
Pedro Jorge Lima da Silva

# Faster R-CNN para extração de caracteres em hidrômetros

Universidade Federal de Pernambuco
www.cin.ufpe.br/~graduacao

Recife
2023

Pedro Jorge Lima da Silva

# Faster R-CNN para extração de caracteres em hidrômetros

Trabalho apresentado ao Programa de Graduação em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Engenharia da Computação.

***Área de Concentração***: *Inteligência Computacional*
***Orientador***: *Veronica Teichrieb (UFPE)*
***Co-Orientador***: *João Marcelo Xavier Natário Teixeira (UFPE)*

Recife
2023

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Silva, Pedro Jorge Lima da.
Faster R-CNN para extração de caracteres em hidrômetros / Pedro Jorge Lima da Silva. - Recife, 2023.
48 p. : il., tab.

Orientador(a): Verônica Teichrieb
Cooorientador(a): João Marcelo Xavier Natário Teixeira
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Engenharia da Computação - Bacharelado, 2023.

1. Redes Neurais Convolucionais. 2. OCR. 3. Deep Learning. 4. Visão Computacional. 5. Detecção de Objetos. I. Teichrieb, Verônica. (Orientação). II. Teixeira, João Marcelo Xavier Natário. (Coorientação). IV. Título.

000 CDD (22.ed.)

Inspirado pelos mestres Ariano Suassuna e Chico Science, como bom pernambucano, este trabalho é um trabalho de resistência e de luta. Pela pesquisa, pela ciência e pela inovação. Dedico este trabalho a todos e todas que lutam para permitir que mais pessoas possam entrar pelas portas da robótica e para que estas permaneçam criando um mundo melhor.

*"Ciência dentro da gaveta ou dentro da cabeça não ajuda ninguém, não faz do mundo um lugar melhor." Eduardo Jorge Souza da Silva.*

# AGRADECIMENTOS

Ainda que esta página fosse infinita, seria impossível agradecer nominalmente a todos e todas que de certa forma colaboraram na caminhada para que eu chegasse até este momento.

Mas não posso deixar de agradecer a todas as forças enviadas a mim durante todos esses anos, as boas e as ruins. Afinal, todas elas me forjaram. Também agradeço aos santos e deuses, a quem muitas vezes recorri pedindo por disposição e equilíbrio para perseverar na caminhada. Nesse mesmo passo, não posso deixar de agradecer a minha mãe Cristina que por muitas vezes foi vitima dos meus estresses da faculdade e também não posso deixar de pedir desculpas por isso. Ao meu pai Eduardo, que foi minha referência como pesquisador e estudante, lembrando daquelas madrugadas que o senhor ficava acordado para terminar o mestrado e todas as montanhas de livros lidas. E claro àquela que está ao meu lado nesta caminhada desde o primeiro dia de aula da faculdade, que nunca largou minha mão e é a maior parceira das minhas loucuras dos últimos anos, Júlia.

Também não posso deixar de agradecer ao mestres que vieram anteriormente nesta caminhada, em especial a Vancleide Jordão, professora magnifica e também uma mãe, não apenas minha mais de várias outras pessoas. Marco histórico da robótica em Pernambuco. Se não fosse por ela, certamente esse estado não teria pessoas que são referência em robótica e tantos títulos em competições. Aproveito para agradecer ao pessoal do Petros, Rodrigo, Lice, Bia e Lalá. Amigos que começaram na robótica e me inspiram a continuar neste mundo para que outras pessoas possam ser que nem eles, GIGANTES. Aos amigos do Whatever, Alice, Bruna, Dudu e Ruben que participaram desta caminhada desde 2008. Aos amigos do "Aninha Morreu", Heitor, Arlindo e Rafinha, minha panelinha querida <3.

Aos amigos que construí no Maracatronics ou que reencontrei lá. Dani, Pinho, Gustavo, Thiago, Gabo, Caio, Joaquim e Zildão, e também a todos os outros que fizeram tornar realidade o Projeto Armorial. Autônomos Coletivos é o melhor projeto sim! Aos amigos da robótica que encontro anualmente no melhor evento de robótica do mundo: a Robocup Brazil Open, ou LARC, para íntimos. Adam & Lang pela confiança e permitir que 12 anos depois da minha primeira LARC eu pudesse voltar ao mesmo lugar mas como Chair. Tatiana Pazelli, Dani Ortiz, Cinthia Aihara, Sarah Thomaz, Jussara, Henrique Foresti e João Paulo por acreditarem e confiarem em mim e no Maraca como coordenadores e operadores da OBR em Pernambuco.

Aos amigos do Voxar, em especial ao grande BOSS João Marcelo e também VT, que confiaram no meu potencial e me levaram para lá e a Maria, por todas as horas de escuta e conselhos para melhoria. Este trabalho também não poderia ser feito sem o apoio incondicional de todos que fazem a InSpace neste momento! Tem muito suor de toda a equipe, guga, jorjão e vic, nestas páginas.

E vamos começar este trabalho!

# ABSTRACT

The analysis of water consumption in residential condominiums is crucial to detect leaks and optimize water usage. Although solutions based on the Internet of Things are widely used, some users still have doubts about the veracity of the collected data. To overcome this problem, a solution using OCR based on object detection with Deep Learning was implemented. Preliminary tests revealed that the OCR solutions from Amazon and Tesseract did not achieve relevant accuracy due to the peculiarities of the water meters. Therefore, a dataset was created with real images of hydrometers from two buildings in Recife and 22 models based on Faster R-CNN were trained and compared for the digit recognition task. Then, the model with the highest mAP was chosen for the implementation of the model-based solution, which achieved an accuracy greater than 90% when evaluating the first 5 digits of the hydrometer.

**Keywords:** OCR, Object Detection, Deep Learning, Computer Vision

# RESUMO

A análise do consumo hídrico em condomínios residenciais é crucial para detectar vazamentos e otimizar o uso da água. Embora soluções baseadas em Internet das Coisas sejam amplamente utilizadas, alguns usuários ainda têm dúvidas sobre a veracidade dos dados coletados. Para superar esse problema, foi implementada uma solução usando Reconhecimento ótico de Caracteres (OCR) baseado em detecção de objetos com Aprendizagem Profunda. Testes preliminares revelaram que as soluções de referência no mercado não atingiram precisão necessária devido às peculiaridades encontradas dos hidrômetros. Portanto, foi criado um conjunto de dados com imagens reais de hidrômetros de dois prédios em Recife, foram treinados 22 modelos baseados na Faster R-CNN e comparados para a tarefa de reconhecimento de dígitos. Em seguida, o modelo com maior mAP foi escolhido para implantação da solução, alcançando uma precisão superior a 90% ao avaliar os 5 primeiros dígitos do hidrômetro.

**Palavras-chave:** OCR, Detecção de Objeto, Aprendizagem Profunda, Visão Computacional

# LISTA DE FIGURAS

# LISTA DE TABELAS

# SUMÁRIO

# 1

# INTRODUÇÃO

Segundo o Instituto Trata Brasil, no Brasil, em 2021 40% da água potável captada foi desperdiçada [3]. A partir da série histórica do mesmo estudo também é possível constatar que existiu um aumento dessa porcentagem desde 2017 e, em 2019, foram perdidos aproximadamente 39% da água potável captada, o equivalente a 7,5 mil piscinas olímpicas. Já em Pernambuco, as perdas chegaram a 45% em 2022 [12]. Além disso, segundo estudos recentes da UNESCO [10], cerca de 2,5 bilhões de pessoas sofrem com a escassez de água por pelo menos um mês do ano, podendo ser agravada nos próximos anos com o aumento da população. Ao fazer o recorte para a população urbana global que já sofre com falta de água, este mesmo estudo alerta que este grupo pode crescer de 930 milhões de pessoas em 2016 para até 2,4 bilhões em 2050.

Diante das numerosas perdas e do crescente número de pessoas afetadas pela escassez de água, as Nações Unidas estabeleceram o Objetivo de Desenvolvimento Sustentável 6 - Água potável e saneamento [11]. Esse objetivo tem como propósito fundamental garantir a disponibilidade e promover a gestão sustentável da água potável e do saneamento para todos.

Assim, tendo em vista a gestão sustentável, gestoras de condomínios têm procurado soluções que as permitam monitorar o consumo de água das áreas comuns e também das unidades residenciais com o intuito de acompanhar o consumo hídrico e registrar a série histórica. Assim, conseguindo identificar possíveis vazamentos e picos de consumo.

Atualmente, para fazer o registro do consumo de água são utilizados hidrômetros de diversos modelos, padronizados e fiscalizados. Para realizar a leitura dos equipamentos, empresas lançam mão de diversas estratégias como time de anotadores, onde pessoas passam uma vez por mês de casa em casa para coletar os dados. Esta estratégia é comum para empresas de saneamento nas cidades, como a Compesa, no estado de Pernambuco.

Entretanto, esta estratégia é lenta e cara para administradoras de condomínio e para quem deseja fazer uma coleta com maior frequência. Assim, outras duas opções são disponibilizadas no mercado, sendo a mais comum delas a obtenção dos dados através de dispositivos IoT acoplados em hidrômetros, como a solução das empresas Techmetria[1] e Aquameter[2], que realizam a leitura através de sensores magnéticos e enviam os dados usando radio-frequência, redes WiFi ou

[1] `http://www.techmetria.com.br/`

[2] `https://www.acquameter.eng.br/leitura-remota-por-telemetria/`

redes móveis como 3G/4G/5G. A segunda opção, chamada de foto-medição, utiliza imagens obtidas através de câmeras de celular ou acopladas ao hidrômetro para capturar o valor do relógio, realizar o reconhecimento ótico de caracteres (OCR) e enviá-los para uma base de dados [1]. A foto medição com OCR se torna ainda mais atrativa já que não onera a residência ou estabelecimento comercial com um equipamento especifico.

Com o intuito de desenvolver uma funcionalidade de leitura de hidrômetros por OCR para o seu aplicativo, uma prestadora de serviços para administradoras de condomínios em Recife contratou a InSpace[3], startup fundada e localizada no Centro de Informática da Universidade Federal de Pernambuco. Observando os requisitos da aplicação que serão aprofundados futuramente, primeiramente foram realizados testes usando o sistema de OCR da Amazon, Textract. Também procedeu-se o desenvolvimento de uma solução específica para o caso baseada em redes neurais convolucionais. Foi feito um breve levantamento das possibilidades contemplando modelos bem estabelecidos na área para detecção de objetos como Yolo V8 [26] e Faster R-CNN [35].

Devido à facilidade de treinamento, robustez nas detecções e a permissividade do seu código, o modelo escolhido foi o Faster R-CNN, provido pelo framework de *Machine Learning* PyTorch [33]. O modelo foi treinado em um dataset coletado pela InSpace em conjunto com a empresa contratante, contendo originalmente 226 imagens e 534 imagens após a realização de *Data Augmentation*.

