UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Reconhecimento de caracteres em imagens de anúncios de produtos

Trabalho de Conclusão de Curso

Yasmim Batista Ferreira

São Cristóvão – Sergipe

2023

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Yasmim Batista Ferreira

Reconhecimento de caracteres em imagens de anúncios de produtos

Trabalho de Conclusão de Curso submetido ao Departamento de Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador(a): Leonardo Nogueira Matos
Coorientador(a): Thiago Dias Bispo e Rafael Andrade da Silva

São Cristóvão – Sergipe

2023

Resumo

A comparação de preço de produtos na hora das compras é uma atividade importante que faz parte do nosso cotidiano. Nos últimos anos essa tarefa tem sido facilitada com o surgimento de *softwares* que buscam trazer para seus usuários dados atualizados sobre ofertas e promoções em supermercados. Um dos *softwares* criados para essa finalidade é o *LudiiPrice*, aplicativo para dispositivos móveis que tem como principal função realizar a busca, monitoramento e comparação de preços de produtos de diversos estabelecimentos, tendo como principal fonte para sua base de dados, uma colaboração com seus usuários que cadastram notas fiscais de compras recentes através da leitura de *QR Code*. Para ampliar sua base de dados, o *LudiiPrice* busca aumentar sua fonte através da busca por ofertas e anúncios de folhetos de supermercados em páginas na *web* e realização da extração dos dados presentes nesses folhetos. Para realizar este feito, o presente trabalho tem como objetivo analisar técnicas e estratégias de detecção, reconhecimento e extração de caracteres em recortes de preços de produtos em folhetos de ofertas de supermercado, visando aprimorar a qualidade das imagens dos recortes para obter a maior precisão e confiabilidade possível na extração de dados.

Palavras-chave: Reconhecimento de caracteres, Detecção de caracteres, Processamento de imagens, Aprendizado profundo

Abstract

Comparing product prices when shopping is an important activity that is part of our daily lives and that in recent years has been facilitated with the emergence of software that seeks to bring its users up-to-date data on offers and promotions at supermarkets. One of the softwares created for this purpose is LudiiPrice, an application for mobile devices that has as main function to perform the search, monitoring and comparison of prices of products from several establishments having, as main source for its database, a collaboration with its users who register recent purchases invoices through QR Code reading. To expand its database, LudiiPrice seeks to increase its source by searching for offers and ads in supermarket flyers on web sites and performing the extraction of the data present in these flyers. To accomplish this goal, this work aims to analyze techniques and strategies for detection, recognition and extraction of characters in cutouts of product prices in supermarket flyers offers, seeking to improve the quality of the images of the cutouts to obtain the highest possible accuracy and reliability in data extraction.

Keywords: *Character recognition, Character detection, Image processing, Deep learning*

Lista de ilustrações

Lista de tabelas

Lista de códigos

Lista de abreviaturas e siglas

API	*Application Programming Interface*
BGR	*Blue Green Red*
BIMP	*Batch Image Manipulation*
CNN	*Convolutional Neural Network*
ConvNet	*Convolutional Network*
GIMP	*GNU Image Manipulation*
JPEG	*Joint Photographic Experts Group*
NumPy	*Numerical Python*
OCR	*Optical character recognition*
OpenCV	*Open Source Computer Vision*
PDF	*Portable Document Format*
PNG	*Portable Network Graphic*
PSM	*Page Segmentation Method*
QR	*Quick Response*
RGB	*Red Green Blue*
SSD	*Single Shot Detector*
SVM	*Support Vector Machine*
TCC	Trabalho de Conclusão de Curso
TIFF	*Tagged Image File Format*
UFS	Universidade Federal de Sergipe
YOLO	*You Only Look Once*

Sumário

1 Introdução

Encontrar o menor preço de um produto é uma atividade essencial e frequente para todos os consumidores que buscam economizar e manter uma boa saúde financeira. Para facilitar a vida desses consumidores, várias ferramentas, sistemas e plataformas foram criados para auxiliar na busca de produtos em supermercados, comparar preços entre eles e indicar ao consumidor qual produto tem o melhor custo benefício.

Um dos sistemas criados com esse intuito é o *LudiiPrice*, projeto desenvolvido na Universidade Federal de Sergipe (UFS) sob a coordenação dos orientadores deste trabalho. O sistema consiste em uma aplicação que permite realizar o monitoramento de produtos de diversos estabelecimentos, utilizando a comparação dos preços registrados em seu banco de dados. Para executar essa tarefa, os usuários fornecem dados sobre produtos através de leitura de um *QR Code* que é emitido na nota fiscal a cada compra que realiza em um estabelecimento. Como esse trabalho colaborativo depende do usuário para ter um campo amostral maior e se manter atualizado, o trabalho relacionado Barbosa (2022) integra ao sistema uma nova fonte de dados recolhendo as informações dos produtos de imagens de folhetos de supermercados divulgadas na Internet. Esta fonte permite ampliar consideravelmente a base de dados de produtos. Mas, para completar essa tarefa, é necessário realizar a extração do texto presente nos recortes dessas imagens, sendo o preço a informação principal para a finalidade geral do sistema seguida pela descrição dos produtos.

A detecção e reconhecimento de preços de produtos tanto de notas fiscais, quanto de folhetos de supermercados é um enorme desafio que pesquisadores e cientistas ainda buscam aperfeiçoar nas últimas três décadas. Durante a busca por uma solução para o problema em questão, várias abordagens foram consideradas. Entre elas, destacam-se o uso de ferramentas de reconhecimento óptico de caracteres (OCR), conforme analisado e executado em Ullah et al. (2018), e o uso de *deep learning* por meio de modelos de redes neurais convolucionais, como abordado por Mosquera e GenÇ (2019). Estas abordagens são as mais avançadas e, até o

momento, apresentaram os melhores resultados.

Portanto, o presente trabalho propõe encontrar a maneira mais assertiva de extrair as informações de recortes de imagens de preços de produtos para realizar a integração ao sistema *LudiiPrice*. Dessa forma, serão abordadas técnicas de processamento de imagem visando alcançar o melhor resultado para a detecção e reconhecimento de caracteres.

1.1 Objetivos

1.1.1 Objetivo geral

O objetivo geral deste trabalho consiste em desenvolver soluções e estratégias de reconhecimento de caracteres em imagens com a finalidade de realizar a extração de informações relacionadas a preços de produtos encontrados em folhetos de supermercados, analisando seus desempenhos para escolha da melhor solução.

1.1.2 Objetivos específicos

- Criação uma base de dados composta de recortes de preços de folhetos de *e-commerces*;
- Criação soluções de reconhecimento de caracteres de preços em folhetos;
- Analise e escolha da melhor solução desenvolvida;
- Integração da solução à API do *LudiiPrice*.

1.2 Metodologia

A metodologia utilizada para a realização da análise abordada por este trabalho foi dividida em criação da base de dados, exploração de técnicas de pré-processamento de imagens com uso de ferramentas de reconhecimento de caracteres (*OCR*) para desenvolvimento de uma solução e desenvolvimento de uma segunda solução baseada em *deep learning*, consistindo em um modelo de classificação de caracteres.

Para a construção da base de dados foram utilizados os resultados obtidos pela busca e recorte de preços de folhetos em sites de banco de dados de folhetos de supermercados realizados pela API criada e desenvolvida em Barbosa (2022), que resulta em recortes semelhantes aos da Figura 1. Dentre os resultados gerados foram selecionados aleatoriamente 2000 recortes de preços para servirem tanto como amostra para testes de pré-processamento de imagem, ferramentas de OCR, quanto para a construção de uma base de dados rotulada para treinamento de modelos de redes neurais para detecção de caracteres.

Figura 1 – Exemplo de *outputs* retornados pela API desenvolvida em Barbosa (2022)

Fonte: Elaborado pela autora

Para o desenvolvimento da primeira solução, após coletar a amostra necessária para conduzir os estudos de caso e testes, foram investigadas combinações e técnicas de pré-processamento de imagens que se concentram na problemática da detecção de caracteres em imagens, tendo como finalidade melhorar a nitidez, tornando mais fácil a sua leitura e aprimorando os resultados das ferramentas utilizadas para o reconhecimento de caracteres. Para essa etapa, foram seguidas principalmente as recomendações sugeridas pelo manual do usuário presente na documentação da ferramenta de OCR e extração de texto *Tesseract-OCR* Amid (2021). O Tesseract-OCR foi selecionado para realizar testes de desempenho com o objetivo de analisar a precisão dos resultados. A finalidade foi avaliar se a ferramenta atende às expectativas e pode extrair informações de maneira assertiva, visando alcançar o objetivo deste trabalho.

Já para o desenvolvimento da segunda solução, foi desenvolvido um modelo de *machine learning* utilizando conceitos de redes neurais convolucionais para realizar a detecção e reconhecimento de maneira mais assertiva dos caracteres de preços presente nas imagens. Tal modelo foi construído e baseado principalmente no trabalho acadêmico de Silva (2022) que, para atingir o objetivo de reconhecimento de caracteres de placas de veículos, utiliza técnicas de *deep learning*, treinando e aprimorando modelos de rede neurais para localizar e reconhecer os padrões dos caracteres presentes nas placas. Tal solução foi adaptada para o reconhecimento de caracteres de preço, que é a principal problemática apontada por este trabalho, acompanhado da aplicação de regras semânticas e melhorias necessárias para alcançar o melhor resultado. Por fim é feita uma análise de desempenho de ambas as soluções para selecionar a mais adequada para o problema.

1.3 Estrutura do Documento

Este documento está estruturado em capítulos, são eles:

- Capítulo 1 - Introdução: Capítulo atual, que apresenta uma contextualização, objetivos e uma breve descrição das metodologias abordadas neste trabalho;
- Capítulo 2 - Fundamentação Teórica: Conceitos importantes, tecnologias e ferramentas existentes para o entendimento deste trabalho;
- Capítulo 3 - Trabalhos Relacionados: Citação dos trabalhos que possuem objetivos semelhantes ou contribuem para a construção do objetivo deste documento;
- Capítulo 4 - Solução com ferramenta de *OCR*: Descrição dos métodos adotados para processamento de imagens, implementando uma solução de reconhecimento de caracteres usando a ferramenta de OCR *Tesseract*;
- Capítulo 5 - Solução com aprendizagem profunda: Desenvolvimento de uma solução para reconhecimento de caracteres baseada em *deep learning* e seus aprimoramentos.
- Capítulo 6 - Análise comparativa: Teste de resultados de execução das soluções propostas para comparação.
- Capítulo 7 - Considerações Finais: Considerações finais acerca do trabalho, limitações, contribuições, e trabalhos futuros.

2

Fundamentação Teórica

Para manter a fácil compreensão deste trabalho em relação a detecção e reconhecimento de caracteres em imagens, o capítulo introdutório apresenta alguns conceitos básicos, ferramentas e técnicas relevantes para o desenvolvimento de redes neurais. Isso inclui tópicos como *deep learning*, classificação de objetos e detecção de objetos. Além disso, serão abordados conceitos e técnicas específicas para processamento de imagens, como segmentação de imagens, detecção de bordas, filtragem, dentre outros.

2.1 Reconhecimento óptico de caracteres

Reconhecimento Óptico de Caracteres ou OCR, também é conhecido como reconhecimento de texto ou extração de texto. As técnicas de OCR modernas são baseadas em aprendizado de máquina e permitem extrair textos manuscritos ou impressos de imagens como pôsteres, placas de rua e etiquetas de produtos, bem como de documentos como artigos, relatórios, formulários e faturas (MICROSOFT, 2021). Normalmente, o processo de OCR envolve a análise de imagens para identificar caracteres individuais e, em seguida, compará-los com um banco de dados de caracteres conhecidos. Dependendo do tipo de texto e da linguagem envolvida, esse processo pode ser realizado de várias maneiras diferentes. Algumas ferramentas, após realizar a extração individual dos caracteres, os agrupam em palavras e realizam a equivalência com o dicionário do idioma extraído, desse modo, pode-se identificar erros de extração e a partir disso, realizar a correção da extração, retornando um resultado mais preciso.

2.1.1 *Tesseract*

Tesseract é uma ferramenta de reconhecimento óptico de caracteres de código aberto que por um tempo foi mantido pelo *Google* e atualmente está hospedado e disponível na plataforma *GitHub*. O processo de reconhecimento utilizado por esta ferramenta é baseado em algoritmos de

aprendizado de máquina e é capaz de lidar com várias fontes, tamanhos, estilos de texto e idiomas. Além disso, a ferramenta é capaz de reconhecer texto em documentos com diferentes formatos, como PDF, TIFF, JPEG, PNG, entre outros. Segundo o Manual do Usuário da ferramenta (AMID, 2021), existem vários parâmetros de configuração da ferramenta que podem ser utilizados para melhorar a sua saída, como por exemplo, especificar o idioma que deve ser feita a detecção. Além disso, é recomendado fazer o uso de processamento de imagens antes de utilizar a ferramenta para melhorar sua performance.

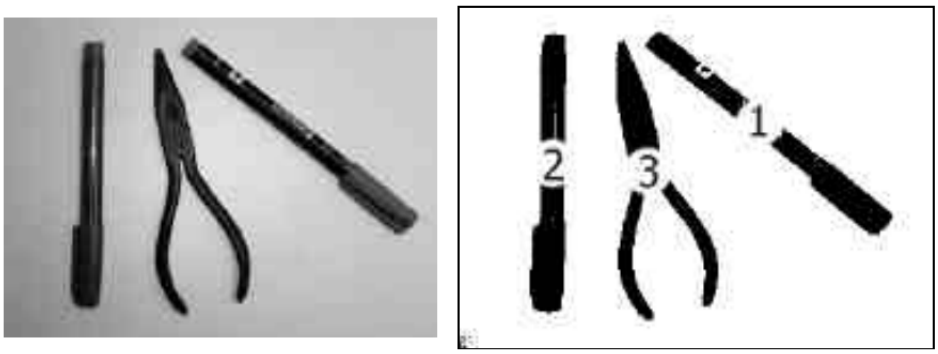
2.2 Processamento de imagens

Segundo Albuquerque (2010), processar uma imagem consiste em transformá-la sucessivamente com o objetivo de extrair mais facilmente a informação nela presente. Com o processamento de imagens, conseguimos alterar aspectos como ruído, nitidez, cores, iluminação, níveis de entrada e entre outros para poder atingir um certo objetivo, seja ele deixar uma imagem mais nítida ou até mesmo dificultar o reconhecimento do que a imagem retrata, como o objetivo de ocultar informações. Com tal poder, podemos utilizar esses conceitos para solucionar vários problemas do nosso dia a dia. Sendo assim, o processamento de imagens é fundamental para a solução de problemas como:

- Detectar distâncias entre pontos presentes em uma imagem;
- Realizar alterações ou correção de cores;
- Fazer uso de filtros lineares mais comuns que consistem geralmente na aplicação de matrizes de convolução na imagem;
- Realizar a segmentação que consiste na divisão da imagem em diferentes pedaços menores que posteriormente podem ser analisados por algoritmos para a busca de informações de nível mais alto;
- Resolver problemas de identificação e reconhecimento de objetos em imagens.

