.

Goud, D. S., M., V., P., A., G., V., Yadav, A. S., and Kumar, A. (2022). Text localization and recognition from natural scene images using ai. In *2022 International Conference on Automation, Computing and Renewable Systems (ICACRS)*, pages 1153–1158. IEEE. Acesso em: 08 de Maio de 2025.

Gupta, A. (2019). Current research opportunities for image processing and computer vision. *Library of Science*, pages 1–4. Acesso em: 14 de Março de 2025.

Haykin, S. (2009). *Neural Networks and Learning Machines*. Pearson Education, New York, NY, USA, 3 edition. Acesso em: 23 de junho de 2025.

Huang, Y., Liu, Y., Ge, Y., Huang, X., and Shan, Y. (2025). Eve: Towards end-to-end video subtitle extraction with vision-language models. *arXiv preprint arXiv:2503.04058*.

Khan, A. A., Laghari, A. A., and Awan, S. A. (2021). Machine learning in computer vision: A review. *ResearchGate Preprint*, pages 1–4. Acesso em: 14 de Março de 2025.

Khattar, S. and Sheenam (2023). Enhancement computer vision using deep learning optimization techniques. In *Proceedings of the Second International Conference on Automation, Computing and Renewable Systems (ICACRS-2023)*, Mohali, Punjab, India. IEEE.

Korting, T. S. (2007). Um paradigma para re-segmentação de imagens de alta resolução. Dissertação de mestrado, Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, SP, Brasil.

Kumar, S., Sharma, M., Handa, K., and Jaiswal, R. (2020). Improve ocr accuracy with advanced image preprocessing using machine learning with python. *Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)*. Acesso em: 14 de Março de 2025.

Liao, M., Wan, Z., Yao, C., Chen, K., and Bai, X. (2020). Real-time scene text detection with differentiable binarization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 11474–11481. AAAI.

Liu, J., Bao, J., Li, B., Feng, Z., and Zhang, S. (2023). The character recognition method based on ocr. In *Proceedings of the 26th ACIS International Winter Conference on Soft-*

ware Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD-Winter), pages 92–95, Taiyuan, China. IEEE. Acesso em: 18 de Abril de 2025.

Mahadevkar, S. V., Khemani, B., Patil, S., Kotecha, K., Vora, D. R., Abraham, A., and Gabralla, L. A. (2022). A review on machine learning styles in computer vision—techniques and future directions. *IEEE Access*, 10:101518–101545.

Manocha, S. K. and Tewari, P. (2021). Comparative study of deep learning models for devanagari ocr. In *2021 International Conference on Smart Generation Computing, Communication and Networking (SMART GENCON)*, pages 1–6. IEEE.

Morris, K. K., Frechette, C., Dukes III, L., Stowell, N., Topping, N. E., and Brodosi, D. (2016). Closed captioning matters: Examining the value of closed captions for all students. *Journal of Postsecondary Education and Disability*, 29(3):231–240. Acesso em: 06 de abril de 2025.

Nimma, D., Sengar, A. S., Kaur, C., Khan, S. A., Anushna, S., and Infant Raj, I. (2024). Revolutionizing image analysis: The integration of mlp and gru architectures in advanced computer vision. In *2024 1st International Conference on Green Industrial Electronics and Sustainable Technologies (GIEST)*, pages 1–8. IEEE.

Pollion (2022). Global accessibility: The role of subtitles and captions in media. `https://pollion.net/subtitles-and-captions`. Acesso em: 06 de abril de 2025.

Raj, S., Gupta, Y., and Malhotra, R. (2022). License plate recognition system using yolov5 and cnn. In *2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 1–6. IEEE. Acesso em: 10 de Abril de 2025.

Rathore, D. K. and Mannepalli, P. K. (2021). A review of machine learning techniques and applications for health care. In *Proceedings of the International Conference on Advances in Technology, Management & Education (ICATME-2021)*. IEEE. Acesso em: 1 de Maio de 2025.

Sharma, R., Kaushik, B., and Gondhi, N. (2020). Character recognition using machine learning and deep learning - a survey. In *2020 International Conference on Emerging Smart Computing and Informatics (ESCI)*, pages 341–345. IEEE.

Singh, A., Bacchuwar, K., and Bhasin, A. (2012). A survey of ocr applications. *International Journal of Machine Learning and Computing*, 2(3):314–318.

Singh, S. and Sehgal, V. K. (2022). A comprehensive study: Image forensic analysis traditional to cognitive image processing. In *2022 8th International Conference on Signal Processing and Communication (ICSC)*, pages 236–243. IEEE. Acesso em: 20 de maio de 2025.

Smith, R. (2007). An overview of the tesseract ocr engine. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR)*, pages 629–633. IEEE.

Srivastava, S., Verma, A., and Sharma, S. (2022). Optical character recognition techniques: A review. In *2022 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS)*.

Vargas, A. C. G., Paes, A., and Vasconcelos, C. N. (2016). Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres. In *Anais do Congresso Brasileiro de Computação*, Niterói, Brasil. Acesso em: 7 de Maio de 2025.

Wang, S. and Cao, J. (2021). Ai and deep learning for urban computing. In Shi, W., Goodchild, M. F., Batty, M., Kwan, M.-P., and Zhang, A., editors, *Urban Informatics*, The Urban Book Series, pages 815–844. Springer, Singapore.

Wibawa, A., Wijaya, N. M., Suhartono, D., and Lucky, H. (2024). Limb detection and classification using computer vision and machine learning. In *2024 International Conference on Artificial Intelligence, Blockchain, Cloud Computing, and Data Analytics (ICoABCD)*. IEEE. Acesso em: 15 de Maio de 2025.

Zhang, C. C., Zhang, K., Ni, R., Liu, H., and Shen, J. (2023). Unleashing the potential of machine learning: An exploration of state-of-the-art algorithms and real-world applications in computer vision. In *2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE)*. IEEE. Acesso em: 23 de Abril de 2025.

Zhang, Y., Liang, T., Huang, X., Cui, E., Guo, X., Chu, P., Li, C., Zhang, R., Wang, W., and Liu, G. (2024). Consensus entropy: Harnessing multi-vlm agreement for self-verifying and self-improving ocr. *arXiv preprint arXiv:2403.XXXX*. Acesso em: 26 de agosto de 2025.