O modelo obtido foi testado em um conjunto de 69 imagens e atingiu 92% de assertividade[4] na leitura do valor do hidrômetro. Além disso, foi realizado o *deployment* do modelo para uma função Lambda na Amazon Web Services (AWS) e o mesmo já foi testado em diferentes modelos de hidrômetro pela empresa contratante, com acurácia semelhante aos testes realizados.

Como contribuições à comunidade, este trabalho apresenta os seguintes resultados:

- Análise de diferentes extratores de características (backbones) aplicados à Faster R-CNN para reconhecimento de caracteres em hidrômetros;
- Dataset contendo imagens anotadas de hidrômetros;
- Módulo para extração de valores em hidrômetros baseado na Faster R-CNN.

Por fim, este documento se organiza da seguinte forma: No Capítulo 2 são apresentados trabalhos relacionados a esta pesquisa, no Capítulo 3 é apresentado o cenário da aplicação a ser desenvolvida bem como possibilidades e limitações impostas pelo cenário e pela contratante, Capítulo 5 apresenta a metodologia dos testes realizados, Capítulo **??** apresenta o dataset utilizado e suas características, Capítulo 4 é apresentada a proposta de solução, Capítulo 6 apresenta os resultados obtidos nos testes, e por fim, o Capítulo 7 apresenta as conclusões obtidas com base nos resultados.

[3] O autor deste manuscrito é sócio-fundador e CTO da InSpace

[4] Neste manuscrito as palavras “assertividade"e “acurácia"serão utilizadas com o mesmo significado

# 2

# TRABALHOS RELACIONADOS

Entender a disposição de objetos em imagens é tarefa essencial e cada vez mais requisitada dentro da visão computacional e processamento de imagens. Ao longo dos anos, muitos métodos foram propostos visando classificar, detectar ou segmentar objetos específicos dentro de imagens. Muitos desses métodos são abordagens simples, passando por segmentação por cores e identificação de contornos. Outros métodos com maior grau de complexidade visam atingir uma maior robustez, extraindo características significativas dos objetos que possam representá-los independente da sua cor, posição ou escala na imagem.

Nesta revisão de literatura estão artigos resultantes de uma busca exploratória não exaustiva nas bases do Google Scholar, MDPI e IEEE Xplore. É possivel clasifica-los em artigos avaliando principais frameworks de OCR disponíveis, artigos relacionados a reconhecimento e detecção de números, e artigos relacionados à extração de números em hidrômetros. Foram utilizados combinação com os seguintes termos da língua inglesa:

- *OCR*
- *Water Meter/Hydrometer*
- *Number Detection*
- *Number Recognition*
- *Number Extraction*

Detre os trabalhos encontrados, Rahim *et al.* [34] conduziram uma revisão de literatura que mostra a importância da coleta dos dados de consumo de água para prevenir vazamentos e permitir o acompanhamento dos gastos pelos usuários, entre outros motivos também importantes. Entretanto, o uso de técnicas de aprendizagem de máquina, elencados no artigo,se restringem aos cenários de previsão de consumo ou detecção de anomalias na rede hídrica, como vazamentos. Nas seções a seguir serão abordados artigos e técnicas para fazer a coleta dos dados com técnicas diversas, que vão desde a utilização de OCR até mesmo o uso de redes neurais convolucionais que tratam o problema como uma detecção de objetos.

## 2.1 FRAMEWORKS DE OCR

É possível localizar um ponto de partida para as pesquisas relacionadas à identificação de números e da localização deles na imagem, a partir da visão de que será necessário construir uma palavra (um conjunto de símbolos, que aqui são algarismos) utilizando a detecção de vários algarismos únicos. Assim, foi feito um levantamento dos frameworks de OCR disponíveis a nível acadêmico e de mercado como Tesseract, EasyOCR, Amazon Textract/rekognition, Azure OCR e Google Cloud Vision.

O Tesseract OCR é um motor de reconhecimento de caracteres desenvolvido pela HP entre 1984 e 1994, em 1995 foi enviado para *UNLV Annual Test of OCR Accuracy* e acabou se provando como pior entre as alternativas disponíveis no mercado na época [39, 40]. Em 2005, seu código foi disponibilizado como open-source e se tornou um dos motores de OCR mais populares e com melhorias feitas pela comunidade. Atualmente, conta com uma interface em Python que permite a desenvolvedores integrá-lo facilmente a diversas aplicações.

O EasyOCR[1] também é um motor para reconhecimento de caracteres, porém este é baseado em redes convolucionais implementadas usando Pytorch, o que permite a execução com ganhos de velocidade quando usada a GPU. O EasyOCR usa ResNet e VGG para fazer a extração de features, LSTM [21] para montar a sequência de rotulagem e Classificação Temporal Conexionista [20] para decodificação. Este OCR oferece a possibilidade de execução local com um módulo em Python e possui um serviço em nuvem oferecido como SaaS.

Também como SaaS ou para operação em Cloud, outros 3 serviços de OCR se apresentam como possibilidades, são eles: o Amazon Textract/Rekognition, o Azure Vision OCR e o Google Cloud Vision. O Amazon Textract/Rekognition [2] é um serviço de OCR baseado em aprendizagem de máquina com custo de US$ 1,50 a cada 1000 páginas, com 1000 páginas gratuitas no mês e com suporte para 6 línguas. Este serviço permite extrair textos planos, manuscritos e tabulares. O Google Cloud Vision [3] segue na mesma linha do anterior, um serviço em nuvem com API, com custo semelhante porém com suporte para 60 línguas distintas. Com suporte para mais de 160 línguas para textos digitados, o serviço da Azure para OCR também segue na mesma linha dos anteriores com suporte para textos manuscritos e digitados com custos em torno de US$ 1.00 a cada 1000 páginas. Diversos estudos como o de Hegghammer [22] realizaram comparações entre sistemas cloud. Já outros estudos como o de Spichkova *et al.* fazem comparações de sistema em cloud com frameworks como Tesseract e EasyOCR [41, 47].

Todas essas tecnologias já foram aplicadas em diversos cenários como aplicações como reconhecimento de placas de carro [4], Leitura de medidores de eletricidade e gás [41], indexação de documentos [24], tradução de documentos [45], digitalização de notas fiscais [37] e até mesmo para rasteamento de alimentos [51] e processamento de linguagem natural [25].

[1] `https://github.com/JaidedAI/EasyOCR`
[2] `https://aws.amazon.com/pt/textract/`
[3] `https://cloud.google.com/vision/docs?hl=pt-br`

## 2.2 DETECÇÃO DE OBJETOS E REDES CONVOLUCIONAIS

Além de tratar os números a serem identificados como palavras, é possível visualizar este problema como um caso de detecção de objetos, onde é necessário localizar e classificar cada número como um objeto diferente. Existem hoje diversos datasets para detecções de objetos, alguns focados em objetos específicos como placas de carro e outros famosos com o escopo mais aberto e generalista. Cabe aqui destacar principalmente 3 sendo eles o PascalVOC [16], o ImageNet [13, 36] e o COCO Dataset [29].

Ao reduzir o escopo para classificação e detecção de algarismos, um dos mais famosos datasets é o MNIST [14]. Este dataset possui 70 mil amostras divididas em 60 mil amostras para treinamento e 10 mil para teste com imagens de 28x28 pixels contendo números escritos a mão. Uma versão estendida deste dataset, chamado EMNIST [9], foi publicada em 2017, com 280 mil amostras divididas em 10 classes. Na mesma linha do MNIST e do EMNIST, o dataset DIDA [27] possui 250 mil amostras de números manuscritos coletados das imagens de documentos históricos manuscritos suecos entre os anos de 1800 e 1940.

Outro dataset no campo do reconhecimento de números é o SVHN dataset [32]. Este dataset possui imagens do mundo real seguindo o mesmo padrão do MNIST, com imagens de 32x32 pixeis centradas em um único caractere. O dataset contém 73257 dígitos para treinamento, 26032 dígitos para teste e mais de 500 mil amostras adicionais obtidas através de imagens coletadas usando o Google Street View.

Para ambos os datasets é possível encontrar projetos baseados em redes neurais convolucionais para detecção e classificação dos dígitos. Entretanto, foram buscados trabalhos focados no dataset SVHN por oferecer características mais próximas do problema proposto. Entre as abordagens encontradas estavam projetos que utilizavam Yolo V2 [44], Yolo V3 [43], Yolo V5 [6, 7], RetinaNet [38, 42], ResNet [28], Faster R-CNN [49], Single Shot Detector (SSD) SSD-300 [4] e o mais recente, SAM [17].

## 2.3 APLICAÇÕES

Ao realizar buscas contemplando o cenário da aplicação, foi possível encontrar trabalhos que apontam soluções com diversos níveis de complexidade. Entre eles, Elrefaei *et al.* [15] propõem uma abordagem usando *template matching* para a resolução do problema usando dispositivos móveis. O pipeline proposto envolve uma camada de pré-processamento, uma etapa de segmentação de dígitos e uma camada de reconhecimento de dígitos usando *template matching*. A camada de pré-processamento utiliza operações de conversão para escala de cinza, binarização, redução de ruído e recorte da área de leitura. Os autores atingiram 96,49% de acurácia no reconhecimento dos dígitos em um total de 114 imagens do mesmo modelo de medidor.

[4] `https://github.com/vidit04/SSD-300-Implementation-on-SVHN-dataset`

Chouiten & Schaeffer propõem um sistema de visão para leitura de medidores de gás também em duas etapas contendo pré-processamento e reconhecimento usando OCR [8]. Os autores aplicaram detecção de área de interesse (ROI), troca de espaço de cor, normalizações e operações morfológicas para permitir a segmentação correta dos números e identificar o que era a parte inteira e o que era a parte decimal. Posteriormente, os números segmentados foram reconhecidos por um OCR. Como resultado final, atingiram uma acurácia média de 93% e de 99% usando celulares iOS.

Cerman *et al.* [5] também propuseram uma aplicação para dispositivos móveis com o intuito de realizar o escaneamento de medidores de energia analógicos. O pipeline de reconhecimento continha diversas etapas de pré-processamento para detecção de *background*, detecção de marcador de dígito decimal vermelho, remoção de ruído, remoção de reflexão e equalização de histograma, aplicação de limiar adaptativo, binarização, detecção de componentes conectados, filtragem de componentes conectados e extração de dígitos e, por fim, normalização do tamanho dos dígitos. Para a classificação dos dígitos, os autores testaram abordagens usando uma rede neural convolucional baseada na LeNet-5 (97% de acurácia) e também Tesseract (86% de acurácia).

Álvares *et al.* [1] propõem a implementação de um aplicativo em dispositivos móveis para leitura de medidores de energia e água baseado em extração de caracteres seguindo o método proposto por Cerman *et al.* [5]. Após o pré-processamento, um segundo estágio contendo uma rede neural convolucional realiza a classificação do digito. Esse método supera o Tesseract em 12%.