Para a construção de um sistema de decisão e classificação de objetos, o processamento de imagem é de extrema importância pois com ele podemos alterar a imagem para aumentar sua nitidez e realçar as principais características do objeto alvo e, a partir dessa etapa, realizar a segmentação para extrair as principais informações. Por fim, tendo um tratamento adequado de imagem, é mais fácil desenvolver um algoritmo de decisão que, a partir dos possíveis dados extraídos da imagem, consiga classificar objetos presentes nela presentes. A Figura 2 representa o resultado da aplicação de técnicas de processamento de imagens para a classificação de objetos.

Figura 2 – Exemplo de processamento de imagens para problemas de detecção e classificação de objetos



Fonte: (ALBUQUERQUE, 2010)

2.3 Classificação de objetos

O problema de classificação de objetos em imagens consiste na identificação de objetos de interesse na identificação de características específicas para que se possa indicar a qual classe aquele objeto pertence.

A solução mais comum e eficiente desse problema vem através do uso de redes neurais. Nessa abordagem é preciso ensinar a uma rede cada classe de interesse e suas características para que, ao chegar na etapa de classificação, o modelo criado possa atribuir, a partir das características identificadas na imagem processada, uma classe de seu conhecimento. Na etapa de treino de modelo voltado para a solução desse tipo de problema, precisamos passar como entrada imagens rotuladas com suas respectivas classes para que a rede possa associar cada classe ao seu conjunto de características.

Esse problema é consideravelmente complicado quando há um grande número de classes para serem categorizadas, ou quando há mais de um objeto de interesse presentes na mesma imagem. Nesse caso deve-se analisar a hierarquia dessas classes semânticas para conseguir uma classificação mais precisa, pois a imagem pode possuir um objeto que pertencer a várias classes simultaneamente (DRUZHKOV; KUSTIKOVA, 2016).

2.3.1 Redes Neurais Convolucionais

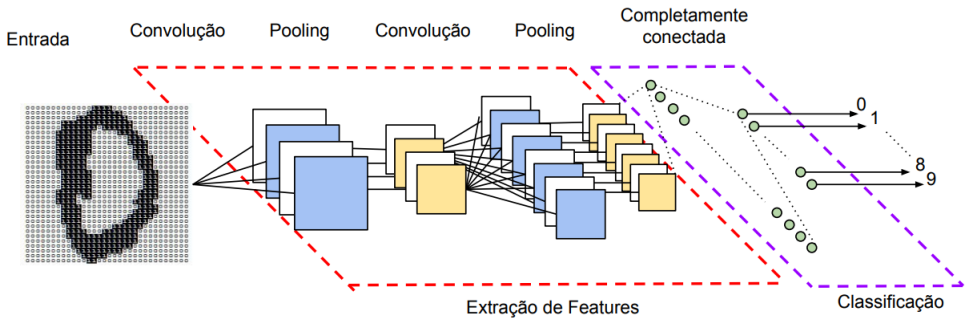
Uma Rede Neural Convolucional (*ConvNet* ou *CNN*) é um algoritmo de aprendizado profundo que processa imagens como entrada. Por meio de operações convolucionais, ela identifica e atribui importância a características presentes nas imagens, aprendendo seus padrões. Isso permite que a rede, ao se deparar com uma nova imagem, possa reconhecer esses padrões e aplicá-los em problemas de previsão de valores, classificação e detecção de objetos.

A arquitetura de uma *ConvNet* é análoga àquela do padrão de conectividade de neurônios

no cérebro humano e foi inspirada na organização do córtex visual. Os neurônios individuais respondem a estímulos apenas em uma região restrita do campo visual conhecida como Campo Receptivo. Uma coleção desses campos se sobrepõem para cobrir toda a área visual (SAHA, 2018).

Uma CNN possui no mínimo três camadas, as camadas principais, que são responsáveis pela extração das características das imagens, sendo cada camada uma operação responsável por uma sub tarefa específica, e uma camada responsável por reunir as informações extraídas e realizar operações de escolha e, ou, decisão, como pode ser visto na Figura 3. Essas camadas são:

Figura 3 – Exemplo de arquitetura de CNN para classificação de dígitos manuscritos



Fonte: (VARGAS; PAES; VASCONCELOS, 2016)

1. Camada de convolução: essa camada é responsável pela subtarefa de extrair as informações e características importantes através da aplicação de filtros na imagem de entrada e mapeando as informações encontradas na forma de dados tendo como saída um *feature map*. Um algoritmo de rede neural pode aplicar quantas camadas convolucionais forem necessárias para extrair as informações desejadas. Cada camada de convolução é normalmente seguida por uma camada de *pooling*.

2. *Pooling*: cada camada de *pooling* tem como entrada o *feature map* gerado por uma camada de convolução e é responsável por simplificar esse resultado, reduzindo as informações coletadas em um *feature map* menor, guardando as informações mais relevantes. Sua saída pode servir de entrada para outra camada de convolução ou para uma camada totalmente conexa.

3. Camada totalmente conectada: Essa camada tem como entrada o *feature map* gerado por uma camada de *pooling* e tem como objetivo reunir todas as informações extraídas durante todo o processo em uma única série de dados na sua forma de saída. É nessa camada que aplicamos a classificação ou detecção de objetos na imagem baseados no resultado obtido pela rede.

Além dessas camadas principais, existem também outras duas camadas importantes que são usadas para incrementar e otimizar o processamento das imagens dentro da rede. Elas são a camada de *dropout* e *activation function*, sendo a primeira responsável por realizar ajustes na rede quando uma rede não consegue generalizar o que foi aprendido e não consegue aplicar o que aprendeu em cima de novos dados. A segunda é responsável por indicar quais neurônios da rede serão ativados para realização da aprendizagem e é responsável pela relação das variáveis da rede (PASSOS, 2021).

As CNN é o tipo de solução sempre utilizada em problemas de classificação ou detecção de objetos. Por utilizar técnicas de segmentação de imagens, nos últimos anos, acadêmicos voltaram seus estudos para está técnica que consiste em atribuir rótulos a cada pixel da imagem com base em seu significado semântico, com o objetivo de identificar as regiões que correspondem a objetos específicos, podendo ser considerado como a chave para a solução do problema de classificação de objetos. Com isso, surgiram três dos modelos de CNN considerados, nos dias atuais, como clássicos para a solução de problemas de classificação. Esses modelos são:

1. *SegNet*: arquitetura de rede neural convolucional composta por um codificador e um decodificador, onde o codificador é responsável por extrair características relevantes da imagem e o decodificador usa essas características para gerar uma segmentação precisa da imagem;

2. *Unet*: rede estrutura em forma de "U", onde o codificador é semelhante à arquitetura convencional de CNNs, mas o decodificador usa operações de *up-sampling* para criar uma saída de segmentação com a mesma resolução da entrada;

3. *TernausNet*: rede semelhante à *U-Net* que adiciona camadas de convolução ternária, que usam três valores possíveis para os pesos dos filtros em vez de dois (positivo e negativo), para reduzir o tamanho da rede e o custo computacional;

2.4 Detecção de objetos

Segundo a pesquisa feita por Zou et al. (2019), que realiza um estudo sobre detecção de objetos em imagens nos últimos 20 anos, o problema de detecção de objetos em imagens consiste em identificar a posição onde um objeto alvo se encontra. Sendo assim, os métodos de detecção podem ser distinguidos em três grupos: abordagens baseadas na extração de características; métodos de busca de padrões; detecção de movimento.

Os métodos de *deep learning* se enquadram no primeiro grupo e são as soluções mais buscadas e exploradas para solução de tal problema. A maneira mais direta para essa abordagem é aplicar um classificador, usado para identificar as regiões de interesse geradas pela aplicação de uma janela deslizante extensa, ou algum método mais econômico. No entanto, a determinação da posição, tamanho e escala do objeto pode ser incorporada à rede neural através da adição

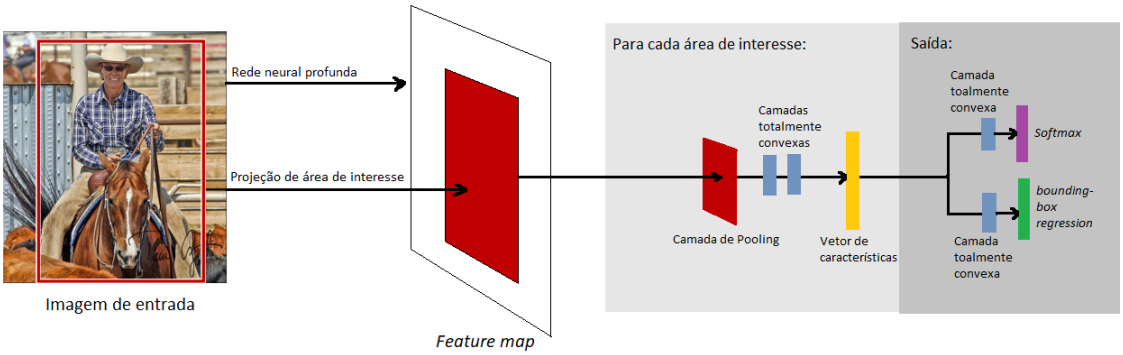
de camadas de um tipo especial para captar essas características (DRUZHKOV; KUSTIKOVA, 2016).

Atualmente, observa-se uma grande relevância no uso das redes para a detecção de objetos. Como exemplos, temos as redes neurais convolucionais *R-CNN*, *SSD*, e *YOLO* como sendo as mais utilizadas para esse tipo de solução.

2.4.1 *R-CNN*

R-CNN (Rede neural convolucional baseada em região) é um algoritmo de detecção de objetos que usa redes neurais convolucionais para identificar objetos em uma imagem. A primeira etapa, geração de propostas de região, é responsável por identificar regiões da imagem que possivelmente contêm objetos. Na segunda etapa, extração de características, cada região é convertida em um vetor de características com o auxílio de uma rede neural pré-treinada. Por fim, o classificador de máquina de vetor de suporte (*SVM*) é treinado para diferenciar regiões que contêm objetos de regiões que não contêm objetos, podendo ser visto mais detalhadamente na Figura 4.

Figura 4 – Exemplo de arquitetura de RCNN



Fonte: (GIRSHICK, 2015)

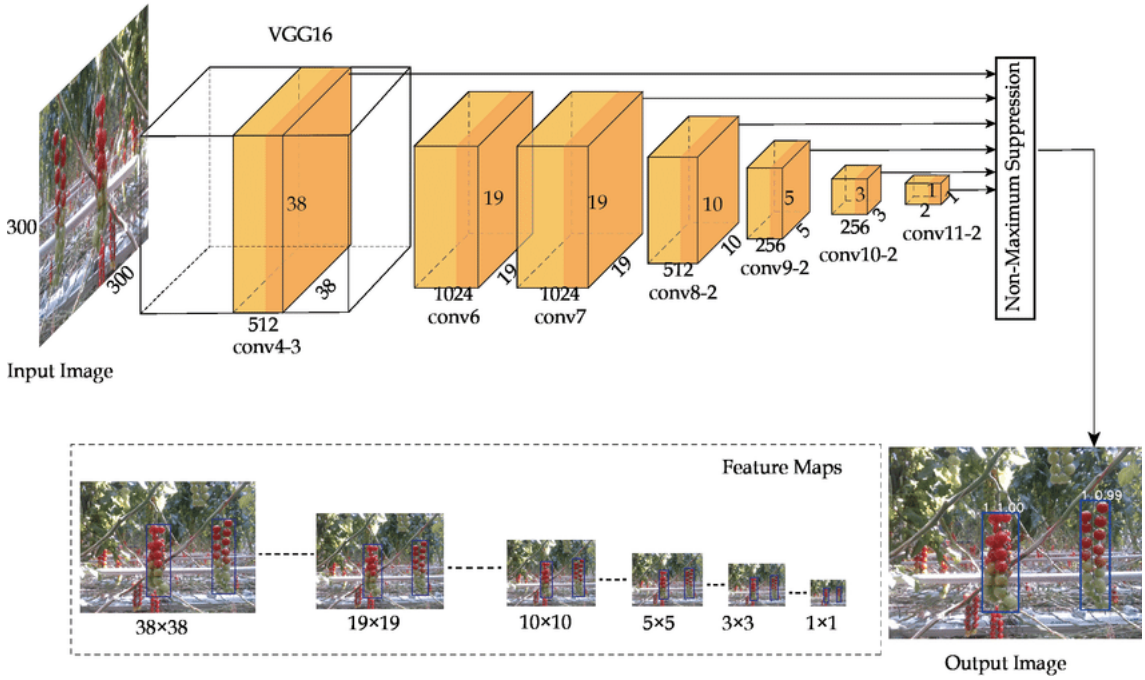
O R-CNN foi uma das primeiras abordagens de detecção de objetos baseadas em redes neurais convolucionais, e sua precisão é relativamente alta em comparação com outros algoritmos de detecção de objetos.

2.4.2 *SSD*

Tendo uma arquitetura baseada em redes de proposta de região (*RPN*), representação multi-escala e caixas de ancoramento como detalha Mazzetto (2019), a *SSD* (*Single-Shot MultiBox Detector*) é composta por duas partes sendo a primeira responsável por gerar os *feature maps* e a segunda parte aplica filtros para detectar o objeto. Para trabalhar com objetos de diversos tamanhos, a rede ajusta as previsões dos *feature maps* de variadas resoluções, ela gera um número

de *anchor boxes* de tamanhos diferentes, que juntos formam um conjunto de *bounding boxes* que englobam o objeto alvo. Com esse conjunto é possível encontrar o centro do objeto e contorná-lo corretamente. Essa arquitetura pode ser observada na Figura 5 que ilustra a arquitetura de um modelo *SSD* criado para detectar tomates em uma plantação.

Figura 5 – Exemplo de arquitetura de RCNN



Fonte: (YUAN et al., 2020)

2.4.3 *YOLO*

Uma das mais recentes abordagens para os problemas de detecção de objetos proposta, o *You Only Look Once* (*YOLO*), chegou com a solução para os problemas de complexidade computacional associados ao *R-CNN*. O *YOLO*, proposto porJoseph et al. (2015), é uma das primeiras soluções baseadas em aprendizado profundo para detecção de objetos. Ele aborda o problema de detecção de objetos como um único problema de regressão, em que as coordenadas das demarcações de área e as probabilidades das classes são calculadas simultaneamente. Isso permite uma abordagem eficiente e integrada para identificar e localizar objetos em uma imagem. Ao contrário das técnicas baseadas em janelas deslizantes, o *YOLO* analisa toda a imagem durante o tempo de treinamento e teste, codificando implicitamente informações contextuais sobre as classes, como apresentado em Joseph et al. (2015).