Spichkova *et al.* [41] fazem uma análise da utilização do Tesseract, Google Cloud Vision, AWS Rekognition e Azure Vision também em uma aplicação móvel para reconhecimento de valores em medidores de energia e gás. A solução proposta por eles descartou o uso do Tesseract e do Azure Vision, ficando com a opção da Google e AWS. Após um pipeline de processamento de imagem, o serviço da AWS performou melhor. Além disso, o artigo propõe também uma arquitetura de solução envolvendo a leitura dos medidores, telas de gerenciamento e do backend para armazenamento dos dados e uma arquitetura com blockchain para envio dos dados à concessionária de energia local.

Hong *et al.* [23] propõem um fluxo completo para detecção do hidrômetro, correção da orientação da imagem, recorte das áreas de interesse e extração do valor dos caracteres e também dos ponteiros. Além disso, os autores montaram um dataset de 1000 imagens com apenas um único modelo de hidrômetro e consideraram imagens com diferentes condições de iluminação, oclusão e diferentes orientações. Também foi desenvolvido sistema robusto baseado em redes neurais convolucionais para leitura automática de instrumentos hidrômetros composto por seis módulos, sendo o primeiro para detecção da posição do relógio, o segundo para alinhamento da orientação, o terceiro para localização dos elementos relevantes. Já o quarto módulo realiza uma detecção de keypoints para localizar e separar cada digito, o módulo 5 faz a leitura dos dígitos e o módulo 6 faz a leitura dos ponteiros. Como resultado, o sistema apresentou um baixo nível de

erro em cenários desafiadores de oclusão e luminosidade.

Em linha semelhante, Yolo V5 também foi utilizado para fazer leitura de medidores de energia com ponteiros [50] e também de hidrômetros [31]. No caso do primeiro, os autores realizaram uma modificação na estrutura da Yolo V5 para melhorar a detecção dos ponteiros através de uma camada de multi-scale. Já o segundo, utilizou Yolo V5 na sua forma original e um dataset de 1573 imagens anotadas para detectar a área que representa a quantidade de metros cúbicos, a quantidade de litros consumida e a localização e classificação dos dígitos.

No Brasil, soluções semelhantes são fornecidas por prestadoras de serviços para condomínios para acompanhamento do consumo de água por apartamento usando medidores individualizados. Alguns exemplos de fornecedoras deste serviço são a Acquameter [5], Aquior [6] e Hidrozap[7].

A solução aqui proposta permeia as aplicações revisadas, porém propõe a utilização de um único modelo de deep learning para a detecção e classificação dos dígitos simplificando os pipelines propostos por Chouiten & Schaeffer e Cerman *et al.* Além disso, não é feita uma proposta de solução para detectar os valores decimais ou os ponteiros que a representam e nem fazer o realinhamento do relógio como proposto por Hong *et al.* visto que o usuário deverá enquadrar corretamente o objeto. Na direção da realização da detecção de objetos, foi utilizado a rede Faster R-CNN como substituta ao modelo Yolo V5 empregado por alguns autores com o intuito de evitar problemas com licenças. O dataset montado e utilizado contempla 5 modelos distintos de hidrômetros e menos de 600 imagens, menor do que os artigos listados, que foram suficientes para treinar os modelos escolhidos e atingir acurácia de 93% no conjunto de validação e também em testes de campo.

[5] https://www.acquameter.eng.br/leitura-por-fotomedicao/
[6] https://www.aquior.com.br/leitura-individual-por-foto-medicao-em-sao-paulo/
[7] http://18.229.97.26/

# 3

# CENÁRIO DA APLICAÇÃO

Este capitulo abordará tópicos relacionados ao cenário onde a tecnologia desenvolvida será utilizada. Para entender as possibilidades e limitações, é necessário entender o contexto brasileiro dos modelos de hidrômetros, bem como as necessidades apontadas pela empresa contratante visto que a tecnologia foi embarcada em um produto que já se encontrava em uso por diversos clientes. Assim, existiram requisitos que precisaram ser atendidos e que em hipótese nenhuma envolviam a remodelação do produto vigente.

## 3.1 MODELOS DE HIDRÔMETRO

No Brasil, a variedade das construções na cidade e no campo representadas pela arquitetura e pela idade das mesmas se apresenta como uma forte evidência da diferença de material utilizado para a construção das mesmas. Esta diferença também se materializa nos equipamentos para medição do consumo de água.

Tais equipamentos, ainda que vistoriados pelo INMETRO [2], apresentam diferenças na sua composição, podendo ser de metal ou plástico (vide Figuras 2 e 1), podem apresentar apenas ponteiros para representar as casas decimais ou dígitos decimais (vide Figuras 2 e 3), ter 4 ou 5 dígitos representando a parte inteira do valor (vide Figuras 1 e 3). Outras diferenças podem ser encontradas entre os modelos de hidrômetros, como a cor do background do número, podendo ser preta ou branca para os dígitos inteiros (vide Figura 3) e vermelha ou branca para os dígitos decimais (vide Figuras 3.Esquerda e 1). Além de todas essas diferenças, alguns modelos possuem as catracas do mecanismo de rotação dos dígitos expostas e outros modelos possuem uma cobertura das catracas (vide Figuras 4). Alguns modelos possuem o vidro que protege o relógio, na forma plana, sem distorcer os números. Já outros modelos apresentam um vidro com distorção para aumentar o tamanho de todos os números e outros modelos possuem vidro com distorção apenas dos números inteiros.

Dessa forma, temos um cenário que o próprio objeto impõe uma grande variedade na sua forma e cor, sendo assim, propicia a utilização de redes neurais convolucionais devido a sua capacidade de generalização, se os dados apresentados para ela forem representativos da realidade.

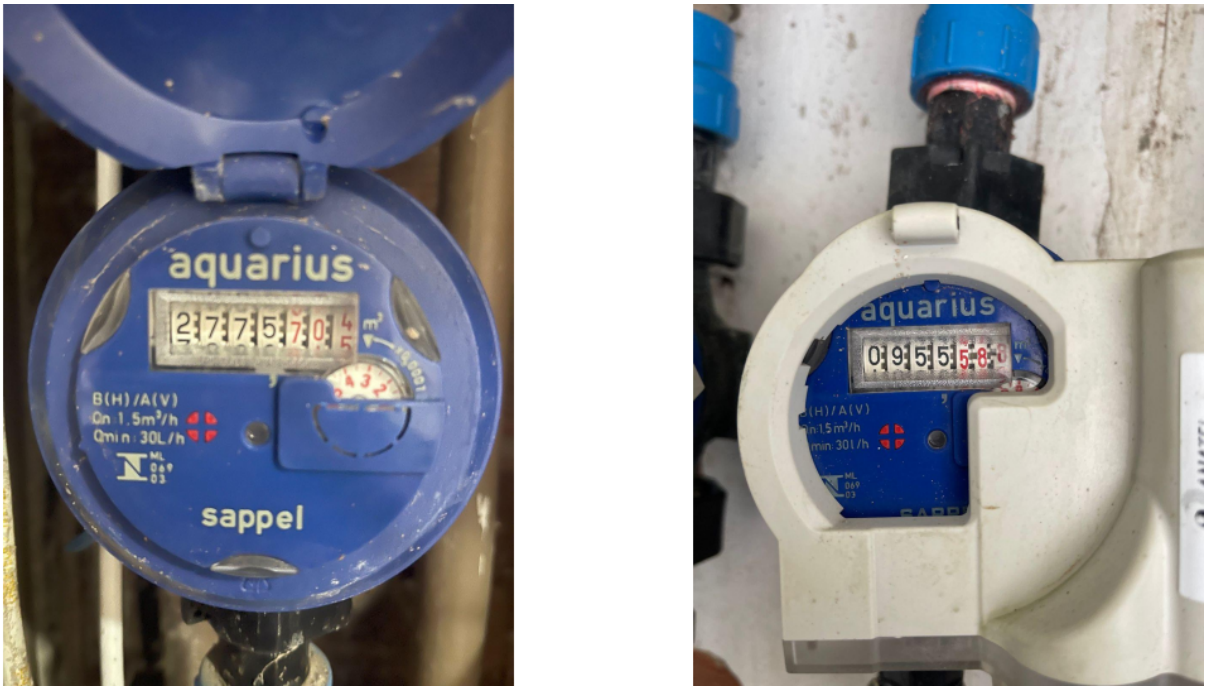


Figura 1: **Direita:** Hidrômetro Classe A Aquarius em plástico, com 5 dígitos inteiros pretos, 3 dígitos decimais vermelhos e background branco, sem equipamento para aquisição de dados via rádio-frequência e com catraca exposta. **Esquerda:** Hidrômetro Classe A Aquarius em plástico com 5 dígitos inteiros pretos, 3 dígitos decimais vermelhos e background branco *com* equipamento para aquisição de dados via rádio-frequência e com catraca exposta.


Figura 2: **Direita:** Hidrômetro classe B em metal, com 4 dígitos inteiros pretos, ponteiros para decimais vermelhos e background branco e *sem* catraca exposta. **Esquerda:** Vista frontal de hidrômetro do mesmo modelo para melhor visualização de seus componentes.



Figura 3: Hidrômetros da marca Aquarius com anel de transmissão por rádio frequência com diferenças na cor do background dos dígitos.




Figura 4: Diferentes modelos de Hidrômetros com vidro que apresenta distorções e diferenças na cor de background dos dígitos.

## 3.2 REQUISITOS APONTADOS PELA EMPRESA CONTRATANTE

A empresa contratante já possuía um *webapp* em execução para gestão de consumo de água e gás em apartamentos. Este webapp foi desenvolvido utilizando a linguagem Javascript/Typescript a partir do framework ReactJS[1]. Logo, foi solicitado que a implementação do reconhecimento de caracteres fosse realizada sem demasiadas alterações na estrutura da aplicação.

[1] `https://github.com/facebook/react`

Outro requisito importante é que a tecnologia fosse robusta a pequenas rotações e variações na luminosidade, visto que não é possivel controlar a iluminação nos locais onde os hidrômetros dos apartamentos ficam dispostos. Também foi pontuado que a aquisição da imagem era feita por dispositivos que não eram proprietários da empresa, logo, não existia um controle do hardware, podendo a câmera ter uma boa qualidade de imagem ou não.

Além disso, outros dois fatores foram relevantes para a decisão da proposta de solução: o primeiro deles foi a sincronia do processo, no qual o usuário deveria submeter a imagem e poder realizar qualquer ajuste no valor recebido logo em seguida, inviabilizando execuções assíncronas. O segundo fator foi a imposição do uso de serviços cloud da AWS, caso necessário.

Por fim, a empresa contratante afirmou que o interessante para ela era apenas a assertividade dos números inteiros (dígitos pretos ou com background preto), que não deveria existir uma preocupação com os dígitos decimais e que esta assertividade deveria ser superior a 90%.

Desta forma, ratifica-se aqui que o escopo deste trabalho era apenas o desenvolvimento de uma tecnologia para Reconhecimento Ótico de Caracteres. Outras funcionalidades como autenticação, armazenamento e recuperação de dados, cálculo de consumo e quaisquer outras estão fora deste escopo.