Mesmo com o *YOLO* tendo demonstrado vantagens de velocidade significativas sobre o *R-CNN*, estudos demonstram que o erro de localização do *YOLO* é significativamente maior do que as variantes *R-CNN* mais recentes, mas esses erros podem ser tratados com a adição de regras para garantir uma maior confiabilidade na detecção.

A versão mais avançada e atualizada é o *YOLOv7*, que apresenta um desempenho superior em termos de precisão e velocidade quando comparado às versões anteriores (*YOLOv6*, *YOLOv5* e *YOLOv4*). Isso se deve a uma série de melhorias implementadas, como um novo *backbone* de detecção e arquiteturas de rede mais simples e limpas em relação às versões anteriores. Essas melhorias tornam a implementação e personalização do *YOLOv7* mais fácil para diferentes aplicações.

O *YOLOv7* também incorpora técnicas avançadas, como aumento de dados e pré-treinamento em conjuntos de dados maiores, aumentando sua capacidade de detectar objetos em imagens e vídeos. Além disso, foram realizados aprimoramentos para mitigar a instabilidade numérica durante o treinamento e permitir o treinamento com múltiplas escalas. Essas melhorias contribuem para que o YOLOv7 tenha uma capacidade de generalização melhorada em novos conjuntos de dados e cenários (BOCHKOVSKIY; WANG; LIAO, 2021).Em resumo, o uso do modelo *YOLOv7* oferece uma série de melhorias em relação às suas versões anteriores, o fazendo uma opção atraente para muitas aplicações de detecção de objetos em imagens e vídeos.

3

Trabalhos Relacionados

Neste capítulo, são abordadas as literaturas relacionadas a detecção e reconhecimento de caracteres de imagens voltados a reconhecimento geral de números e preços de produtos, sejam em folhetos de supermercados, notas fiscais ou etiquetas de produtos.

Os trabalhos relacionados escolhidos como relevantes foram encontrados nas bases de busca acadêmicas através da *string* de busca "(extraction OR OCR OR detection) AND (flyer OR grocery OR supermarket OR price tag)".

As buscas foram realizadas nas bases de pesquisa IEE Xplore Digital[1], Google Acadêmico[2] e Research Gate[3] em Agosto de 2022. As quantidades de trabalhos encontrados em cada base podem ser observadas na Tabela 1.

Tabela 1 – Quantidade de artigos por base de dados

Base	**Quantidade de artigos**
IEE	6
Research Gate	4
Google Acadêmico	20

Como principal critério para seleção dos trabalhos encontrados durante a busca, foi considerado principalmente o ano de publicação, pois assim podemos garantir que os resultados da pesquisa estejam atualizados e incluam as tecnologias e práticas mais recentes para a solução do problema proposto. Além disso, como vários trabalhos abordavam metodologias e ferramentas semelhantes, foram selecionados representantes de diferentes técnicas que apresentassem de forma precisa e detalhada suas soluções. Após analisar os artigos encontrados na busca e aplicar o critério de exclusão, apenas 2 foram selecionados como sendo os melhores trabalhos científicos com técnicas e metodologias atuais e relevantes para a detecção e extração de

[1] Disponível em: https://ieeexplore.ieee.org/Xplore/home.jsp
[2] Disponível em: https://scholar.google.com.br/?hl=pt
[3] Disponível em: https://www.researchgate.net/

caracteres, principalmente dígitos numéricos, de imagens. Além desses dois trabalhos, mais um foi adicionado como base para estudo de solução para a problemática apresentada, sendo esse também uma solução para detecção de caracteres em imagens que não são relacionadas ao contexto de compras em supermercados.

3.1 *OCR Engine to Extract Food-Items, Prices, Quantity, Units from Receipt Images, Heuristics Rules Based Approach*

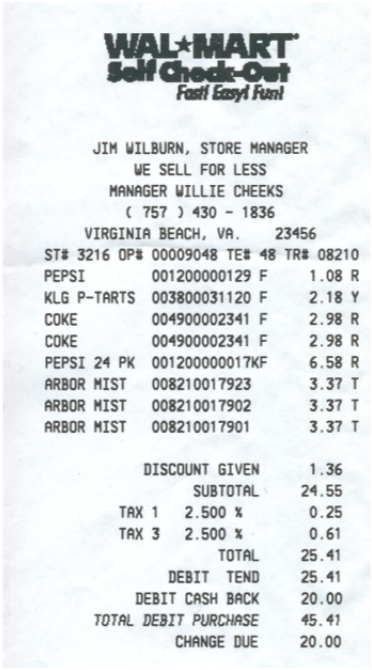
Servindo de grande base para a fase experimental deste trabalho, Ullah et al. (2018) traz uma solução para aumentar a eficiência de sistemas tradicionais de *OCR* para reconhecer nome e preços de produtos presentes em notas fiscais de supermercados, analisando a precisão da ferramenta de reconhecimento óptico de caracteres *Tesseract-OCR*, suas limitações e como contorná-las.

A metodologia apresentada pelo trabalho consiste na aplicação de uma série de operações de pré-processamento de imagens com o intuito de remover o fundo da imagem que contém a nota, realizar sua binarização, alinhar a imagem mantendo o texto o mais reto possível e realizar o redimensionamento da imagem com uma resolução de 300 dpi com a finalidade de melhorar a performance e precisão do *Tesseract-OCR*.

Em seguida, os autores aplicam a imagem ao OCR e o resultado da extração é salvo em um arquivo de texto. Após guardar o resultado da extração, os autores utilizam um algoritmo de pós-processamento para normalizar os nomes e medidas de produtos, seguindo uma abordagem heurística, resultando na extração apresentada na Figura 6.

A abordagem apresentada pode ser aplicada na solução final deste projeto principalmente na detecção e extração de caracteres de recortes de descrição de produtos. Dessa forma, podemos utilizar uma ferramenta já existente para extrair em forma de texto corrido as informações presentes na imagem da descrição de um produto e em seguida encaminhar esse resultado para um modelo de processamento de linguagem natural para identificar e categorizar as principais informações sobre o produto analizado.

Figura 6 – Resultado de aplicação de algoritmo em nota fiscal



WAL★MART
Self Check-Out
Fast! Easy! Fun!

JIM WILBURN, STORE MANAGER
WE SELL FOR LESS
MANAGER WILLIE CHEEKS
(757) 430 - 1836
VIRGINIA BEACH, VA. 23456
ST# 3216 OP# 00009048 TE# 48 TR# 08210
PEPSI 001200000129 F 1.08 R
KLG P-TARTS 003800031120 F 2.18 Y
COKE 004900002341 F 2.98 R
COKE 004900002341 F 2.98 R
PEPSI 24 PK 001200000017KF 6.58 R
ARBOR MIST 008210017923 3.37 T
ARBOR MIST 008210017902 3.37 T
ARBOR MIST 008210017901 3.37 T

DISCOUNT GIVEN 1.36
SUBTOTAL 24.55
TAX 1 2.500 % 0.25
TAX 3 2.500 % 0.61
TOTAL 25.41
DEBIT TEND 25.41
DEBIT CASH BACK 20.00
TOTAL DEBIT PURCHASE 45.41
CHANGE DUE 20.00

Item name	Price	Quantity (unit)
Pepsi	1.08	1 unit
klg p-trrts	2.18	1 unit
coke	2.98	1 unit
coke	2.98	1 unit
pepsi	6.58	24 pack
arbor mist	3.37	1 unit
arbor mist	3.37	1 unit
arbor mist	3.37	1 unit

Fonte: (ULLAH et al., 2018)

3.2 *Neural Network-Based Price Tag Data Analysis*

Laptev et al. (2022) realizaram uma comparação entre as redes neurais *Unet*, *MobileNetV2*, *VGG16* e *YOLOv4-tiny*,com o objetivo de encontrar a solução mais eficiente para o estudo de dados segmentados resultantes da problemática de segmentação de imagens de etiquetas de preços de produtos. O estudo teve como finalidade a coleta de informações relevantes, tais como o código de barras, preço, descrição, entre outros, por meio da análise de imagens de etiquetas de produtos. A análise comparativa buscou identificar a rede neural que melhor se adequasse às necessidades do estudo, a fim de obter resultados mais precisos e eficientes na coleta dessas informações.

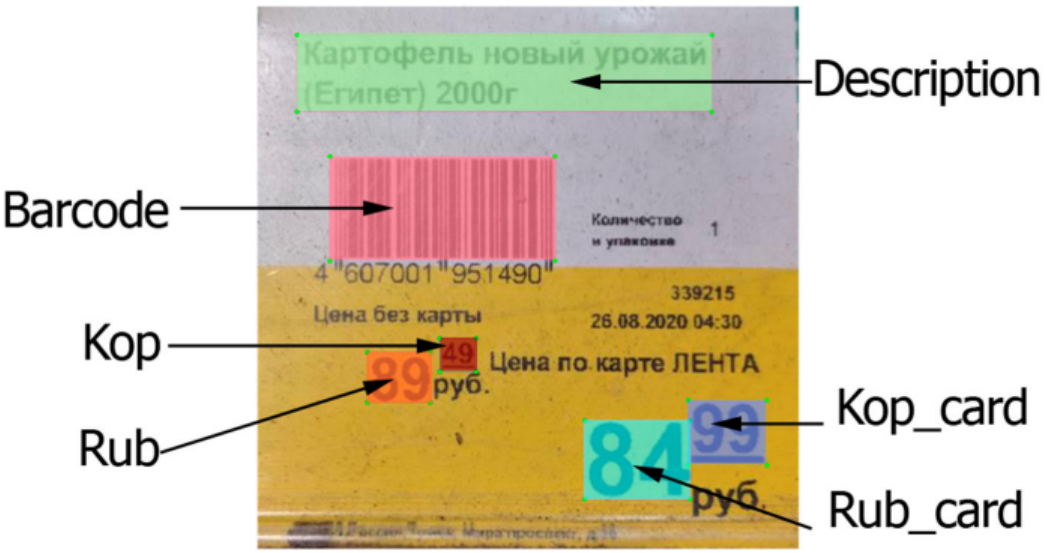
Para alcançar os resultados na análise, o trabalho científico segue os seguintes passos para a preparação do banco de dados de imagens: segmentar imagens para destacar áreas que contêm dados de interesse; selecionar as coordenadas dos segmentos; realizar o recorte dos segmentos; aplicar uma ferramenta de OCR nas áreas selecionadas; buscar informações adicionais na descrição do produto. Após a preparação da base de dados, todas as 4 redes neurais serão desenvolvidas e seus resultados serão submetidos à ferramenta de extração de texto *EasyOCR*.

As áreas de interesse das imagens de etiquetas de preços de produtos foram rotuladas da seguinte forma (Figura 7):

- *Description*: descrição do produto

- *Barcode*: código de barras do produto
- *Rub*: preço do produto em rublo russo
- *Kop*: preço do produto em kopek
- *Rub_card*: preço do produto em rublo russo com desconto Lenta aplicado
- *Kop_card*: preço do produto em kopek com desconto Lenta aplicado

Figura 7 – Exemplo de rotulação



Fonte: (LAPTEV et al., 2022)

Após a rotulação das diversas imagens do banco de dados com variados layouts de etiquetas de preço, cada modelo de rede neural foi desenvolvido e apresentado detalhadamente. Em seguida todos os modelos são submetidos a realizar a previsão em cima das imagens de teste, como demonstrado na Figura 8.

Após analisar todas as métricas de cada modelo, conclui-se que o modelo *YOLOv4-tiny* é o mais adequado para a análise de segmentação de etiquetas de preços de produtos. Ele demonstrou uma precisão de 96,92%, superando a precisão da *UNet* em 4,7%, que ocupou o segundo lugar em termos de desempenho. Portanto, o *YOLOv4-tiny* é considerado o modelo ideal para essa tarefa específica.

O resultado apresentado por Laptev et al. (2022) indica uma certa vantagem no uso do *YOLOv4-tiny* para solução de problemas de segmentação de informações em etiquetas de preço de produtos, nos induzindo a seguir com a família de modelos *YOLO* para solucionar o problema de extração de caracteres em recortes de preços de produtos.

Figura 8 – Exemplo de resultado de treino do modelo *YOLOv4-tiny*

Fonte: (LAPTEV et al., 2022)

3.3 AtalaIA : uso de Deep Learning para reconhecimento automático de placas de licença automotiva em dispositivos com recursos limitados

Silva (2022) apresenta detalhadamente o desenvolvimento de uma solução baseada em *deep learning* para realizar o reconhecimento automático de placas de veículos automotivos, fazendo uma análise exploratória de técnicas e modelos de redes neurais para alcançar o melhor resultado no reconhecimento das placas e até mesmo realizar o reconhecimento dos caracteres nela presentes.

Como metodologia para detectar e reconhecer os caracteres presentes nas placas dos veículos, os modelos utilizados na análise foram adaptados para realizar a categorização de 35 classes, sendo elas os dígitos numéricos de 0 a 9 e todas as letras maiúsculas do alfabeto, sendo o dígito zero pertencente à mesma classe da letra "O". Os modelos recebem como entrada a placa do veículo já recortada da captura realizada pela câmera de monitoramento, a partir dela é feita a previsão para detectar os caracteres e. Seguindo o padrão de placas brasileiro, são selecionadas as 7 *bounding boxes* que possuem a maior pontuação de confiança e, em seguida, elas são ordenadas de acordo com suas coordenadas, como demonstrado na Figura 9.

Aplicando a formatação utilizada nas placas de trânsito no Brasil, é possível identificar se o caractere é um número ou letra com base em sua posição. Por fim, para manter uma maior confiabilidade na detecção, foram seguidas regras de substituição para reduzir erros ao aplicar o reconhecimento, nessas regras, caso um número seja reconhecido na posição de uma letra, ele deve ser substituído pela letra que tem o formato mais semelhante ao número detectado, como por exemplo, 4 é substituído por A, 8 por B, e assim sucessivamente. O inverso acontece para letras encontradas em posições de números.

Figura 9 – Exemplo de funcionamento de modelo adaptado para detecção e reconhecimento de caracteres em placa de veículo

Fonte: (SILVA, 2022)

Após a definição das regras para detecção foram escolhidos os modelos para a realização da análise exploratória e de desempenho. Foram analisados os modelos da família *YOLO* (versões 4 *Tiny*, 5 *Nano* e *Small*), *SSD* e *CenterNet* (Figura 10). Com a escolha feita, todos os modelos foram treinados para detectar as placas dos veículos e foram adaptados para realizar também a detecção e o reconhecimento dos caracteres presentes nas placas.

Figura 10 – Exemplo de previsão de modelos em uma amostra de placa

Fonte: (SILVA, 2022)

Como conclusão o trabalho aponta o modelo *YOLOv5 Nano* como a melhor opção para realizar a detecção das placas dos veículos por apresentar um *frame rate* duas vezes maior e uma precisão de 4% superior ao modelo *YOLOv5 Small*. Já para a detecção e classificação dos caracteres, o modelo *YOLOv5 Nano* se destacou com as melhores métricas, sendo a segunda com a melhor arquitetura nas classificações das detecções, assim, sendo escolhida como ideal para prosseguir com o desenvolvimento da solução proposta pelo autor.