# 4

# SOLUÇÃO PROPOSTA

Dadas as diversas possibilidades de implementação da extração de caracteres, independente da tecnologia a ser utilizada para esta finalidade, é necessário que seja realizada a captura da imagem. Em qualquer cenário, o ideal é reduzir o máximo possível de informações adjacentes não utilizáveis e por esse motivo preferiu-se limitar o usuário a tirar a foto e não carregar uma imagem capturada anteriormente. Assim, implementou-se a captura da imagem com a câmera do celular, dentro do próprio webapp, considerando apenas a região dos dígitos (vide Figura 5) em uma área reduzida.


Figura 5: Interface da página de coleta de dados e recorte da imagem para considerar apenas a região dos dígitos.

Esta imagem recortada deve ser revisada pelo leiturista antes do envio para checagem da

qualidade da imagem. Após confirmação, a imagem era transformada em uma string Base64 e enviada para uma função Lambda da AWS onde é feita a extração dos dados. Após a extração dos dados, uma string é retornada para o webapp. Caso esteja completamente correta, o leiturista pode aceitá-la. Em caso de erro, o leiturista pode tirar uma nova foto e fazer uma nova leitura ou corrigir a string manualmente. Após a confirmação do valor, o valor e a imagem são enviados para o servidor do cliente. Este fluxo pode ser visualizado na Figura 6.

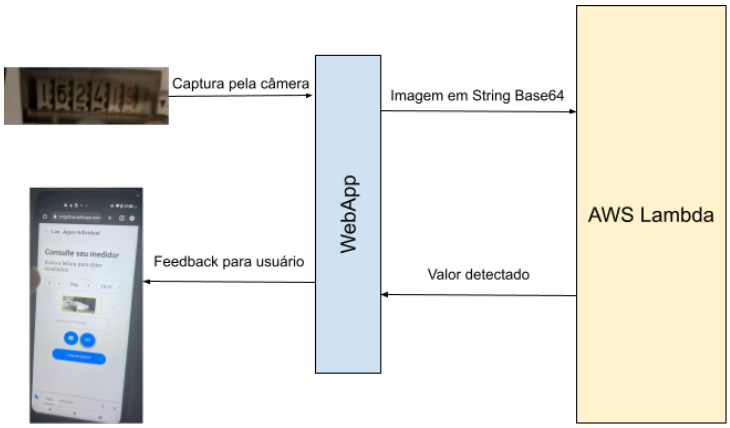


Figura 6: Fluxo de captura de imagem, envio para a AWS Lambda e retorno do valor detectado.

Para a utilização do AWS Textract, a imagem recebida como string em Base64 é encaminhada para o serviço Textract. Para cada resposta encontrada, os caracteres eram filtrados para que apenas os caracteres 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9 permanecessem presentes em cada palavra. Após esta etapa, a maior palavra era a escolhida como resposta final.

Para a utilização da detecção de objetos, na função Lambda temos o seguinte fluxo: A string em Base64 que representa a imagem é recebida e reconvertida em imagem do OpenCV. Esta imagem por sua vez é normalizada e transformada em um tensor com tamanho $[1,3,640,640]$ onde temos $[batch, channels, width, heigth]$. Este tensor é enviado ao modelo de detecção de objetos, que retorna as bounding boxes, as labels e os scores de cada detecção. As detecções são filtradas, primeiramente, pelo seu score com um threshold de 0.8. Em seguida, as bounding boxes filtradas são ordenadas de acordo com o seu valor na coordenada X, ou seja, caixas mais à esquerda são posicionadas nas primeiras posições de um vetor. Este vetor forma a palavra a ser retornada como resposta final.

Este último fluxo se assemelha ao processo sugerido por *CERMAN et al.* [5] e por *ALVARES et al.*[1], porém apresenta diferenciações por não necessitar dos diversos passos de transformações morfológicas aplicados à imagem para extrair bordas e contornos e também necessitar de menos amostras do que relatado nos trabalhos citados. Neste caso, o modelo de deep learning será o responsável por fazer a extração das features necessárias para o reconhecimento de cada dígito na etapa de treinamento e aplicá-los para reconhecimento na etapa da inferência.

# 5

# METODOLOGIA

Neste capítulo serão abordados os métodos utilizados para testar as tecnologias selecionadas para compor o módulo de reconhecimento de dígitos.

## 5.1 ABORDAGEM COM OCR

Durante o processo de construção da solução, foram testadas duas tecnologias de OCR de mercado. A primeira delas envolveu o uso do Tesseract e um módulo para React JS. Já a segunda abordagem envolveu a utilização do serviço Textract da Amazon para fazer o reconhecimento dos dígitos. Abaixo será explicado com detalhes a implementação de cada um desses métodos, os testes feitos e os resultados obtidos em cada um das etapas.

Importante ressaltar que a escolha por essas duas soluções aconteceram pela popularidade, no caso da Tesseract, e pela indicação de uso de plataforma cloud da Amazon pelo cliente contratante, no caso da AWS Textract.

### 5.1.1 Tesseract

Considerando o cenário de restrições apresentadas, primeiramente foi considerada a opção de realizar a detecção dos caracteres de forma embarcada no próprio navegador do dispositivo móvel. Assim, seria possível realizar a medição de todos os hidrômetros do edifício sem mandar nenhuma requisição para a nuvem ou para o backend, diminuindo o tempo de resposta e diminuindo também a quantidade de dados trafegados na rede.

Foi implementada uma *Single Page Application* (SPA, ou Aplicação de Página Única) utilizando React JS e o Tesseract.JS. Este último é uma API do Tesseract implementada usando Javascript que permite a execução de OCR no próprio navegador. Nesta versão, a página continha apenas um elemento de câmera e um botão de captura, que ao ser clicado fazia a captura da imagem e a enviava para o Tesseract (Figura 7). A aplicação ficou hospedada no Github Pages para a realização dos testes.

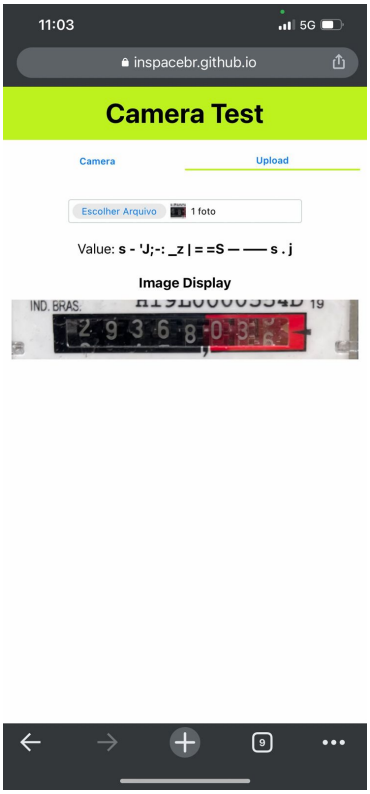


Figura 7: SPA para teste com Tesseract.

No primeiro momento, toda a imagem era enviada para o Tesseract e os resultados obtidos apresentavam caracteres aleatórios que não correspondiam ao observado no relógio. Foram adotadas estratégias para recortar apenas a área de interesse contendo o texto que deveria ser reconhecido e mesmo assim o Tesseract devolvia caracteres não alfa-numéricos e muitas vezes sequências que não faziam sentido. Alguns resultados de testes preliminares e testes reais podem ser vistos na Figura 8.

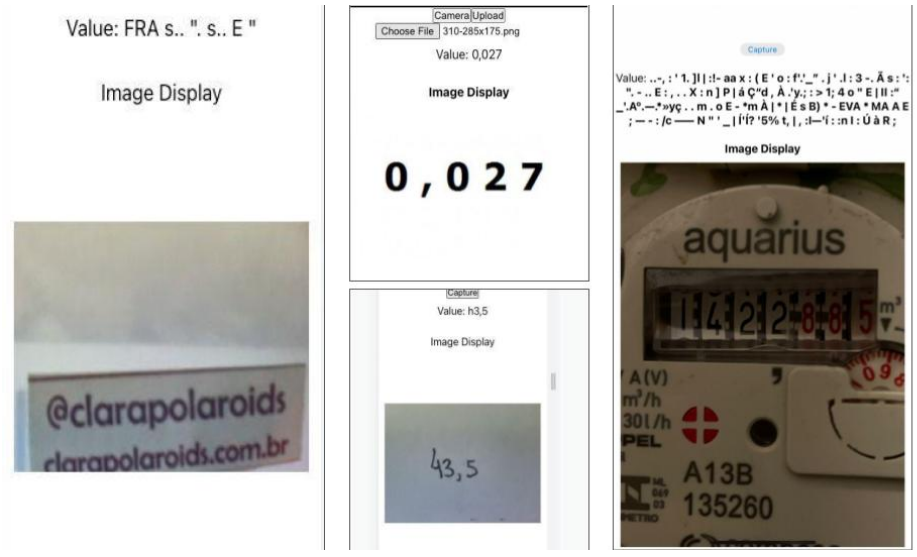


Figura 8: Resultados com a SPA para teste com Tesseract.

### 5.1.2 AWS Textract

Com os resultados obtidos no teste anterior, foi descartada a ideia de utilizar o Tesseract embarcado no navegador e foi tentada uma nova abordagem usando o AWS Textract. Neste caso, a implementação da interface gráfica já foi feita considerando a aplicação do cliente e com recorte da área de interesse para evitar que textos adjacentes ao relógio fossem levados em consideração.

Testes de laboratório apontaram um bom resultado na identificação dos dígitos como é possível ver na Figura 9. Entretanto, os testes em campo apontaram uma fragilidade da solução, pois o serviço da Amazon muitas vezes não era capaz de retornar uma palavra contendo os dígitos ou apresentava números que não existiam em função da exposição da catraca presente no hidrômetro (Figura 10).

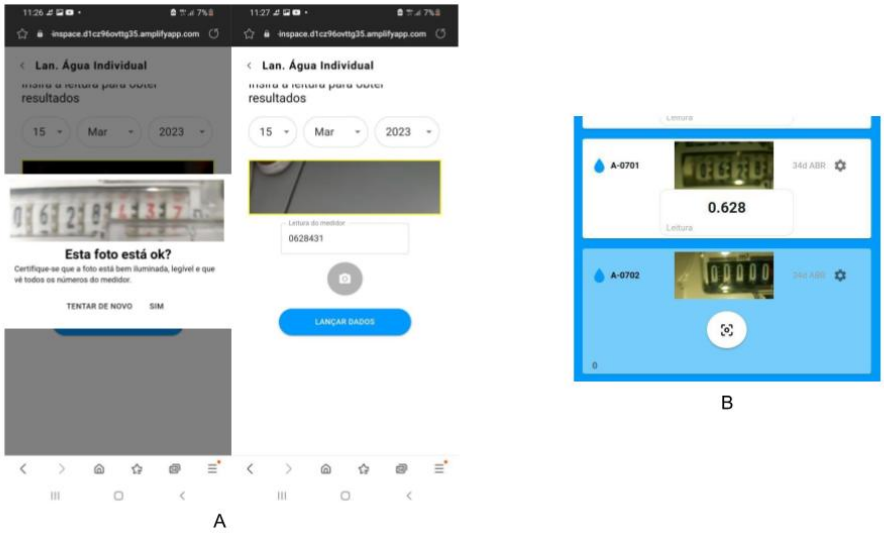


Figura 9: Resultados positivos da utilização do AWS Textract.