A metodologia apresentada no trabalho Silva (2022) se aproxima muito da solução desejada para a detecção de caracteres de preço em folhetos, onde temos um dicionário de caracteres de interesse e uma formatação a seguir. Desse modo, aplicando esse método podemos identificar com mais precisão cada caractere numérico presente em um recorte de preço, em seguida, podemos aplicar um tratamento de dados para garantir uma melhor assertividade no resultado da extração final.

4

Solução com ferramenta de *OCR*

Este capítulo apresenta a descrição de desenvolvimento de uma das soluções para a problemática abordada por este trabalho, com a finalidade de realizar um experimento comparativo com outras soluções propostas mais a frente. Essa fase tem como objetivo aplicar as técnicas de processamento de imagens e detecção e reconhecimento de caracteres encontradas, estudadas e referenciadas no capítulo Trabalhos Relacionados, mais precisamente na seção 3.1.

Como primeira solução proposta, foi considerado o uso da ferramenta de extração de texto de imagens *Tesseract-OCR*[1], aplicada através da implementação de *scripts* desenvolvidos na linguagem de programação *python*, para realizar a extração das informações de preço. Para isso, foi necessário fazer a coleta e preparação das imagens de preços, configurar a ferramenta de extração para alcançar os melhores resultados possíveis e, por fim, analisar os resultados para encontrar vantagens e desvantagens no uso da solução.

4.1 Coleta e preparação de imagens para *dataset*

Figura 11 – Exemplo de recorte de preço gerado pelo modelo descrito em Barbosa (2022)

Fonte: Elaborado pela autora

Para a coleta de imagens de preços de folhetos de supermercados, foi utilizado o modelo descrito em Barbosa (2022) para gerar 1810 recortes de preços de produtos, semelhantes ao da Figura 11, dos quais foram selecionados aleatoriamente 1030 recortes. Após selecionar as

[1] Disponível em: https://github.com/tesseract-ocr/tesseract

imagens, foram desenvolvidos dois métodos para automatizar a preparação das imagens do *dataset*. O primeiro método, seção 4.1.1, faz o uso de *deep learninge* que, com o intuito de melhorar a qualidade das imagens de entrada, padroniza as imagens de entrada. Já o segundo método, seção 4.1.2, utilizada um conjunto de técnicas de processamento de imagens mais simples, descritos no manual da ferramenta *Tesseract* (AMID, 2021), para melhorar o desempenho da ferramenta. Ambos os métodos serão descritos detalhadamente a seguir.

4.1.1 Preparação de *dataset* utilizando *deep learning*

Como primeira abordagem para realizar o pré-processamento das imagens com o intuito de deixá-las mais legíveis, foi sugerido o desenvolvimento de um auto codificador (*autoencoder*), utilizando a linguagem de programação *Python*. Este *autoencoder* consiste em um modelo de *CNN* sequencial que tem como entrada uma imagem com dimensões de 100 por 200 *pixels* e 3 canais de cores, e como saída uma imagem com mesmas dimensões, mas com apenas 1 canal de cor. O objetivo principal dessa rede é padronizar o fundo, remover o ruído e aplicar filtro de binarização a imagem de entrada. Esse modelo foi proposto para poder automatizar o pré-processamento de qualquer recorte de preço do qual se deseje extrair as informações independente da forma de extração.

4.1.1.1 Edição de imagens

Para o treino e teste do auto codificador, as imagens selecionadas foram separadas em dois conjuntos para a construção de um *dataset*. O primeiro conjunto, nomeado conjunto de treino, consiste de 1000 imagens de recortes de preço que serão utilizados, como o seu nome sugere, para treino do modelo. Já o segundo conjunto, nomeado conjunto de teste, consiste dos 30 recortes constantes que serão utilizados para testar o modelo criado.

Para criar os *targets* das imagens de entrada, todos os recortes selecionados foram submetidos a uma edição utilizando a ferramenta de edição de imagens *GIMP*[2], onde para cada padrão de preço, ou seja para cada padrão de *design* usado nos folhetos, foi realizado uma sequência de operações para realizar a remoção de ruído (alteração de contraste, nitidez e níveis de entrada), conversão para escala de cinza, inversão de cores em alguns casos e alteração na exposição para padronizar o fundo da imagem. Para ajudar nessa edição foi adicionado ao GIMP o plugin BIMP[3] (*Batch Image Manipulation*) que permite aplicar uma série de ações em um grupo de imagens através de linha de comando. Dessa forma foi possível separar as imagens que possuíam características semelhantes, descobrir a sequência de operações que traria a melhor legibilidade e, em seguida, aplicar essa sequência ao lote, como pode ser visto na Figura 12.

Após repetir o processo para todos os lotes de imagens criados, foi preciso preparar o *dataset* para realizar o treino do auto codificador. Nessa etapa foi criado um ambiente de

[2] Disponível em: https://www.gimp.org/
[3] Disponível em: https://github.com/alessandrofrancesconi/gimp-plugin-bimp

Figura 12 – Exemplo de edição de lote de recortes de preços com fundo azul e texto vermelho

Fonte: Elaborado pela autora

desenvolvimento no *Google Colaboratory*[4], que permite a execução de códigos *Python* em um navegador, sendo comumente utilizado no desenvolvimento de algoritmos de *machine learning*. Para realizar a leitura e alterações de imagens, foi utilizada a biblioteca do *Python OpenCV* (*Open Source Computer Vision Library*) que traz com ela várias operações de manipulação e conversão de imagens. Essa biblioteca permite fazer a leitura dos arquivos no formato de uma matriz de *pixels* com o tipo *nparray*, pertencente a biblioteca *NumPY*, que traz uma coleção de operações matemáticas de alto nível com suporte para processamento de grandes matrizes multidimensionais.

4.1.1.2 Preparação do *dataset*

Para o preparo do *dataset*, todas as imagens passaram por um algoritmo de pré-processamento para padronizar o tamanho das imagens e conversão dos arquivos para um formato aceito como entrada para o modelo que será desenvolvido.

Para a padronização do tamanho das imagens, foi elaborado o Código 1 que centraliza a imagem em um espaço com dimensão 200 por 100 *pixels*, adicionando *pixels* na cor preta para preencher os espaços vazios ao redor da imagem, tendo como resultado uma imagem semelhante a representada na Figura 13.

Figura 13 – Exemplo de aplicação de 1 em recorte de preço

Fonte: Elaborado pela autora

Já, para converter o formato dos arquivos criados pelo Código 1, foi implementado Código 2 que realiza a leitura dos arquivos criados pelo Código 1 utilizando a biblioteca *OpenCV*, que por sua vez transforma um arquivo de imagem em um objeto do tipo *numpy*, e utlizando a

[4] Disponível em: https://colab.research.google.com/drive/1T06vTBUx5gNJf5tSktzN8Ffx5z5XLBDv

Código 1 – Código utilizado para padronização das dimensões das imagens

```
def addPadding(path: str, width: int, heigth: int):
    image = cv2.imread(path)
    old_image_height, old_image_width, channels = image.shape
    padding_color = (0, 0, 0)
    # Cria uma imagem com as dimensoes indicadas preenchida com preto
    result = np.full((heigth, width, channels), padding_color,
        dtype=np.uint8)
    # Calcula o centro da imagem
    x_center = (width - old_image_width) // 2
    y_center = (heigth - old_image_height) // 2
    # Faz uma copia da imagem original no centro da criada
    result[y_center:y_center+old_image_height,
        x_center:x_center+old_image_width] = image
    return result
```

Fonte: Elaborado pela autora

biblioteca *NumPy* esses arquivos são salvos agora no formato de arquivo *numpy*. Essa conversão foi feita pois arquivos com a extensão *npy* são um formato de arquivo aceito como entrada do modelo e, por se tratar de um arquivo de texto, tem um tempo de leitura mais rápido comparado com um arquivo de imagem. Assim, com essa conversão, o processo de treino pode ser acelerado pois, desse modo, a cada execução do código de treino, não é necessário realizar a leitura e conversão das imagens do *dataset*.

Código 2 – Código utilizado para conversão das imagens para arquivos *numpy*

```
def saveTrainAsNparray(filesPath: str, labelPath: str, arrayPath:
    str, labelArrayPath: str):
    train = []
    label = []

    files = getFiles(filesPath)
    filesLabel = getFiles(labelPath)
    qtd_arquivos = len(files)
    for i in range(0, qtd_arquivos):
        img = cv2.imread(f'{filesPath}/{files[i]}')
        label_img = cv2.imread(f'{labelPath}/{filesLabel[i]}')
        file_name = files[i].replace('.jpg', '.npy')
        label_name = filesLabel[i].replace('.png', '.npy')
        np.save(f'{arrayPath}/{file_name}', img)
        np.save(f'{labelArrayPath}/{label_name}', label_img)
```

Fonte: Elaborado pela autora

4.1.1.3 Desenvolvimento do modelo

O modelo utilizado foi criado com o auxílio da biblioteca *Keras*, utilizando o modelo de rede neural sequencial para construir um auto codificador. Cada camada foi adicionada de acordo com as necessidades do problema, reduzindo a imagem em **feature maps** com as informações das áreas de interesse representadas pelos caracteres numéricos de preço, buscando coletar as principais características para depois reconstruir a imagem em seu tamanho original apenas com as informações relevantes.

Para automatizar o pré-processamento dos recortes de preços, a primeira versão do modelo usou uma camada de entrada que aceita uma imagem de dimensão de 200 por 100 *pixels* e 3 canais de cores. A camada de saída resulta em uma imagem com as mesmas dimensões, mas com apenas um canal de cores. Essa imagem com cores binárias é composta por *pixels* de valores 0, representando a ausência de informação (ou seja, o fundo da imagem), e por *pixels* de valores 1, representando a área de interesse, ou seja, o preço presente na imagem.

Após a montagem do modelo sequencial, foi realizado o treino tendo os 1000 recortes pré-processados e suas versões editadas sendo suas "*labels*", como descrito em 4.1.1.1 sendo usadas como alvo para aprendizagem da rede, formando o conjunto de treino e as 30 imagens restantes e suas versões editadas como conjunto de teste. O modelo foi treinado utilizando 200 *epochs* e durante o seu treino, foi possível analizar que durante as primeiras *epochs* a função de perda teve uma redução significativa, mas por volta da *epoch* 80 o valor da função de perda se manteve oscilando entre 7 e 9, indicando que o modelo não estava sendo capaz de aprender novos padrões para realizar as predições.

Com isso, uma segunda versão do modelo foi criada para servir o mesmo propósito da primeira versão, mas separando o fundo da imagem da área de interesse, ou seja, a nova versão teria a mesma camada de entrada e como saída retornaria duas imagens, sendo a primeira uma apresentação da área de interesse do recorte e a segunda corresponderia a seu *background*. Cada camada junto com seus parâmetros de entrada e saída podem ser vista de forma detalhada no Apêndice A.

Após montar o novo modelo, o *dataset* foi adaptado para suportar mais uma *label* aos conjuntos de treino e teste. Essa nova label corresponde à versão de saída anterior do modelo, mas com suas cores invertidas. Dessa forma uma *label* corresponde ao fundo da imagem, enquanto a outra, corresponde à área de interesse da imagem.

Repetindo todo o processo de treino do modelo com os mesmos parâmetros da versão anterior, foi observado o desempenho de aprendizado a partir da função de perda que, por sua vez apresentou valores muito maiores do que a versão implementada anteriormente, precisando receber vários ajustes e testes para apresentar melhores resultados. Nesta etapa do processo, o tempo destinado para a fase experimental estava próximo do fim, por conta disso, uma segunda abordagem mais dinâmica foi sugerida para poder realizar os testes com a ferramenta de OCR e

analisar seus resultados dentro do prazo.

4.1.2 Preparação de *dataset* utilizando processamento de imagens

Na segunda abordagem explorada, foram seguidas as instruções apresentadas na documentação do *Tesseract-OCR* (AMID, 2021). Dentre as técnicas citada na referência, foram utilizados em cada imagem a seguinte sequência de técnicas:

1. Redimensionamento: a imagem é padronizada tendo sua largura fixada para 300 *pixels* e tendo a sua altura aumentada proporcionalmente;
2. Conversão de cores: a imagem tem suas cores convertidas do *RGB/BGR* para uma escala de cinza;
3. Aplicação de *blur*: uma função de *Gaussian Blur* é aplicada a imagem com o intuito de realizar a remoção de ruído;
4. Binarização: a imagem é binarizada com a limiarização utilizando o método de Otsu (TOROK, 2016);

Para realizar os passos citados acima, foi criado um *script*, que pode ser visto no Código 3,na linguagem de programação Python que, principalmente, através dos métodos de manipulação de imagem presentes na biblioteca OpenCV, tornou possível realizar de maneira rápida e prática processamento das imagens e realizar os testes com a ferramenta de extração. Nesse *script*, foram aplicadas as seguintes operações:

1. Aplicação do redimensionamento padrão da biblioteca com largura fixa de 300 *pixels*;
2. Conversão de cores do tipo BGR para escala de cinza padrão da biblioteca;
3. Aplicação de uma máscara de de dimensões 3x3 aplicando um *blur* gaussiano;

Um exemplo da aplicação das operações listadas acima pode ser vista na Figura 14, onde na esquerda temos a imagem de entrada do *script*, enquanto na direita se encontra o resultado da aplicação do *script*.

Código 3 – Código utilizado para processar as imagens seguindo os passos citados

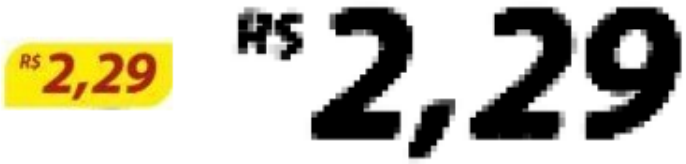
```
def preprocessamento(path: str, file: str):

    image = cv2.imread(path)
    image = image_resize(image, width=300)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(image, (3, 3), 0)
    _, otsu = cv2.threshold(blur, 0, 255,
        cv2.THRESH_BINARY+cv2.THRESH_OTSU)

    cv2.imwrite(fr"out\{file}", otsu)
```

Fonte: Elaborado pela autora

Figura 14 – Exemplo de aplicação do script de pré-processamento



Fonte: Elaborado pela autora

4.2 Configuração do Tesseract-OCR

Para a configuração da ferramenta de extração de texto, foram seguidas as instruções também apresentadas em Amid (2021). O primeiro parâmetro a ser configurado foi o *psm* (método de segmentação da página), sendo escolhida a opção 7, que corresponde a interpretação do texto presente na imagem sendo como sendo um texto de linha única. Como o *Tesseract* é otimizado para reconhecer padrões de frases e palavras, é recomendado desabilitar os dicionários e lista de palavras (*load_system_dawg* e *load_freq_dawg*). Por fim, para prevenir interpretações errôneas, como interpretação de letras ao invés de números, uma lista de caracteres permitidos é criada contendo todos os dígitos entre 0 e 9 e os caracteres "."e ","para indicar a separação de reais e centavos no preço. Todas essas configurações foram implementadas em Código 4

Código 4 – Código utilizado para realizar a extração do texto das imagens com a ferramenta *Tesseract-OCR* configurada

```
1 def ocr(path: str) -> str:
2     image = Image.open(path)
3     ocr_result = pytesseract.image_to_string(image,
          lang=language,config="--psm 7 --oem 1 -c
          tessedit_char_whitelist=0123456789., load_system_dawg=0
          load_freq_dawg=0")
4
5     return ocr_result
```

Fonte: Elaborado pela autora

4.3 Testes preliminares

Para testar e analisar o desempenho da ferramenta, foram selecionadas 30 recortes tentando manter equilibrada a quantidade de imagens para cada padrão de *design* de preço. Após essa separação, as imagens foram submetidas ao pré-processamento já descrito e serviram de *input* para a ferramenta de extração. Na Figura 15 podemos observar o comportamento da ferramenta em alguns casos.