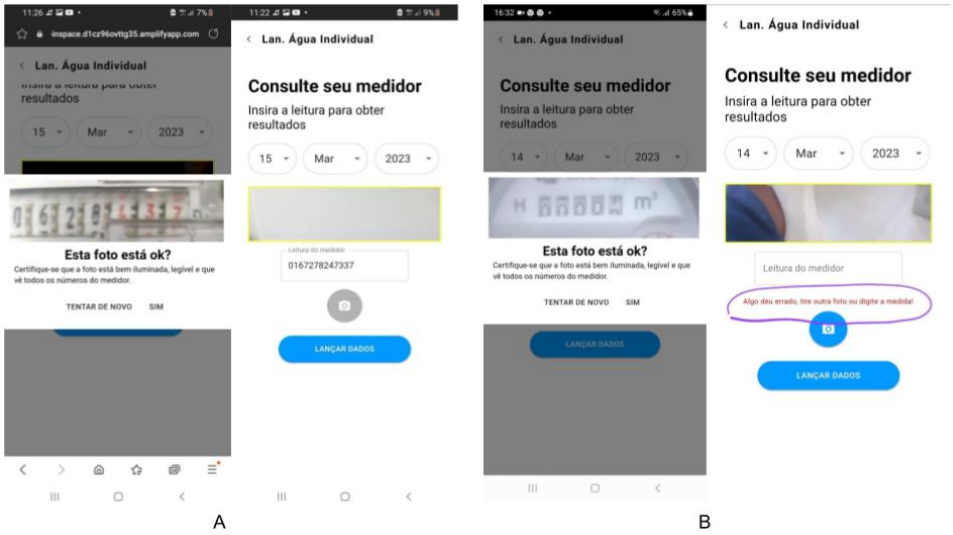


Figura 10: Resultados negativos da utilização do AWS Textract.

Ainda com o intuito de manter a abordagem usando OCR e o AWS Textract, foram aplicadas camadas de pré-processamento nas imagens, com aplicação de borramentos, ruídos e operações morfológicas, mas estas não apresentaram melhorias significativas no resultado. Outra observação interessante foi feita a partir da avaliação do score de confiança entregue pelo AWS Textract para as palavras detectadas. Mesmo no caso de palavras corretas, a pontuação de confiança era abaixo de 50%, em raros casos a confiança passou de 80%.

## 5.2 ABORDAGEM COM DETECÇÃO DE OBJETO

Ao estabelecer que cada digito é um objeto em particular, é possível estabelecer um outro caminho para a resolução do problema, agora pela ótica de detecção de objetos. É importante ressaltar que este caminho é diferente da abordagem da classificação, pois existirá sempre mais de um objeto presente na cena e a sua localização é importante para o correto ordenamento dos dígitos.

### 5.2.1 Dataset

#### *5.2.1.1 Coleta dos dados*

Para a captura das imagens, foi criada uma aplicação web simples, idêntica à página que seria usada no webapp da empresa contratante. A página contém um elemento de câmera e dois botões, sendo um deles para controle do flash e outro para realizar a captura (Figura 11).

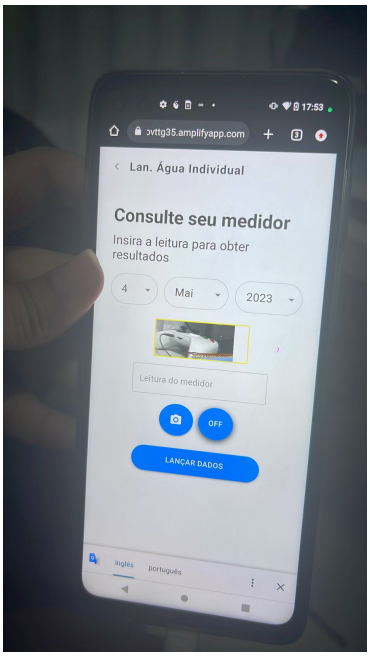


Figura 11: Tela de captura das imagens para montagem do dataset.

Após a captura, a imagem foi transformada em uma string em Base64 e enviada também para um repositório no serviço de bucket da Amazon, o Amazon S3. Devido ao formato alongado,

decidiu-se por aplicar operações de *zero-padding* nas imagens, construindo uma borda preta ao redor da mesma para que ficassem no formato quadrado ou proporções 1:1. Assim, ao utilizá-las no treinamento, não seria necessário realizar um reescalonamento da mesma, distorcendo os números que eram os elementos a serem identificados. Na Figura 12 é possível ver algumas imagens resultantes.

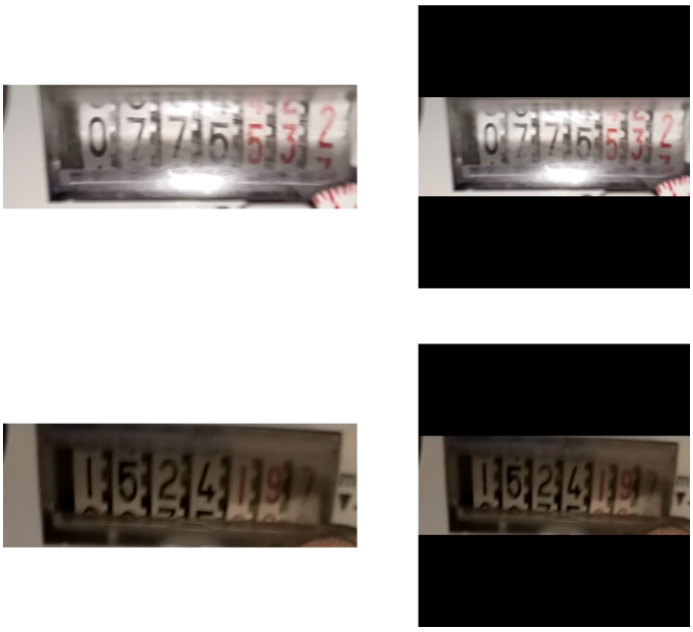

Figura 12: Imagens originais e após aplicação do zero-padding.

Todas as imagens do dataset foram capturadas com dois dispositivos celulares, sendo um deles o Motorola G8 Power[1] com câmera de até 16 megapixels e um Motorola Moto G100[2] com câmera de até 64 megapixels. As fotos foram capturadas considerando a iluminação do próprio local, sendo influenciada pela iluminação natural e artificial dos prédios. O flash das câmeras foi usado de forma aleatória, sem seguir distribuição específica e a escolha de qual dispositivo usar também foi feita de forma aleatória sem seguir distribuição específica.

Os dados foram anotados usando a ferramenta Roboflow[3], que permite realizar as anotações, obter estatísticas do balanceamento do dataset pra cada classe, aplicar transformações para fazer *data augmentation*, exportar o dataset e as anotações, realizar treinamentos em nuvem própria, e fazer o deployment do modelo com acesso via *API*. Esta última, só é possivel se houver acesso a uma assinatura *premium*.

As anotações foram realizadas definindo a menor região quadrada que continha um dígito (*bounding box*) e, para cada região, um classe referente ao dígito. Na Figura 13 está um exemplo de uma anotação realizada.

[1] https://www.tudocelular.com/Motorola/fichas-tecnicas/n6123/Motorola-Moto-G8-Power.html

[2] https://www.tudocelular.com/Motorola/fichas-tecnicas/n6759/Motorola-Moto-G100.html

[3] https://roboflow.com/

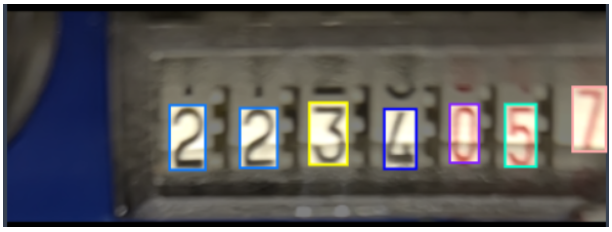

Figura 13: Exemplo de anotação dos dígitos presentes em um hidrômetro. Cada cor representa uma classe distinta.

#### *5.2.1.2 Descrição do Dataset*

O conjunto total de imagens deste dataset possui 226 amostras com 1498 anotações, contendo em média 6,6 anotações por imagem. As imagens foram coletadas seguindo o protocolo já citado. Já as anotações foram feitas para 10 classes, correspondendo aos dígitos de 0 até 9. Na Figura 14, é possível verificar a distribuição da quantidade de anotações nas classes.


Figura 14: Balanceamento das classes no dataset.

As imagens foram coletadas em dois prédios, sendo um deles com menos de 10 anos e o outro com mais de 20 anos. Assim, foi possível ter uma grande variabilidade no número presente nos hidrômetros, bem como uma variabilidade na posição de cada dígito no hidrômetro. Essa variabilidade foi alcançada, como é possível ver nas Figuras 15 e 16 que contêm o mapa de calor representando a frequência do posicionamento de cada dígito nas imagens.

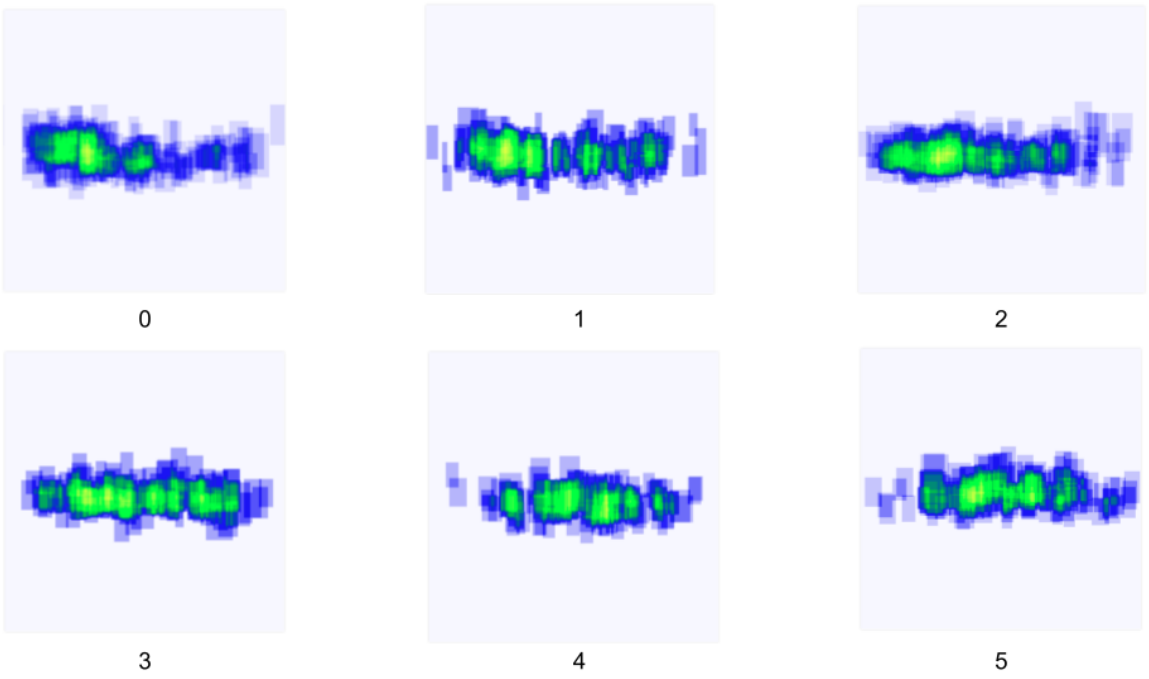


Figura 15: Frequência da distribuição dos dígitos 0, 1, 2, 3, 4 e 5 nas imagens do dataset.

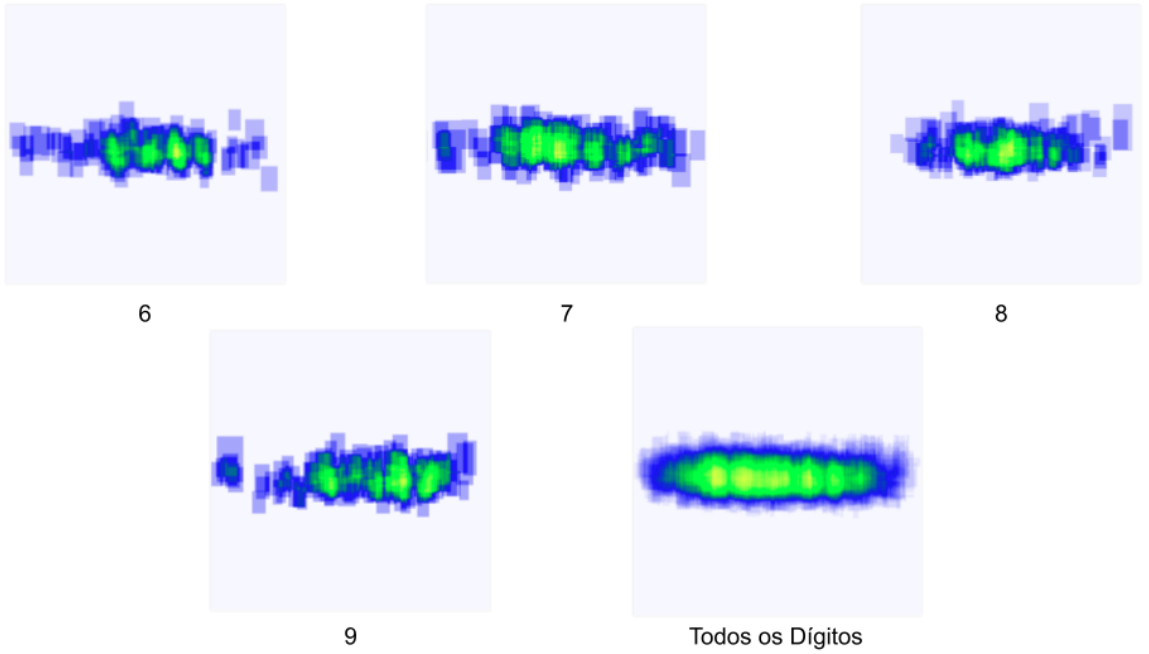


Figura 16: Frequência da distribuição dos dígitos 6, 7, 8, 9 e todos os dígitos nas imagens do dataset.

Para o conjunto de treinamento e validação, o dataset foi divido na proporção de 70% para treinamento e 30% para validação. Por padrão, o Roboflow faz o split antes de aplicar as

transformações e cada exemplo de treinamento pode gerar até 3 outros exemplos com transformações e não são aplicadas transformações mas imagens de validação. Para isso, foram escolhidas as seguintes transformações:

- **Rotação** entre -15º e +15º para simular a variação do posicionamento do leiturista
- **Saturação** entre -25% e +25% para simular a variação na luminosidade ou qualidade de câmera
- **Exposição** entre -25% e +25% para simular a variação na luminosidade ou qualidade de câmera
- **Ruído** entre até 3% dos pixels para simular a variação na qualidade de câmera

Abaixo é possível ver a Figura 17 com a configuração das transformações no Roboflow.

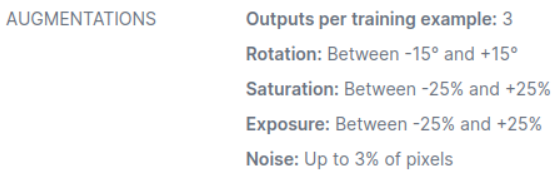


Figura 17: Transformações aplicadas para data augmentation configuradas no Roboflow.

Não foi usado conjunto de teste para aumentar a quantidade de imagens no conjunto de treinamento e validação e porque os testes seriam realizados em campo. Assim, o dataset final totalizou 465 imagens para treinamento e 69 imagens para validação (Figura 18).

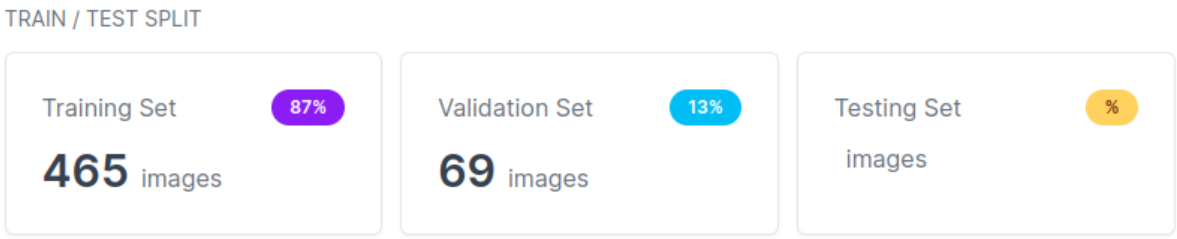


Figura 18: Divisão final do dataset em conjunto de treinamento e validação.

### 5.2.2 Modelo preliminar para detecção de objeto

Após a coleta do dataset, um modelo preliminar foi treinado usando a própria ferramenta da anotação. Na plataforma é possível treinar um modelo que fica aberto ao público. Esta ferramenta foi usada para validar se o conjunto dos dados era suficiente para criar um modelo que pudesse extrair os dígitos de imagens de hidrômetros.

Neste modelo, não era possível alterar parâmetros ou condições de early stopping, por exemplo. O modelo foi treinado por aproximadamente 300 épocas usando o conjunto de treinamento do dataset e foi testado apenas no conjunto de validação.

O modelo resultante apresentou resultados bons para as métricas de mAP, precision e recall, conforme ilustrado na Figura 19.

Embora o Roboflow forneça uma API de detecção para este modelo treinado, ela só fica disponível para os usuários pagantes. Devido ao custo elevado foi pensado na possibilidade de usar modelos mais recentes do Yolo que são conhecidos pela precisão, acurácia e velocidade de execução. Porém, foi verificado que as versões mais recentes possuem uma licença restritiva e seu custo para empresas de pequeno porte é de US$ 5000,00/ano, o que inviabilizaria a solução para o cliente.

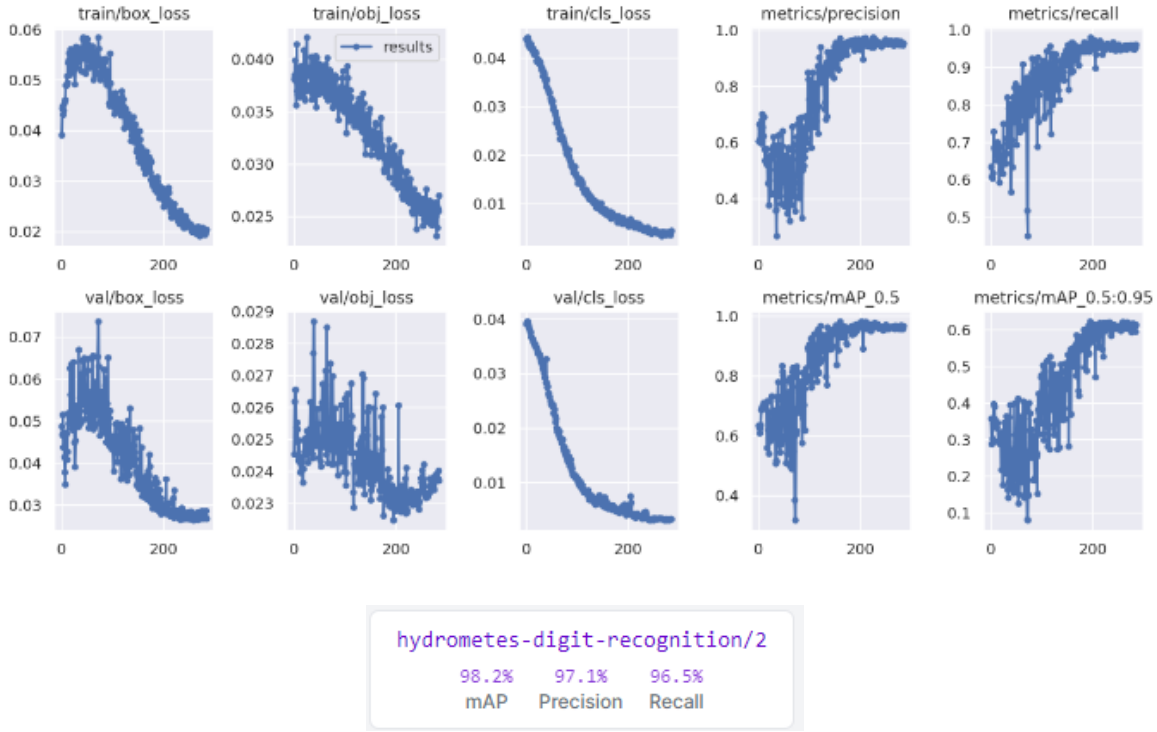


Figura 19: Gráfico de loss e métricas de treinamento e o resultado final para mAP, Precision e Recall.

Como possibilidade, foram levantadas duas opções sendo uma delas o uso da SSD-300 com modelo proposto inicialmente por Liu *et al.* [30] ou da Faster R-CNN de Ren *et al.* [35]. Decidiu-se iniciar testes com a última devido à facilidade de encontrar código atualizado e bem documentado que realizasse seu treinamento.

## 5.3 FASTER R-CNN

As *Region-based CNNs* ou regiões com características extraídas com CNNs são também pioneiras entre as redes que usam aprendizagem profunda para detecção de objetos [19], assim

como a SSD. Primeiramente, as R-CNNs utilizam uma busca seletiva [46] para extrair regiões de proposição para a imagem de entrada e tais regiões são selecionadas com diferentes tamanhos, formas e escalas. Cada região é avaliada por uma rede neural pré-treinada onde as features serão extraídas para posterior avaliação por múltiplas SVMs para a determinação da classe. Ao mesmo tempo, as features extraídas e a classe atribuída são utilizadas por um regressor linear para estimar a bounding box daquela região. Um esquema simplificado pode ser visto na Figura 20.