Figura 15 – Amostra de teste de ferramenta em imagens pré-processadas

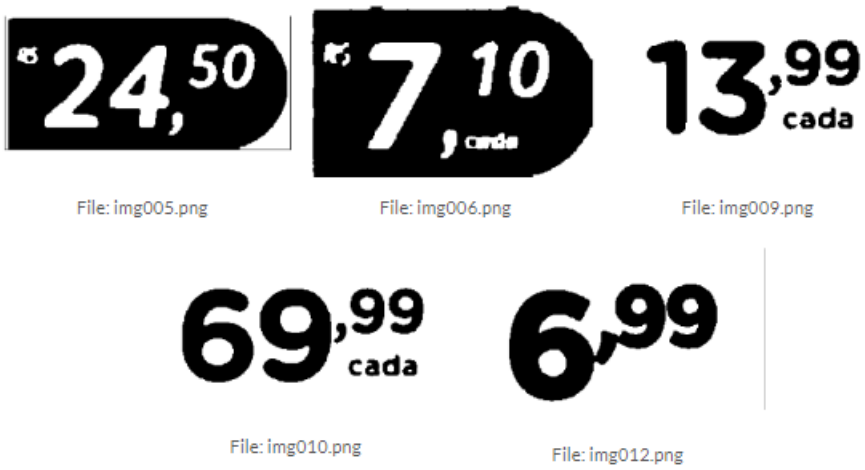
Fonte: Elaborado pela autora

Em geral, a ferramenta apresentou bons resultados em imagens que originalmente já continham uma qualidade maior, um grande contraste e possuíam seus dígitos alinhados, como no caso das imagens 001, 002, 003 e 004 na Figura 15. No entanto, em alguns casos recorrentes, tanto por conta do desalinhamento que ocorre entre os dígitos correspondentes aos reais e aos de

centavos, como também no posicionamento da vírgula que realiza essa divisão, a ferramenta não foi capaz de ler com precisão o caractere de vírgula, como também, no caso de desalinhamento dos caracteres, alguns dígitos em posições mais elevadas e tamanho menores, como no caso das imagens 005, 006, 009, 010 e 012, também na Figura 15

Analisando esses últimos cinco casos citados, podemos interceptar o processo de extração para analisar as imagens antes de passar pela ferramenta de *OCR*. Como apresentado na Figura 16, o pré-processamento conseguiu deixar os caracteres de preço destacados em relação ao fundo de cada imagem e bem mais legíveis comparado com as imagens originais de cada caso. Mas mesmo com o destaque e legibilidade, a ferramenta não foi capaz de reconhecer alguns dos caracteres.

Figura 16 – Imagens de teste 005, 006, 009, 010 e 012 após de passar pelo *script* 3



Fonte: Elaborado pela autora

Um dos erros mais graves e desafiadores de resolver é a incapacidade de ler caracteres resultando em respostas vazias, mesmo com todo o esforço em melhorar a qualidade do resultado da ferramenta. Isso ocorreu, por exemplo, no caso da imagem 013 (Figura 15) e em imagens com baixíssima qualidade, o que tornou o uso da ferramenta pouco confiável para o propósito deste trabalho. Além da importância de reconhecer corretamente todos os dígitos dos preços, é fundamental que o caractere ","ou "."seja identificado corretamente para separar reais e centavos no preço, a fim de validar os resultados da extração. Infelizmente, o *Tesseract-OCR* não consegue fazer isso com precisão.

Dessa forma, para conseguir uma solução ótima para a problemática apresentada por este trabalho, será necessário o desenvolvimento de uma nova abordagem baseada no uso de *deep learning* para realizar a detecção e reconhecimento dos caracteres semelhante com o que foi abordado no trabalho relacionado Silva (2022), sendo todo o processo para este desenvolvimento descrito na seção a seguir.

5

Solução com aprendizagem profunda

Este capítulo apresenta o desenvolvimento de uma segunda solução para a problemática abordada por este trabalho, com o objetivo de realizar uma análise comparativa com a solução proposta anteriormente. Nesta fase será desenvolvido um modelo *CNN* para reconhecer individualmente os caracteres presentes nos recortes de preço.

O preço, diferente de palavras escritas que podem ser validadas com um pós processamento e serem comparadas com um dicionário, necessita de uma identificação única e precisa para cada caractere numérico, desse modo, podemos diminuir a quantidade de erros de uma extração. Junto com a aplicação de regras após a detecção, como visto em Silva (2022), é possível garantir que cada caractere seja reconhecido com maior assertividade e, validando a formatação do preço para o padrão que utilizamos no mercado. Com isso, para uma segunda solução do problema de detecção de caracteres numéricos em recortes de preço foi desenvolvido um modelo que reconhece e classifica caracteres numéricos e outros recorrentes em padrões de preço. Esse modelo foi treinado usando uma abordagem de aprendizado de máquina supervisionado e validado em um conjunto de dados de preços com diferentes formatos, estruturas e resoluções.

A descrição das ferramentas auxiliares para essa etapa de desenvolvimento e a descrição de cada passo realizado serão descritas detalhadamente a seguir.

5.1 Ferramentas utilizadas

Para endereçar uma solução para o problema proposto, diversas ferramentas foram utilizadas para auxiliar em diversas etapas do desenvolvimento, desde a coleta e preparação de dados até a implementação de regras semânticas. Nesse caso, a linguagem de programação *Python* e as ferramentas *Dataturks*[1], *Google Colab*[2] e *YOLOv7*[3] foram as principais escolhas para

[1] Disponível em: https://dataturks.com/
[2] Disponível em: https://colab.research.google.com/
[3] Disponível em: https://github.com/WongKinYiu/yolov7

o desenvolvimento, *Dataturks* foi usado para preparação de dados, *Google Colab* foi usado como ambiente de desenvolvimento e *YOLOv7* foi usado como a principal ferramenta para alcançar os resultados desejados. As seções a seguir descrevem cada ferramenta em detalhes.

5.1.1 Dataturks

Dataturks é uma ferramenta de anotação de dados criada para auxiliar na construção de *datasets* formados por conjuntos de dados rotulados para treinamento de modelos de *machine learning*. A ferramenta permite que os usuários usem seus próprios conjuntos de dados e os rotulem com informações relevantes, como categorias, *tags*, anotações de texto e regiões de interesse. Para o desenvolvimento deste projeto, o Dataturks foi utilizado para realizar a rotulação dos caracteres especiais e numéricos presentes em recortes de preços de folhetos de supermercados. Para utilizá-lo foi feita uma instalação baseada em *Docker*[4], sendo hospedada em uma instância do *AWS*[5].

5.1.2 Google Colab

O *Google Colaboratory*, ou em termos curtos *Colab*, é um ambiente de desenvolvimento em linguagem de programação *Python* em navegadores, desenvolvido pela *Google*. Uma das principais vantagens do Colab que foi muito utilizada no desenvolvimento deste projeto é a integração com os demais serviços do *Google*, como o *Google Drive*, que realiza armazenamento e compartilhamento de dados na nuvem, e o *Google Sheets*, ferramenta de criação e edição de planilhas. Além disso, o *Colab* também oferece suporte a bibliotecas populares de aprendizado de máquina e ciência de dados, como o *TensorFlow*, *PyTorch* e *Pandas*, bibliotecas também utilizadas no desenvolvimento da solução.

Esta ferramenta serviu como ambiente de desenvolvimento do treino do modelo de rede neural utilizado para a detecção de caracteres, bem como para sua execução e aplicação das regras semânticas que processam os resultados obtidos aumentando a qualidade do *output* final. Junto a este ambiente também foram utilizadas diversas bibliotecas que, em cada etapa, auxiliaram o desenvolvimento da solução.

5.1.3 Yolov7

O *YOLOv7* (*You Only Look Once version* 7) é um modelo de rede neural convolucional utilizado para detecção de objetos em imagens e vídeos em tempo real pertencente à família *YOLO*, descrita na seção 2.3.2. Ele utiliza uma abordagem de detecção de objetos em uma única etapa (*one-stage detection*), que é mais rápida e precisa do que abordagens em duas etapas (*two-stage detection*). O YOLOv7 é uma evolução do YOLOv5 e apresenta melhorias

[4] https://hub.docker.com/r/klimentij/dataturks
[5] http://3.82.191.59

como uma nova arquitetura de rede neural, aumento do número de camadas, uso de técnicas de aumento de dados (*data augmentation*) e pré-processamento de imagens. Ele é utilizado em diversas aplicações como vigilância por vídeo, detecção de objetos em automóveis autônomos e reconhecimento de objetos em tempo real em jogos (BOCHKOVSKIY; WANG; LIAO, 2021).

Como mencionado na seção 2.3.2, esta versão é a mais recente e simplificada para a implementação de uma versão personalizada, proporcionando resultados mais rápidos e precisos. Portanto, é a escolha ideal para resolver o problema em questão. Assim, o modelo foi treinado especificamente para detectar caracteres especiais e numéricos presentes nas imagens recortadas de preços.

5.2 Criação do dataset

A criação de *datasets* é uma etapa fundamental para o desenvolvimento de modelos de *machine learning*, seja ele um modelo com finalidade de detecção de objetos, reconhecimento de padrões e até mesmo classificação de objetos. Para a realização dessa tarefa, pode-se usar diversas ferramentas que auxiliam as etapas de coleta e rotulação de dados. Neste projeto, para realizar a detecção de caracteres de preço, foram coletados uma gama de recortes de preço de produtos para construir um *dataset* composto por 3 conjuntos, sendo eles imagens para treino, validação e teste. tais processos serão detalhados a seguir.

5.2.1 Coleta e seleção de imagens

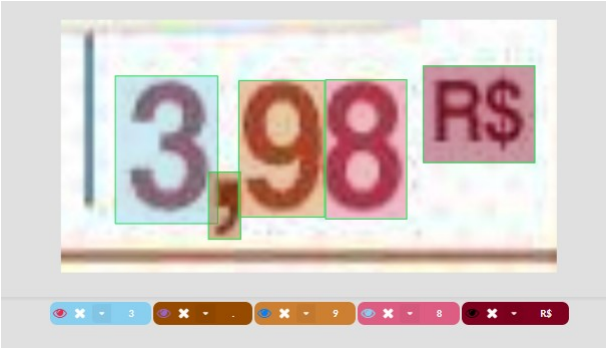
Para a coleta de imagens, foram seguidos os mesmos passos apresentados em 4.1, onde foram escolhidas dos resultados retornados pelo modelo desenvolvido em Barbosa (2022), 1000 imagens de recortes de preço, vale ressaltar que dentro dos resultados retornados também estão inclusos detecções errôneas. Assim, alguns dos recortes não correspondem a preços, podendo ser imagens de produtos e até mesmo descrições. A partir desta seleção começou-se a etapa de rotulação.

5.2.2 Rotulação

Com o objetivo de concluir a tarefa de rotulação das imagens de forma eficiente e com alta acurácia, foi escolhido a ferramenta *Dataturks* e, como já mencionado anteriormente, esse software permite a classificação de segmentos de imagem em diferentes categorias. Dessa forma, essa ferramenta foi utilizada para demarcar a área de cada caractere de preço presente no recorte e classificá-las de acordo com suas classes correspondentes. Para realizar essa etapa, foram escolhidas as classes que representassem os números de 0 a 9, ponto decimal (.), símbolo da moeda (R$) e uma categoria para representar outros símbolos que possam aparecer nos recortes, se encaixando nela partes de descrições ou imagens de produtos e outras informações irrelevantes para a leitura de um preço.

Após configurar a ferramenta de rotulação com as classes citadas, as 1000 imagens selecionadas no processo anterior foram submetidas ao processo de rotulação das áreas de interesse, como demonstrado na Figura 17, seguindo as classificações descritas anteriormente.

Figura 17 – Exemplo de rotulação de recorte de preço utilizando a ferramenta *Dataturks*



Fonte: Elaborado pela autora

5.2.3 Montagem

Com as imagens rotuladas, deu-se início ao processo de montagem do *dataset*. Primeiro, como o modelo de rede neural usa formatos específicos de *input*, foi preciso converter o arquivo de rotulação gerado pelo *Dataturks* para o modelo de *input* aceito pelo *YOLOv7*, que consistem em um arquivo em formato de texto contendo as identificações de cada classe rotulada, seguida dos pontos centrais das coordenadas de cada marcação de área de interesse e da largura e altura dessa área respectivamente, como pode ser visto na Figura 18. Cada arquivo de imagem utilizado durante o treino do modelo, necessita de um arquivo de rotulação próprio, referidos daqui em diante como arquivo de *label*. Esses arquivos de *labels* são associados as suas respectivas imagens durante o início do processo de treino e são usados pelo modelo como referência para o seu aprendizado.