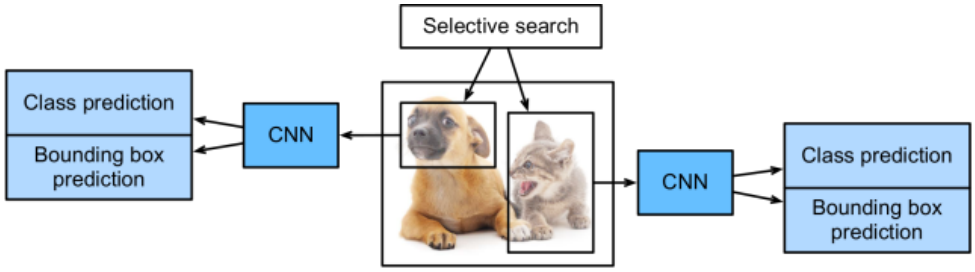


Figura 20: Modelo simplificado da R-CNN. Imagem retirada do livro Dive Into Deep Learning [48].

Embora robusta e apresentando bons resultados na busca por objetos, a R-CNN original apresentava um problema de gargalo na sua performance, a necessidade de passar em uma CNN cada uma das regiões propostas. Além de ser uma operação altamente custosa, também era ineficaz, pois existiam sobreposições entre as regiões que eram avaliadas múltiplas vezes. Assim, Girshick *et al.* sugerem uma modificação na estrutura original onde a CNN seria aplicada na imagem toda ao invés de cada região, surgindo assim a Fast R-CNN [18]. Essa nova CNN também seria treinável, o que apontava para um melhor resultado na extração de features.

As features extraídas pela rede convolucional em conjunto com as regiões propostas pela Busca Seletiva alimentavam uma camada de *ROI Pooling*, que posteriormente alimentava uma camada *Fully connected* para gerar a predição da classe e a bounding box resultante. Um esquema simplificado para a Fast R-CNN pode ser visto na Figura 21.

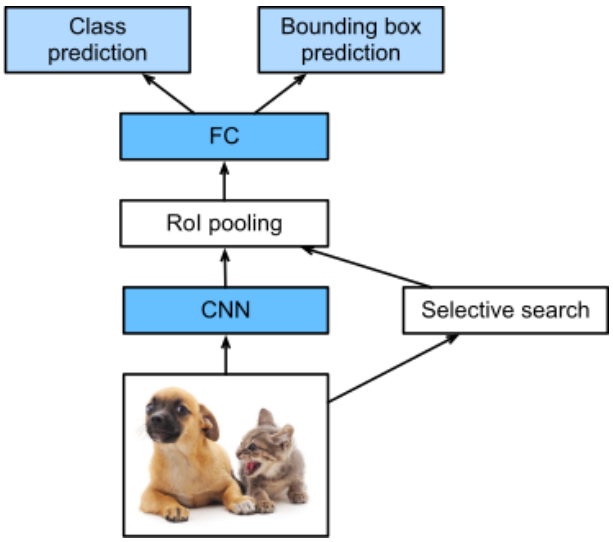


Figura 21: Modelo simplificado da Fast R-CNN. Imagem retirada do livro Dive Into Deep Learning [48].

Para aumentar a robustez e a acurácia da detecção, modelos de R-CNN precisam gerar mais regiões na Busca Seletiva, entretanto, isso acarreta em mais custo computacional. Com o intuito de reduzir a quantidade de regiões propostas e não perder a acurácia, Ren *et al.* [35] propuseram trocar o módulo de Busca Seletiva por uma rede neural de proposição de regiões (RPN). Esta nova rede neural aplica uma convolução na saída da CNN para gerar um novo vetor de features que será utilizado para gerar as regiões. Para essas regiões são definidas sua classe como background ou objetos e uma bounding box. Considerando as novas caixas e as classes, as sobreposições são removidas por um algoritmo de *non-maximum suppression*, gerando por fim as propostas de regiões necessárias pela camada de *ROI Pooling* já existente da Fast R-CNN, seguindo, agora, os mesmos passos desta. Um esquema simplificado para a Faster R-CNN pode ser visto na Figura 22.

Vale ressaltar que a adição da rede de proposta de região como parte da Faster R-CNN permite que a RPN seja treinada em conjunto com o modelo maior. Assim, o objetivo deste modelo não é ser acurado apenas na predição da classe ou da bounding box, mas também na proposição de regiões a serem avaliadas.

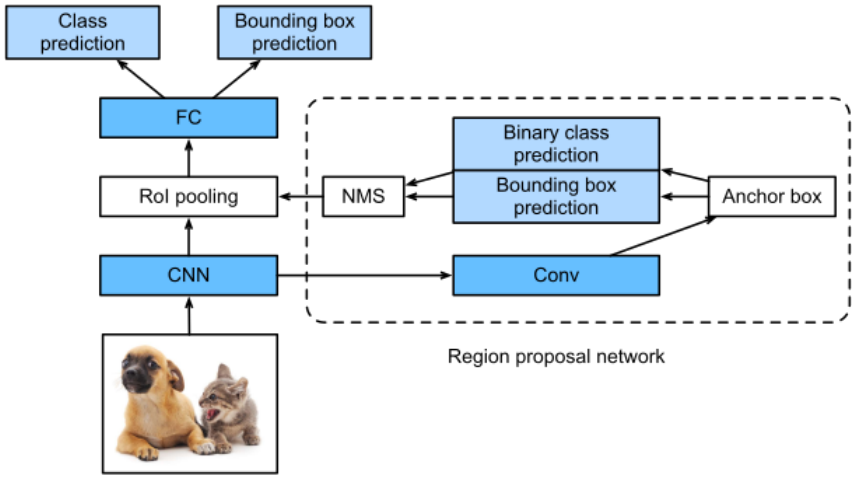


Figura 22: Modelo simplificado da Faster R-CNN. Imagem retirada do livro Dive Into Deep Learning [48].

Também é importante chamar a atenção que no artigo original da Faster R-CNN os autores usaram uma VGG-16 como backbone para extração de features antes de enviá-las para a RPN e para a camada de *ROI Pooling*. Neste trabalho, este backbone será modificado para diferentes modelos com o intuito de avaliar a assertividade dos mesmos.

## 5.4 TREINAMENTO

Para o treinamento foi utilizado um pipeline de código aberto disponível no Github[4]. Este pipeline permite o uso de datasets no formato Pascal-VOC [16] e também o uso de diferentes backbones como estrator de features para a Faster R-CNN. Além disso, neste código é possivel treinar novos modelos, fazer avaliações com métricas como mAP e Recall implementados a partir da biblioteca *pycocotools*[5] e também testar modelos treinados usando códigos prontos para inferência. Além disso, o pipeline permite também a exportação do modelo para o framework ONNX, tanto para GPU quanto para CPU. Este framework também é muito usado para deployment de redes neurais.

Foram treinados 22 modelos, cada um com um backbone distinto que pode ser visto na Tabela 1. Quatro destes possuem implementações fornecidas pelo próprio Pytorch através do seu módulo Torchvision e os demais foram escolhidos aleatoriamente. Os 4 modelos fornecidos pelo Pytorch foram pré-treinados no COCO Dataset [29]. Vale ressaltar que os modelos pré-treinados foram treinados com 50 épocas neste experimento, já os modelos que não tinham sido pré-treinados foram treinados com 100 épocas e este foi o critério de parada adotado.

O treinamento foi realizado em um computador Desktop equipado com uma NVIDIA RTX 3090, Intel Xeon Silver 4214 com 24 cores reais e 24 virtuais, 96 GB de memória RAM e 1 TB SSD. Os parâmetros de treinamento dos modelos foram definidos de acordo com a Tabela 2

[4] https://github.com/sovit-123/fasterrcnn-pytorch-training-pipeline
[5] https://github.com/cocodataset/cocoapi/tree/master

Tabela 1: Backbones escolhidos como extratores de características para a Faster R-CNN.

| **Backbone** | **Disponível no Torchvision** | **Pré-treinado** | **Dataset pré-treinamento** |
|---|---|---|---|
| convnext_tiny | Não | Não | - |
| custom_resnet | Não | Não | - |
| darknet | Não | Não | - |
| efficientnet_b0 | Não | Não | - |
| mbv3_small_nano_head | Não | Não | - |
| mini_darknet_nano_head | Não | Não | - |
| mini_darknet | Não | Não | - |
| mini_squeezenet1_1_small_head | Não | Não | - |
| mini_squeezenet1_1_tiny_head | Não | Não | - |
| mobilenetv3_large_320_fpn | Sim | Sim | COCO Dataset |
| mobilenetv3_large_fpn | Sim | Sim | COCO Dataset |
| mobilevit_xxs | Não | Não | - |
| nano | Não | Não | - |
| regnet_y_400mf | Não | Não | - |
| resnet18 | Não | Não | - |
| resnet50_fpn | Sim | Sim | COCO Dataset |
| resnet50_fpn_v2 | Sim | Sim | COCO Dataset |
| resnet101 | Não | Não | - |
| squeezenet1_0 | Não | Não | - |
| squeezenet1_1_small_head | Não | Não | - |
| squeezenet1_1 | Não | Não | - |
| vitdet_tiny | Não | Não | - |

usando as capacidades da GPU. Outros parâmetros disponíveis no pipeline foram usados no seu valor padrão ou não foram utilizados.

Tabela 2: Lista de parâmetros utilizados para o treinamento dos 7 modelos citados na Tabela 1 e valores adotados.

| Parâmetro | Valor adotado | Descrição |
|---|---|---|
| –model | <Modelos da Tabela 1> | Nome do modelo a ser treinado |
| –data | "./data/" | Caminho para o arquivo de configuração |
| –device | "cuda"(Default) | Dispositivo de treinamento |
| –epochs | 50 ou 100 | Número de épocas do treinamento |
| –workers | 4 (Default) | Número de workers para pré-processamento dos dados, transofmações e "aumentações" |
| –batch | 16 ou 8 | Tamanho do batch de treinamento |
| –lr | 0.001 | Taxa de aprendizagem para o otimizador |
| –imgsz | 640 (Default) | Tamanho da imagem de entrada da rede |
| –name | <Nome do modelo>_training | Nome do diretório para salvamento dos resultados |
| –mosaic | 0 | Probabilidade para uso de mosaico |
| –use-train-aug | True | Uso de aumento de dados no treino como borramento, escala de cinza, contraste de brilho, instabilidade de cor, gama aleatória (todos ao mesmo tempo) |
| –disable-wandb | True | Desativar monitoramento com Weights and Biases |

## 5.5 TESTES

Após o treinamento, foi feito um teste com o conjunto de imagens de validação para fazer uma análise qualitativa das detecções a partir da análise das posições de bounding box e da classe predita pelos modelos. Em seguida, foram realizados outros dois testes considerando o algoritmo para ordenamento das detecções e formação das palavras, gerando os números presentes nos hidrômetros.