Figura 18 – Exemplo de anotação no modelo aceito pelo *Yolov7* e resultante da aplicação do 5

```
0 0.170791 0.336211 0.139051 0.277327
1 0.338560 0.499568 0.187417 0.626834
2 0.448894 0.782593 0.120914 0.303920
3 0.615907 0.516663 0.185906 0.615437
4 0.789721 0.510965 0.197997 0.634432
```

Fonte: Elaborado pela autora

Para realizar a conversão do arquivo de rotulação gerado pela ferramenta *Dataturks*, que possui estrutura demonstrada na Figura 19, foi implementado um *script* que, para cada linha presente no arquivo que se deseja converter, um novo arquivo de *label* é criado. Assim, em cada linha da rotulação temos informações relevantes para conversão como:

- *content*: referencia o arquivo de imagem dauqela rotulação;
- *annotation*: consiste na lista de rótulos feitos naquela imagem. cada rótulo possui uma *label* e *points*, descritos a seguir;
- *label*: indica o nome da classe a qual aquela rótulo foi atribuído;
- *points*: consiste em uma lista de pares de coordenadas de cada ponto do rótulo desenhado;
- *imageWidth*: indica a largura da imagem que foi rotulada;
- *imageHeight*: indica a altura da imagem que foi rotulada;

Figura 19 – Exemplo de arquivo de rotulação gerado pela ferramenta *Dataturks*


```
"content": "/uploads/2c91808284b5bab601852c7b0b240014/08009c0a-25f1-4d89-9894-3add5a9a15a0___test0151.jpg",
"annotation": [
    {
        "label": ["R$"],
        "shape": "rectangle",
        "points": [[0.24151308674193306, 0.16100872449462206], [0.0776292064527642,0.16100872449462206], [0.0776292064527642, 0.3910211880583679],[0.24151308674193306, 0.3910211880583679]],
        "notes": "",
        "imageWidth": 112,
        "imageHeight": 63
    },
    {
        "label": ["9"],
        "shape": "rectangle",
        "points": [[0.500277108251147,0.16484226555401782], [0.23720035305011283,0.16484226555401782], [0.23720035305011283,0.8702138204828384], [0.500277108251147,0.8702138204828384]],
        "notes": "",
        "imageWidth": 112,
        "imageHeight": 63
    }]
```


Fonte: Elaborado pela autora

A partir dessas informações, podemos criar o arquivo de *label* no modelo *Yolov7*, onde cada linha é composta por um elemento presente na lista de *annotation*. Com isso, para cada linha dessa lista, aplicamos o *script* de conversão Código 5 que, recebe o nome da classe que aquele rótulo foi atribuído, e as demais informações da anotação. Com isso, obtemos os pontos mínimos e máximos de cada eixo da imagem e calculamos sua posição central em relação ao resto da imagem, calculando também a altura e largura da área rotulada. Com todas as informações necessárias, é feita a a formatação da linha a ser escrita no novo arquivo, repetindo o processo para todos os elementos da lista de *annotation*.

Código 5 – Código utilizado para realizar a conversão da rotulação do padrão utilizado pelo Dataturks para o padrão YOLO

```
def convert_to_yolo(label, data):
    #label -> nome da classe de rotulacao
    #data -> informacoes do posicionamento da area rotulada

    labels = ["R$", "6", ".", "9", "5", "outros", "8", "4", "3",
        "2", "7", "0", "1", "X"]
    image_width = data['imageWidth']
    image_height = data['imageHeight']

    xmin = image_width * min(data['points'][0][0],
        data['points'][1][0], data['points'][2][0],
        data['points'][3][0])
    ymin = image_height * min(data['points'][0][1],
        data['points'][1][1], data['points'][2][1],
        data['points'][3][1])
    xmax = image_width * max(data['points'][0][0],
        data['points'][1][0], data['points'][2][0],
        data['points'][3][0])
    ymax = image_height * max(data['points'][0][1],
        data['points'][1][1], data['points'][2][1],
        data['points'][3][1])

    # calculo da proporcao de coordenadas
    x_center = ((xmax + xmin) / 2.0) / image_width
    y_center = ((ymax + ymin) / 2.0) / image_height
    width = (xmax - xmin) / image_width
    height = (ymax - ymin) / image_height

    return labels.indexof(label) + (" %.6f %.6f %.6f %.6f\n" %
        (x_center, y_center, width, height))
```

Fonte: Elaborado pela autora

Após realizar a conversão das 1000 imagens, essas foram divididas em 2 conjuntos, enquanto mais 200 imagens, que foram coletadas da mesma maneira das primeiras 1000, foram adicionadas para completar a montagem do *dataset*, formando o terceiro conjunto. Desse modo, a divisão foi a seguinte:

- Conjunto de treino: composto por 900 imagens e suas respectivas *labels*;
- Conjunto de validação: composto por 100 imagens e suas respectivas *labels*;
- Conjunto de teste: composto pelas 200 imagens adicionais;

5.3 Desenvolvimento do modelo

Nesta seção, discutiremos o processo de desenvolvimento de um modelo *YOLOv7* para a detecção de caracteres em recortes de imagens de preço, configurando o modelo para realizar um treino com um *dataset* personalizado seguindo as nomenclaturas estabelecidas na fase de rotulação das imagens do *dataset* descritas na seção anterior.

5.3.1 Configuração

O primeiro passo para configuração do modelo para realizar o treino com um *dataset* personalizado foi preparar o ambiente de desenvolvimento no *Google Colab*, importando as bibliotecas necessárias para execução do modelo, clonando o repositório do *YOLOv7* e preparando o modelo para receber um treino com dados customizados. Os passos realizados a seguir foram feitos para a criação de uma customização de modelo utilizando as 13 classes definidas no processo de rotulação.

Em seguida, foram feitas cópias dos arquivos de configuração do modelo para alteração e customização segura. Primeiro foi feita uma cópia do arquivo "yolov7.yaml"presente no diretório "yolo/training"(repositório do *YOLO*). Nessa cópia foi alterado somente o número de classes utilizadas para 13, correspondendo ao número de classes obtidas na rotulação. Em seguida foi feita uma cópia do arquivo "coco.yaml"presente no diretório "yolo/data", alterando as seguintes configurações:

- Caminho para arquivos de treino: ./data/13classes/treino
- Caminho para arquivos de teste: ./data/13classes/validacao Caminho para arquivos de teste: ./data/13classes/teste
- Número de classes: 13
- Nome das 13 classes: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., R$, outros

Por fim, os dados dos conjuntos do *dataset* foram colocados nos seus respectivos diretórios já citados acima. No caso do conjunto de treino e validação, os arquivos foram separados dentro de seus diretórios sendo uma pasta para as imagens e outra para as respectivas *labels*.

5.3.2 Treino

Com o modelo devidamente configurado, começou-se as rodadas de treino utilizando o peso padrão do *yolov7* como base, observando as métricas resultantes e testando os parâmetros utilizados para encontrar os melhores resultados. Por fim, os seguintes parâmetros foram escolhidos:

- Tamanho do lote: 16
- Epochs: 50
- Tamanho da imagem: 320 x 320

Ao concluir o treino com as 50 *epochs*, o modelo apresentou as métricas descritas na Tabela 2 como resultado.

Tabela 2 – Métricas do treino do modelo

Classes	***Precision***	***Recall***	***mAP*@0.5**	***mAP*@0.5:0.95**
all	96,9%	96,5%	97,7%	72,7%
R$	97,4%	98,0%	75,4%	55,9%
6	98,8%	94,5%	92,0%	48,8%
.	94,1%	79,7%	93,8%	62,0%
9	98,1%	93,9%	97,6%	71,4%
5	98,9%	97,3%	97,9%	72,5%
outros	98,2%	100%	99,5%	82,3%
8	97,7%	100%	99,5%	86,4%
4	97,8%	95,7%	95,5%	83,9%
3	94,2%	100%	98,1%	75,7%
2	100%	95,1%	99,5%	74,1%
7	98,2%	100%	99,5%	75,7%
0	96,5%	100%	99,5%	82,3%
1	99,7%	100%	99,6%	74,4%

5.3.3 Regras semânticas

Com o modelo pronto para realizar as detecções dos caracteres de preço, algumas imagens foram submetidas à detecção para verificar o resultado puro retornado pelo modelo. A partir disso foram criadas regras semânticas que, em resumo, são instruções que estabelecem uma relação entre os elementos de um conjunto de dados ou informações, sendo utilizadas para validar dados, processá-los ou interpretá-los. Nesse caso, as regras foram pensadas e criadas para tratar os resultados do modelo, aproximando-os da leitura real de um preço feito por um humano.

Como o resultado retornado pelo modelo consiste em um *dataframe*, a biblioteca *Pandas*[6] foi utilizada para manipular este tipo de dado e aplicar as regras criadas. A seguir serão descritas as regras semânticas criadas para o tratamento do *output* do modelo.

5.3.3.1 Regar 1 - Correção de nomenclatura de *labels*

O primeiro ponto observado foi a nomenclatura das classes retornadas na detecção do modelo. Durante a etapa de treino, houve um equívoco na configuração do modelo e a nomenclatura das classes foi inserida fora da ordem estabelecida na conversão feita pelo Código 5. Por conta desse erro, a detecção era feita corretamente, mas o nome da classe de cada detecção não correspondia ao número real ali representado, como pode ser visto na Figura 20.

[6] https://pypi.org/project/pandas/

Figura 20 – Exemplo de resultado de detecção com nomenclatura *labels* não correspondentes com imagem

	xmin	ymin	xmax	ymax	confidence	class	name
0	11.830430	9.534207	31.780355	25.690315	0.860929	0	R$
1	30.840948	10.342523	60.493511	54.935070	0.942646	9	6
2	61.028389	10.695570	89.679451	54.613766	0.929469	9	6
3	90.008255	25.518642	99.905441	42.414494	0.853569	2	outros
4	99.525146	11.154955	116.324471	37.959347	0.890751	3	0
5	108.499214	39.590118	130.149353	51.933529	0.742253	5	2
6	116.587105	11.305234	133.548752	37.923656	0.883130	11	8

Fonte: Elaborado pela autora

Assim, foi necessário realizar uma correção de nomenclatura, para que quando a detecção for transformada em texto plano, os caracteres sejam extraídos com o nome correspondente ao ilustrado na imagem. Desse modo, a regra se apresenta no Código 6, onde, seguindo a indexação da nomenclatura das *labels* tanto na configuração do modelo, como em Código 6, conseguimos encontrar a equivalência entre as *labels* e trocá-las pelo valor correto. Então para cada *dataframe* resultado pelo modelo, todas as detecções terão suas *labels* corrigidas.

Código 6 – Código utilizado para correção da nomenclatura das *labels*

```
def regra1(df):
    # df -> DataFrame contendo resultado da deteccao
    label_dict = {
          "R$": "R$",
          ".": "6",
          "outros": ".",
          "0": "9",
          "1": "5",
          "2": "outros",
          "3": "8",
          "4": "4",
          "5": "3",
          "6": "2",
          "7": "7",
          "8": "0",
          "9": "1"
    }
    for i, row in df.iterrows():
        class_name = row['name']
        df.at[i,'name'] = label_dict[class_name]
    return df
```

Fonte: Elaborado pela autora

5.3.3.2 Regra 2 - Ordenação das labels

Um dos pontos importantes a se tratar é a ordenação das detecções. Como o modelo traz como resultado um *dataframe* onde as detecções ocorrem de forma paralela e são adicionadas fora de ordem, podemos ordená-las seguindo a ordem de leitura ocidental através das informações do posicionamento da área detectada, como apresentado no Código 7.

Código 7 – Código utilizado para ordenação das *labels* baseado na sua posição

```
1 def regra2(df):
2     # df -> DataFrame contendo resultado da deteccao
3     df_final = df.sort_values(by=['xmin', 'ymin'], ascending=[True,
          True]).reset_index(drop=True)
4     return df_final
```

Fonte: Elaborado pela autora

5.3.3.3 Regra 3 - Descarte de não preços

Como o modelo foi treinado para reconhecer objetos classificados como "outros", ou seja, informações presentes no recorte que não diz respeito ao valor de um produto, pode ocorrer de a imagem submetida para detecção não corresponda a uma imagem de preço, podendo ser uma imagem de descrição e até mesmo a imagem de um produto, como pode-se ver na Figura 21. Assim, como a detecção de objetos que não se encaixam como preços é irrelevante para a finalidade desta solução, é preciso descartar qualquer detecção que não retorne o valor real de um preço de um produto.

Figura 21 – Exemplo de recorte que não corresponde a um recorte de preço

Fonte: Elaborado pela autora

Assim, para o descarte de detecções de imagens que não correspondem a preços é feita uma contagem de detecções que correspondem à caracteres numéricos. Como preços são comumente apresentados em folhetos seguindo o formato de pelo menos um caractere de real, um separador decimal (vírgula) e dois caracteres de centavos, caso não sejam encontrados pelo menos 3 caracteres numéricos, essa detecção é descartada por não se tratar de um preço. Essa lógica foi utilizada para implementação da regra no Código8.

Código 8 – Código utilizado para validar se a imagem detectada representa ou não um preço

```
def regra3(df):
  # df -> DataFrame contendo resultado da deteccao
  number_labels = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
  drop_indexes = []
  numbers_count = 0

  for i, row in df.iterrows():
    class_name = row['name']
    if class_name in number_labels:
      numbers_count += 1

  if numbers_count < 3:
    for i in range(len(df)):
      drop_indexes.append(i)
  df_result = df.drop(df.index[drop_indexes])
  return df_result
```

Fonte: Elaborado pela autora

5.3.3.4 Regra 4 - Descarte de detecções com baixa confiabilidade

Em alguns casos, principalmente quando há números em meio a descrições que ocorrem em certos recortes de preço, detecções de caracteres numéricos são feitas e contabilizadas, como pode ser visto na Figura 22. Por tanto, foi criada uma regra para garantir o descarte de tais detecções. Nessa regra, é levado em conta o *score* de confiança de cada detecção, que consiste em um valor atribuído pelo modelo para indicar o quão certo daquela detecção ele está. Dessa forma podemos definir um valor mínimo para uma detecção ser aceita como um caractere de preço.

Figura 22 – Exemplo de detecções com baixa confiabilidade realizadas pelo modelo treinado

Fonte: Elaborado pela autora

Para realizar a checagem do *score* de confiança, foi implementada a regra presente no Código 9 onde foi estabelecido um limite de 66% de confiança após analisar alguns casos em imagens com resolução menores. Com isso, basta percorrer o *dataframe* e checar se o *score* de

confiança está abaixo do limite. Como foi implementada uma checagem simples para essa regra, em certo momento foi decido adicionar o descarte das detecções classificadas como "outros", por não serem relevantes para a leitura do preço, em ambos os casos, caso afirmativo, a detecção é descartada.

Código 9 – Código utilizado para descarte de detecções com confiabilidade baixa

```
def regra4(df):
    # df -> DataFrame contendo resultado da deteccao
    drop_indexes = []
    for i, row in df.iterrows():
        if (row['confidence'] < 0.66) or (row['name'] == 'outros'):
            drop_indexes.append(i)
    df_result = df.drop(df.index[drop_indexes])

    return df_final
```

Fonte: Elaborado pela autora

5.3.3.5 Regra 5 - Remoção de vírgulas duplicadas

Durante a detecção de valores de preços acima de 999 reais, o modelo apresentou resultados onde o ponto de marcação de unidade de milhar pode ser confundido e categorizado como o separador entre reais e centavos, sendo categorizado com a classe ".". Por conta disso, as detecções continham separadores decimais duplicados. Para contornar este problema, foi criada uma regra demonstrada no Código10, que detecta a ocorrência de detecções classificadas como separadores decimais e, caso seja detectado mais de um caractere da classe ".", mantém apenas a detecção que tiver maior score de confiança e que se encontre mais a direita entre os números que representem reais e centavos.

Código 10 – Código utilizado para remoção de detecções classes "."extras

```
def regra5(df):
  # df -> DataFrame contendo resultado da deteccao
  numeros = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
  separadores = ['.']
  separadores_index = []
  index_menor_confianca = 0
  drop_indexes = []

  menor_confianca = 10
  for i, row in df.iterrows():
    if row['name'] in separadores:
      separadores_index.append(i)
      if row['confidence'] < menor_confianca:
        menor_confianca = row['confidence']
        index_menor_confianca = i

  if len(separadores_index) > 1:
      if separadores_index[0] == index_menor_confianca:
      drop_indexes.append(separadores_index[0])
    else:
      drop_indexes.append(index_menor_confianca)

  df_result = df
  if len(drop_indexes) > 0:
      df_result = df.drop(df.index[drop_indexes])
  return df_final
```

Fonte: Elaborado pela autora

5.3.3.6 Regra 6 - Formatação de preço

Por fim, a regra semântica apresentada no Código 11 foi implementada para tratar os casos em que não foi identificado um caractere da classe ".", dessa forma aplicando uma formatação ao resultado resultando em um número com 2 casas decimais. Para isso, foi realizada a distinção de quais caracteres se referem ao valores de reais e quais se referem a valores de centavos baseados em suas posições e na posição ou suposta posição do caractere de separação ".". Ao aplicar essa regra, pode-se verificar se o resultado final da detecção obedece ao padrão de preço, tendo pelo menos um caractere de real, um separador decimal e dois caracteres de centavos, podendo ser descartada caso esse padrão não seja obedecido.

Código 11 – Código utilizado para inserção de separador decimal

```
def regra6(df):
  # df -> DataFrame contendo resultado da deteccao
  numeros = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "."]
  indexes = []
  have_dot = False
  for i, row in df.iterrows():
    indexes.append(i)
    if row['name'] == '.':
        have_dot = True
    if row['name'] in others:
      other_index.append(i)
  if have_dot:
    df_result = df
  else:
    dot_index = -2
    # adicao de linha no dataframe contendo a classe "."
    line = pd.DataFrame({"xmin": 0.0, "ymin": 0.0, "xmax": 0.0,
       "ymax": 0.0, "class": 1, "name": "."}, index=[dot_index])
    if len(df) > dot_index and (not df.empty):
      # insere o ponto e ajusta os indexes
      df_result = pd.concat([df.iloc[:dot_index], line,
         df.iloc[dot_index:]]).reset_index(drop=True)
    else:
      df_result = df
  return df_final
```

Fonte: Elaborado pela autora

5.3.4 Resultado da extração

As regras descritas são aplicadas em sequência no resultado da detecção do modelo, e por fim o *dataframe* que contem o resultado final tem as informações de *labels* coletadas e salvas em um arquivo de texto associado a imagem de entrada. Tal conversão foi desenvolvida através do Código 12

Código 12 – Código utilizado para inserção de separador decimal

```
def extract_results(df, image):
  # df -> DataFrame contendo resultado da deteccao
  # image -> nome do arquivo de entrada sem extensao
  preco = ""
  for index, row in df.iterrows():
    if row['name'] != 'R$':
      preco += row['name']

  with open('\output\' + image + '.txt', 'w') as file:
    file.write(preco)
    file.close()
```

Fonte: Elaborado pela autora

5.4 Testes preliminares

Após aprimorar com o uso das regras semânticas o resultado da detecção de caracteres realizado pelo modelo desenvolvido, uma seção de testes foi realizada com a finalidade de observar os erros e acertos cometidos pelo modelo. Assim, pode-se encontrar casos especiais os quais possam ser tratados com a implementação de novas regras semânticas ou até mesmo submetendo o modelo à um novo treino.

Para a realização dos testes do modelo, primeiramente foram selecionadas 2000 imagens de recortes de preços, coletadas da mesma maneira que foram coletadas as imagens para treino. Essas 2000 imagens tiveram o seu conteúdo anotado manualmente em uma tabela[7] no *Google Sheets*, onde todos os resultados dos testes realizados foram documentados. Enquanto todas as imagens foram submetidas ao modelo para realizar a detecção, seus resultados foram salvos na mesma tabela lado a lado com a respectiva anotação feita manualmente. Desse modo podemos realizar uma validação dos resultados propostos pelo modelo.

Como resultado, dentre as 2000 imagens analisadas, o modelo executou 1826 detecções corretas e 174 detecções incorretas, configurando em uma taxa de acerto de 91,3%. Mesmo com uma taxa de acerto consideravelmente alta, é preciso analisar os casos de erro em busca de formas de contornar os erros cometidos aumentando consequentemente a taxa de acerto.

5.4.1 Análise de resultados preliminares

Analisando os resultados adquiridos no teste anterior, pode-se perceber um padrão dentre as 174 imagens onde o modelo não conseguiu realizar uma detecção assertiva. Dentre essas imagens foi possível observar alguns padrões que foram agrupados como:

[7] Disponível em: https://bit.ly/3MVLUyS

- Baixa resolução: imagens com a resolução menor do que as utilizadas no *dataset* de treino;
- Preços parcelados: imagens que não representavam o valor total de um produto e deveriam ser descartadas;
- Outros: outros erros que não apresentaram semelhanças entre si;

A contabilização desses erros pode ser encontrada na Tabela 3. Esses erros podem ser contornados através da submissão do modelo à um retreino, ampliando seu *dataset* de treino com imagens de baixa resolução, para que o modelo possa aprender a identificar os caracteres nesse tipo de imagem. Já para as imagens de preço parcelado, é preciso adicionar um identificador ao modelo para esse tipo de imagem para que o modelo possa distinguir com facilidade e realizar o descarte apropriadamente.

Tabela 3 – Classificação e contabilização de tipos de erro cometidos pelo modelo

Tipos de imagens	**Quantidade de detecções**
Imgens de baixa resolução	134
Imagens de preço parcelados	10
imagens sem padrão identificado	30

5.5 Refinamento do modelo

Afim de que o resultado final do modelo seja aprimorado, o modelo foi submetido a um retreino e, para isso, foi necessário realizar uma nova coleta de material e rotulação para um novo *dataset*. Essa nova rotulação teve como foco imagens de baixa resolução e imagens de preços parcelados. Nesse segundo caso, foi incluído uma nova classe para representar o símbolo que indica o parcelamento de um preço, o "X". Essa nova classe foi adicionada tanto na ferramenta de rotulação, como nas configurações do modelo que, a partir de então, será treinado para reconhecer e classificar 14 classes. A seguir, serão descritos os passos seguidos para realizar o retreino do modelo.

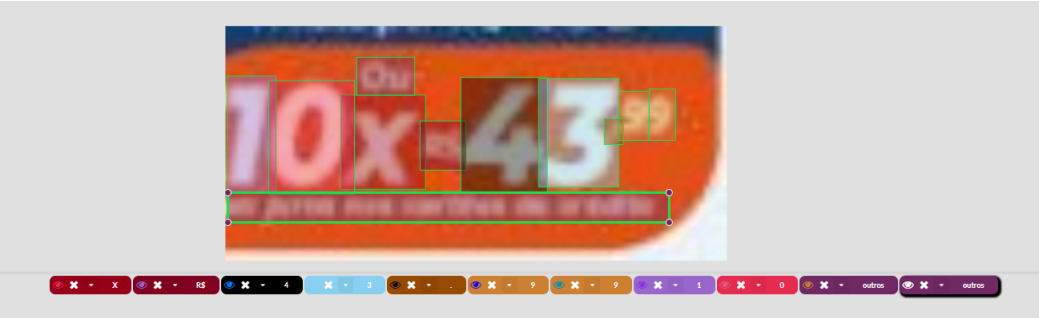
5.5.1 Coleta, seleção e rotulação de imagens

Para o retreino do modelo, foram coletadas as imagens utilizadas na Seção 5.4 que se encaixavam nos casos de erro de imagens de baixa resolução e preços parcelados. Além disso, foi executado uma segunda vez o modelo descrito em Barbosa (2022), com o intuito de arrecadar mais uma gama de amostras de imagens que se encaixem nos casos já citados. O modelo foi configurado para coletar dados de folhetos publicados entre os meses de janeiro e abril com o intuito de conseguir dados mais atuais para a nova rotulação e testes, resultando na coleta de 1586 novas imagens. Após a coleta, foram selecionadas as 144 imagens utilizadas nos testes já citados e mais 877 imagens dentre as conseguidas através do uso do modelo de coleta de recortes,

resultando em um total de 1021 imagens dentro do padrão desejado para a construção do novo *dataset*.

Após reunir todas as imagens para o novo *dataset*, um novo projeto foi criado no *Dataturks* com as novas imagens e seguindo as mesmas configurações e esquema de execução, com a diferença de que uma nova classe foi inserida para identificar e classificar o símbolo do parcelamento comumente utilizado na forma da letra "X". Em seguida, as imagens foram submetidas ao processo de rotulação e por fim conversão, descritos respectivamente em 5.2.2 e 5.2.3, resultado em imagens rotuladas como as da Figura 23.

Figura 23 – Exemplo de rotulação com a nova classe X



Fonte: Elaborado pela autora

5.5.2 Novas configurações e retreino do modelo

Após a preparação das imagens, um *dataset* foi montado tendo como conjunto de treino 921 imagens e suas respectivas *labels* e como conjunto de validação 100 imagens e suas *labels*. Com isso, o modelo foi reconfigurado para se adequar aos novos parâmetros do *dataset*. Primeiramente o número de classes foi alterado para 14, enquanto a nova classe "X"foi inserida no final da lista de classes, presentes respectivamente na cópia dos arquivos "yolov7.yaml"e "coco.yaml"criadas anteriormente. Ainda na cópia do arquivo "coco.yaml", foram ajustado os diretórios de treino e validação para os arquivos do novo *dataset*.

Além disso, uma nova regra semântica foi adicionada ao conjunto de regras que são aplicadas no pós-processamento. Essa regra é responsável por identificar se a detecção possui a nova classe adicionada ao *dataset*, o "X", e se caso existir essa classe, é checado se seu *score* de confiança é maior do que 70%. Caso afirmativo, aquela detecção é descartada pois o recorte se trata de um preço parcelado, como pode ser visto na implementação do Código 13. Esta nova regra é aplicada logo após a Regra 1 descrita em 5.3.3.1.

Com a configuração do modelo pronta e a nova regra adicionada, foi executado a rodada de treino tendo como base o peso do modelo que esta sendo desenvolvido, executando o treino em um total de 30 *epochs*, mantendo os demais parâmetros utilizados durante o primeiro treino. Após submeter o modelo ao retreino, ele apresentou as métricas exibidas na Tabela 4.

Código 13 – Código utilizado para verificação de preço parcelado

```
def regraDescartarPrecoParcelado(df):
  # df -> DataFrame contendo resultado da deteccao
  drop_indexes = []
  drop = False

  for i, row in df.iterrows():
    if (row['name'] == "X") and (row['confidence'] > 0.7):
      drop = True

  if drop:
      for i in range(len(df)):
        drop_indexes.append(i)

  df_result = df.drop(df.index[drop_indexes])
  return df_result
```

Fonte: Elaborado pela autora

Tabela 4 – Métricas do retreino do modelo

Classes	***Precision***	***Recall***	***mAP*@0.5**	***mAP*@0.5:0.95**
all	99,1%	99,6%	99,3%	79,1%
R$	98,6%	99,8%	99,4%	74,0%
6	99,4%	99,9%	99,5%	82,3%
.	99,2%	99,5%	99,4%	59,5%
9	99,6%	100%	99,6%	80,1%
5	99,6%	100%	99,5%	83,5%
outros	95,7%	96,2%	98,1%	70,9%
8	100%	99,7%	99,5%	81,9%
4	99,8%	100%	99,5%	84,0%
3	99,4%	100%	99,2%	81,5%
2	99,9%	99,7%	99,5%	84,9%
7	99,8%	100%	99,5%	83,8%
0	97,7%	99.5%	98,5%	79,6%
1	99,8%	99,7%	99,5%	82,1%
X	99,1%	100%	99,5%	79,9%

Fazendo um comparativo entre as Tabelas 2 e 4, percebemos um aumento significativo nas porcentagens de cada métrica, o que indica que atingimos o objetivo de aprimorar o nosso modelo para a detecção de preços. Assim, submetemos o modelo a mais algumas rodadas de teste a fim de observar a ocorrência dessa melhoria e tentar encontrar mais casos de erro que possam ser contornados.

5.6 Testes preliminares com refinamento

Para finalizar a etapa de desenvolvimento, foi reunido um conjunto de teste formado imagens coletadas através do modelo descrito em Barbosa (2022) configurado para coletar

folhetos publicados no mês de março, afim de evitar o uso de recortes repetidos durante os testes. Dessa forma o modelo retornou um conjunto de 524 recortes de preço, das quais 500 foram selecionadas para serem aplicadas como teste.

O processo de aplicação do teste foi o mesmo aplicado em 5.4, resultando em 498 casos de acerto e 2 casos de erro, atingindo uma taxa de 99,6% de acerto. Enquanto isso, analisando os casos de erro percebeu-se que as imagens de entrada se referiam a imagens de preços que tiveram um desconto aplicado, como pode ser visto na Figura 24. Nesses casos a detecção deveria ser descartada por não se tratar do preço real do produto, mas o modelo identificou os caracteres corretamente e considerou aquele recorte como o preço real do produto.

Figura 24 – Exemplo de recorte de preço com desconto

Fonte: Elaborado pela autora

Como a característica principal que indica um preço com desconto é a presença de um risco sob o preço, seria necessário uma grande quantidade de imagens com esse aspecto para que o modelo fosse submetido a outro treino e começasse a reconhecer esse padrão, o que levaria muito tempo de busca para montagem do *dataset* de ajustes para o aperfeiçoamento do modelo, podendo até mesmo interferir na detecção de outras classes. Sendo assim, essa versão do modelo sendo a com o melhor refinamento.

6

Análise comparativa

A análise comparativa das soluções desenvolvidas durante a realização desse trabalho é essencial. Com ela podemos identificar com facilidade a abordagem mais eficiente para solucionar a problemática abordada por esse trabalho. Nesse contexto, este capítulo descreve o teste realizado para obter os resultados das extrações das soluções desenvolvidas, a comparação realizada com os resultados esperados e, a partir dessa comparação, o desempenho de cada solução.

6.1 Preparação para o teste

Antes de realizar o teste, foram coletadas, através através da solução desenvolvida em Barbosa (2022), novos 1000 recortes de preço. Esses foram selecionados a fim de cobrir as diversas ocorrências de padrões de possíveis entradas para as soluções. Ou seja, dentre os 1000 recortes, estão presentes imagens com baixa e alta qualidade, preços parcelados, preços com descontos e recortes que não correspondem a preços. Esses recortes serão utilizados como objeto de detecção tanto da primeira solução, quanto das duas versões, com e sem refinamento, da segunda solução. Dessa forma, para cada recorte de imagem, haverá três resultados de extração, sendo cada um correspondente a uma solução. A fim de facilitar a compreensão dessa etapa, as soluções foram nomeadas como: Solução com ferramenta de *OCR*, correspondendo a primeira solução descrita no capítulo 4; Solução com *deep learning* sem refinamento, correspondendo a primeira versão da segunda solução descrita na seção 5.3; Solução com *deep learning* com refinamento, correspondendo a segunda versão da segunda solução descrita na seção 5.5.

Para a realização do teste, foi desenvolvido um *script* na linguagem de programação *python*, que pode ser visto no código 14, onde as imagens de entrada passam pelas três soluções e seus resultados são adicionados a uma tabela[1] no *Google Sheets*, a fim de facilitar a comparação.

[1] Disponível em: https://bit.ly/3MVLUyS

Código 14 – Código utilizado para aplicação do teste comparativo

```
def detect(model_path, path_weight_sr, path_weight_cr,
   detection_path):
    # resultados de extracao
    results = []
    # modelo yolov7 sem refinamento
    model_sem_refinamento = loadModel(model_path=model_path,
       path_weight=path_weight_sr)
    # modelo yolov7 com refinamento
    model_com_refinamento = loadModel(model_path=model_path,
       path_weight=path_weight_cr)

    for image in os.listdir(detection_path):
        print(image)
        path = f'{detection_path}/{image}'
        # solucao deep learning sem refinamento
        result_sem_refinamento = model_sem_refinamento(path)
        # solucao deep learning com refinamento
        result_com_refinamento = model_com_refinamento(path)
        # solucao com ferramenta OCR
        tesseract = ocrBinarizado(path)

        df_result_sr = aplicar_regras_sem_refinamento(
                        result_sem_refinamento.pandas().xyxy[0]
                        )
        df_result_cr = aplicar_regras_com_refinamento(
                        result_com_refinamento.pandas().xyxy[0]
                        )
        result = convert_results_to_table(tesseract, df_result_sr,
           df_result_cr, path, index)
        results.append(result)

    df = pd.DataFrame(data=results)
    df.to_excel('results.xlsx', index=False)
```

Fonte: Elaborado pela autora

6.2 Aplicação do teste e análise de resultados

Após preparar o ambiente de execução do teste, as 1000 imagens foram submetidas e, após a execução, foram contabilizada as métricas para análise comparativa. Para essa contagem, todas as imagens tiveram o seu valor real anotado e esse valor foi comparada com cada resultado da solução. Dessa forma, se a solução conseguisse identificar o valor do preço corretamente, ela contabiliza um ponto de acerto, caso contrário, receberia um ponto de erro e por fim teria a sua acurácia calculada. As pontuações de acerto, erro e porcentagem de acurácia de cada solução podem ser observadas na tabela 5.

Tabela 5 – Métricas do resultado do teste comparativo

Solução	**Acerto**	**Erro**	**Acurácia (%)**
Solução com ferramenta de *OCR*	231	769	23,1%
Solução com *deep learning* sem refinamento	344	656	34,4%
Solução com *deep learning* com refinamento	911	9	99,1%

Analisando os resultados obtidos no teste, percebemos que a solução com *deep learning* com refinamento foi a que apresentou uma melhor assertividade. Com uma acurácia de 99,1%, errando apenas 9 detecções dentre as 1000 realizadas, a solução conseguiu se sobressair comparada as outras propostas. O segundo melhor desempenho foi a solução com *deep learning* sem refinamento, com uma acurácia de 34,4% seguido pela solução com ferramenta de *OCR* com acurácia de 23,1%.

Como analisado durante os testes preliminares de cada solução, as soluções com ferramenta de *OCR* e com *deep learning* sem refinamento não conseguiam cobrir os casos de imagens com baixa resolução e realizar o descarte de preços parcelados, sendo a primeira a que mais cometeu erros na identificação de números nas imagens. Já a solução com *deep learning* com refinamento, conseguiu distinguir corretamente esses casos em que as outras soluções falharam, realizando corretamente os descartes quando preciso, falhando apenas em alguns casos de preços com desconto e cometendo algumas detecções errôneas em casos não específicos. Sendo assim, a solução com *deep learning* com refinamento tem se provado, dentre as soluções propostas, ser uma solução adequada para o problema proposto por esse trabalho, tendo a menor contagem de erros e, por consequência, a maior acurácia.

7

Considerações Finais

O projeto apresentado neste documento surgiu com a finalidade de incrementar o *LudiiPrice* com a funcionalidade de extração de caracteres de preço de produtos presentes em folhetos de supermercados. Para isso, foram elaboradas possíveis soluções baseadas em técnicas de processamento de imagem e *deep learning* para reconhecer e detectar caracteres numéricos e especiais relevantes para a leitura de preços em recortes de folhetos.

Através da utilização de desenvolvimentos prévios relacionados ao *LudiiPrice*, foi possível coletar imagens de recortes de preço para a construção de *datasets*. Esses *datasets* serviram de base para o desenvolvimento e testes das soluções desenvolvidas nesse trabalho. A primeira solução desenvolvida consiste no uso da ferramenta de detecção de caracteres *Tesseract-OCR*, junto com técnicas de pré-processamento de imagens, com o intuito aprimorar os resultados da ferramenta. Já a segunda solução proposta foi composta de uma primeira versão, onde um modelo de *CNN* foi desenvolvido, e uma segunda versão onde foi realizado seu aprimoramento, junto da aplicação de regras semânticas para a formatação e validação dos resultados de extração. Para essa solução, o *YOLOv7* foi treinado para reconhecer os caracteres presentes nos recortes de preço, e na segunda versão, foi aplicado um retreino nesse modelo focando nos casos de falha em detecções feitas durante testes realizados na primeira versão, com o intuito de refinar e aprimorar o modelo desenvolvido. Para o desenvolvimento das versões baseadas em *deep learning*, foram destinadas 1000 imagens para o treino da primeira versão e mais 992 imagens para o desenvolvimento do seu aprimoramento, resultando na segunda versão.

Os passos seguintes ao desenvolvimento das soluções consistem em uma análise comparativa. A primeira solução e as duas versões da segunda solução, com e sem refinamento, foram submetidas a um teste com 1000 imagens. Nesse teste, foi contabilizado os acertos e erros de cada solução. Assim após a contabilização, a primeira solução baseada em ferramenta de *OCR* alcançou uma acurácia de 23,1%, a segunda solução baseada em *deep learning* alcançou uma acurácia de 34,4% e, por fim, a segunda solução após passar pelo refinamento, alcançou uma

acurácia de 99,1%, sendo essa última a que se mostrou mais assertiva e confiável para a solução da problemática de detecção de caracteres de preço em recorte de folhetos de supermercado.

Apesar dos resultados apresentados pela solução proposta por este trabalho para a extração de preços de produtos de imagens terem sidos satisfatórios, ainda é preciso explorar a detecção e extração de caracteres presentes em recortes de descrições de produtos em folhetos de supermercados. Pois, mesmo com várias ferramentas de extração de texto presentes no mercado, é preciso desenvolver futuramente um estudo que mantenha a assertividade e qualidade da extração nesse cenário. Tal estudo é necessário para a finalização da integração desse trabalho com a plataforma do *LudiiPrice*.

Referências

ALBUQUERQUE, M. P. d. A. Márcio Portes de. Processamento de imagens: Métodos e análises. *Centro Brasileiro de Pesquisas Físicas – CBPF/MCT*, 2010. Citado 2 vezes nas páginas 15 e 16.

AMID. *Improving the quality of the output*. 2021. Tesseract documentation. Disponível em: <https://tesseract-ocr.github.io/tessdoc/ImproveQuality.html>. Acesso em: 19 out 2022. Citado 5 vezes nas páginas 12, 15, 29, 33 e 34.

BARBOSA, M. de S. *Avaliação de técnicas para detecção de produtos e preços em folhetos digitais de supermercados*. São Cristóvão, 2022. Citado 9 vezes nas páginas 4, 10, 11, 12, 28, 39, 52, 54 e 56.

BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. Yolov7: An efficient and enhanced yolo for real-time object detection. *arXiv preprint arXiv:2106.09797*, 2021. Citado 2 vezes nas páginas 21 e 39.

DRUZHKOV, P.; KUSTIKOVA, V. A survey of deep learning methods and software tools for image classification and object detection. *Pattern Recognition and Image Analysis*, Springer, v. 26, n. 1, p. 9–15, 2016. Citado 2 vezes nas páginas 16 e 19.

GIRSHICK, R. Fast r-cnn. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. [S.l.: s.n.], 2015. Citado na página 19.

JOSEPH et al. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. Disponível em: <http://arxiv.org/abs/1506.02640>. Citado na página 21.

LAPTEV, P. et al. Neural network-based price tag data analysis. *Future Internet*, v. 14, n. 3, 2022. ISSN 1999-5903. Disponível em: <https://www.mdpi.com/1999-5903/14/3/88>. Citado 3 vezes nas páginas 24, 25 e 26.

MAZZETTO, M. Detecção e classificação de múltiplos componentes em linha de montagem automotiva usando deep learning. Pato Branco, PR, Brasil, 2019. Disponível em: <http://repositorio.utfpr.edu.br/jspui/handle/1/4607>. Citado na página 19.

MICROSOFT. *Overview - Optical Character Recognition (OCR) | Microsoft Azure*. 2021. <https://learn.microsoft.com/pt-br/azure/cognitive-services/computer-vision/overview-ocr>. Acesso em: 10 de maio de 2023. Citado na página 14.

MOSQUERA, H. P.; GENÇ, Y. Recognition and classifying sales flyers using semi-supervised learning. In: *2019 4th International Conference on Computer Science and Engineering (UBMK)*. [S.l.: s.n.], 2019. p. 1–6. Citado na página 10.

PASSOS, B. T. *O mundo do ponto de vista das Redes Neurais Convolucionais (RNCs)*. 2021. Disponível em: <https://ateliware.com/blog/redes-neurais-convolucionais>. Acesso em: 26 out. 2022. Citado na página 18.

SAHA, S. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. 2018. Disponível em: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>. Acesso em: 26 out. 2022. Citado na página 17.

SILVA, J. V. S. Atalaia: Uso de deep learning para reconhecimento automático de placas de licença automotiva em dispositivos com recursos limitados. São Cristóvão, SE, Brasil, ago. 2022. Disponível em: <http://ri.ufs.br/jspui/handle/riufs/16257>. Citado 5 vezes nas páginas 12, 26, 27, 36 e 37.

TOROK, L. Método de otsu. *Instituto de Computação (UFF)*, 2016. Citado na página 33.

ULLAH, R. et al. Ocr engine to extract food-items, prices, quantity, units from receipt images, heuristics rules based approach. fev. 2018. Citado 3 vezes nas páginas 10, 23 e 24.

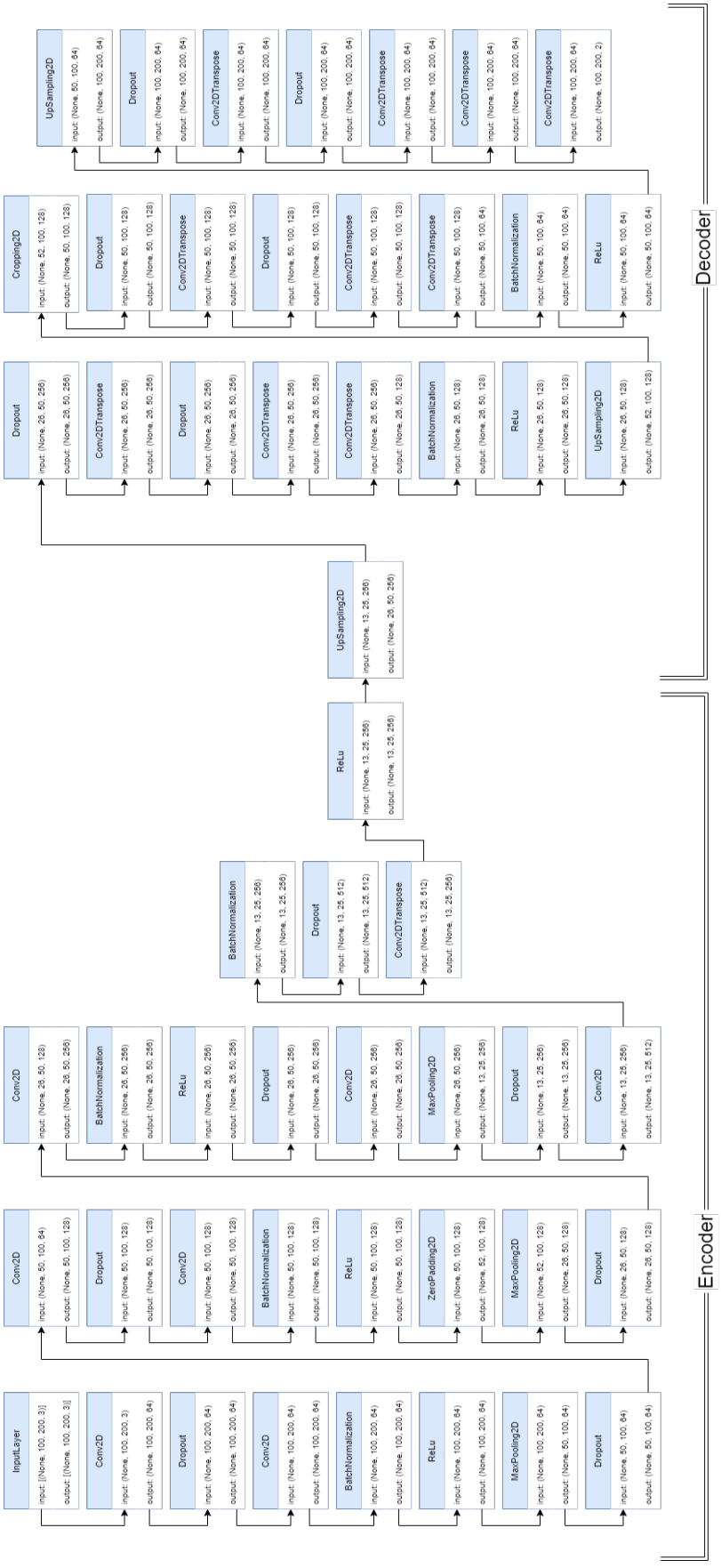
VARGAS, A. C. G.; PAES, A.; VASCONCELOS, C. N. Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres. In: SN. *Proceedings of the xxix conference on graphics, patterns and images*. [S.l.], 2016. v. 1, n. 4. Citado na página 17.

YUAN, T. et al. Robust cherry tomatoes detection algorithm in greenhouse scene based on ssd. *Agriculture*, v. 10, p. 160, 05 2020. Citado na página 20.

ZOU, Z. et al. Object detection in 20 years: A survey. 05 2019. Citado na página 18.

Apêndices

APÊNDICE A – Arquitetura final de *autoencoder*



Fonte: Elaborado pela autora