Nesta última fase, o primeiro teste considerou todas as detecções para comparar a palavra predita com a palavra anotada. Ou seja, foi considerado como acerto quando os caracteres da palavra predita estavam na mesma ordem da palavra anotada e se quantidade de caracteres era igual em ambas. O segundo teste considerou apenas os primeiros 5 caracteres da palavra predita e também os primeiros 5 caracteres da palavra anotada. Logo, foi considerado um acerto se a ordem dos 5 primeiros caracteres de ambas era igual.

Em seguida, foi feito o deployment da solução usando AWS Lambda e foram conduzidos testes qualitativos em 3 prédios distintos considerando todo o fluxo da solução proposta descrita no próximo capítulo.

# 6

# RESULTADOS

Este cápitulo apresentará os resultados para os testes realizados considerando os testes de inferência para os cinco melhores e cinco piores modelos, teste de formação de palavra e teste em operação.

## 6.1 TESTES DE DETECÇÃO

Considerando os resultados do treinamento usando o aumento de dados, a Tabela 3 apresenta os valores das métricas avaliadas durante o treinamento. Tais resultados apontam que o modelo resnet50_fpn_v2 obteve o maior mAP para a detecção enquanto o modelo custom_resnet obteve o pior mAP. Outro fator a ser avaliado é o tamanho final do modelo, que pode impactar no tempo de carregamento e de resposta do serviço. Dessa forma, o modelo resnet101 foi o modelo mais pesado e o modelo mini_squeezenet1_1_tiny_head foi o modelo mais leve.

Analisando as métricas de treinamento dos 5 piores modelos, foi possível observar que as losses de region proposal, objeto e classe tenderam a zero desde o inicio do treinamento. Entretanto, a loss da bounding box apresentou um comportamento crescente e depois tendeu a zero. Dessa forma, tais modelos poderiam alcançar um resultado melhor quando treinados por mais épocas. As exceções, neste caso, foram os modelos mbv3_small_nano_head e custom_resnet, que não conseguiram chegar no ponto de uma loss decrescente para bounding box. Isso pode indicar uma possível divergência e que a rede seria muito simples para este problema. Na Figura 23 é possível observar um exemplo resultante do teste de inferência para estes modelos.

Entre as 5 com maior mAP, duas foram pré-treinadas no COCO Dataset e outras três não foram. Dentre elas, as três não treinadas também foram as menores e apontam possibilidade de uso inclusive embarcado no dispositivo móvel. A mini_squeezenet1_1_small_head e a squeezenet1_1_small_head apresentaram espaço para melhoria com um treino com mais épocas, já que a loss da bounding box ainda não tinha estabilizado. Na Figura 24 é possível analisar um exemplo de resultado dos testes de inferência onde todos os números vísiveis foram detectados corretamente e sem sobreposição de bounding box.


Figura 23: Um exemplo de resultado do teste de inferência aplicado com os 5 piores modelos. Dígitos visíveis não foram detectados, aconteceram detecções erradas ou múltiplas detecções no mesmo objeto.

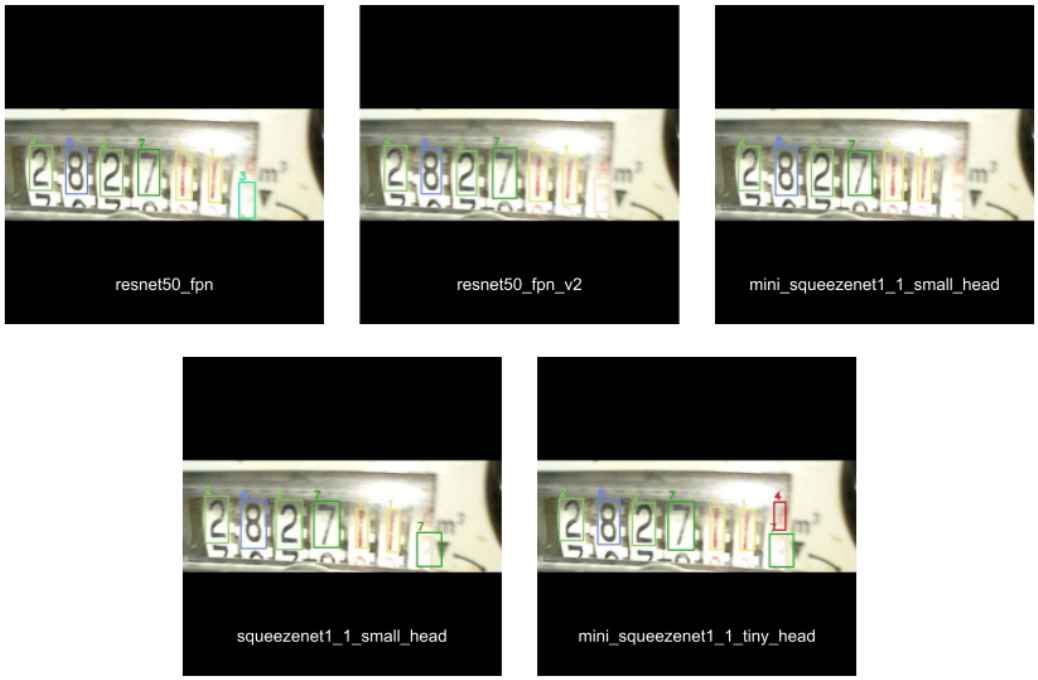


Figura 24: Um exemplo de resultado do teste de inferência aplicado com os 5 melhores modelos. Todos os dígitos visíveis foram detectados corretamente.

Tabela 3: Resultado das métricas de mAP e Recall para os treinamentos. Em amarelo estão destacados os modelos com maior mAP. Em vermelho estão destacados os modelos com menor mAP. O Average Recall apresentado é da época de máximo mAP.

| **Backbone** | **mAP 0.50:0.95** | **Average Recall 0.50:0.95** | **Época do max mAP** | **Tamanho do modelo final** |
|---|---|---|---|---|
| convnext_tiny | 0,458 | 0,417 | 98 | 277MB |
| custom_resnet | 0,008 | 0,010 | 100 | 67MB |
| darknet | 0,173 | 0,220 | 98 | 321MB |
| efficientnet_b0 | 0,307 | 0,305 | 99 | 321MB |
| mbv3_small_nano_head | 0,067 | 0,105 | 100 | 7MB |
| mini_darknet_nano_head | 0,076 | 0,137 | 98 | 8MB |
| mini_darknet | 0,057 | 0,103 | 94 | 34MB |
| mini_squeezenet1_1_small_head | 0,510 | 0,449 | 100 | 16MB |
| mini_squeezenet1_1_tiny_head | 0,496 | 0,441 | 36 | 9MB |
| mobilenetv3_large_320_fpn | 0,475 | 0.41 | 44 | 73MB |
| mobilenetv3_large_fpn | 0,477 | 0.43 | 48 | 73MB |
| mobilevit_xxs | 0,338 | 0,342 | 100 | 73MB |
| nano | 0,471 | 0,429 | 96 | 11MB |
| regnet_y_400mf | 0,375 | 0,356 | 91 | 110MB |
| resnet18 | 0,350 | 0,362 | 97 | 154MB |
| resnet50_fpn | 0,56 | 0,49 | 50 | 158MB |
| resnet50_fpn_v2 | 0,58 | 0,50 | 29 | 165MB |
| resnet101 | 0,432 | 0,40 | 99 | 703MB |
| squeezenet1_0 | 0,485 | 0,436 | 97 | 114MB |
| squeezenet1_1_small_head | 0,506 | 0,442 | 100 | 62MB |
| squeezenet1_1 | 0,495 | 0,442 | 70 | 114MB |
| vitdet_tiny | 0,485 | 0,428 | 96 | 87MB |

## 6.2 TESTES DE CONSTRUÇÃO DE PALAVRA

O algoritmo para construção de palavra a partir das detecções encontradas se mostrou assertivo, sendo capaz de acertar 93% dos casos submetidos. Neste teste foi usado o modelo *resnet50_fpn_v2* devido ao seu maior mAP quando comparado com os demais modelos.

O maior índice de falhas acontece quando algum caractere fica parcialmente ocluso por uma divisão no vidro do hidrômetro, impedindo que a rede faça a detecção correta devido à distorção. Nas Figuras 25 e 26, é possível ver casos corretos e incorretos.

Analisando a Figura 26, também é possível verificar que os dígitos mais significativos na cor preta foram corretamente identificados em 6 das 7 imagens.

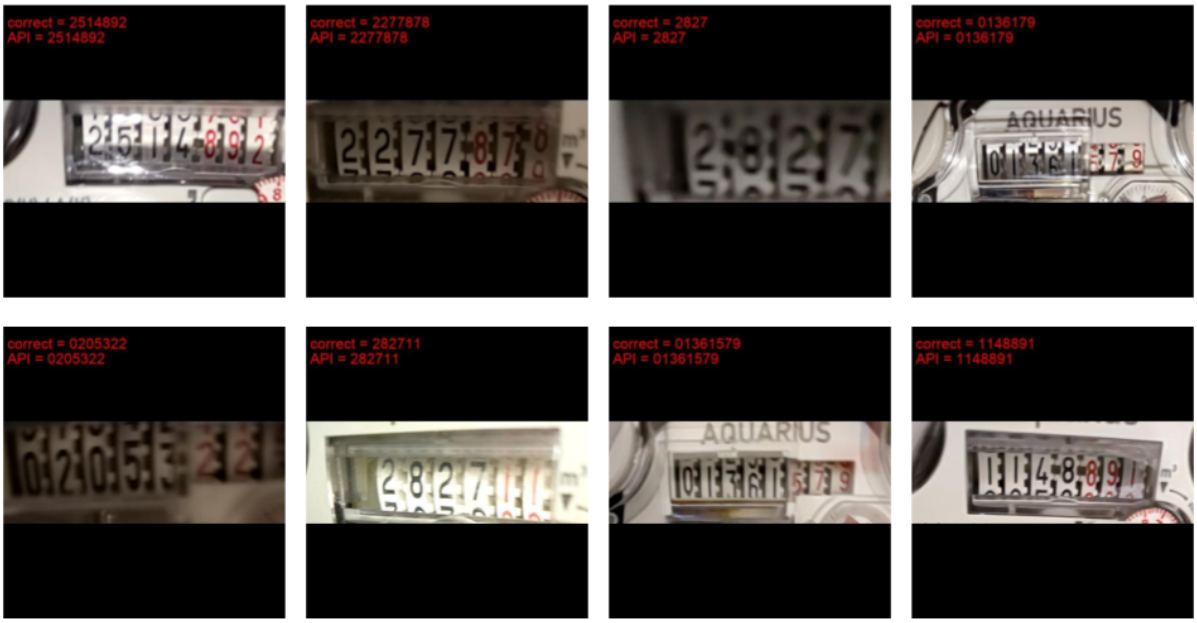


Figura 25: Resultados corretos obtidos em testes para extração do valor do hidrômetro.

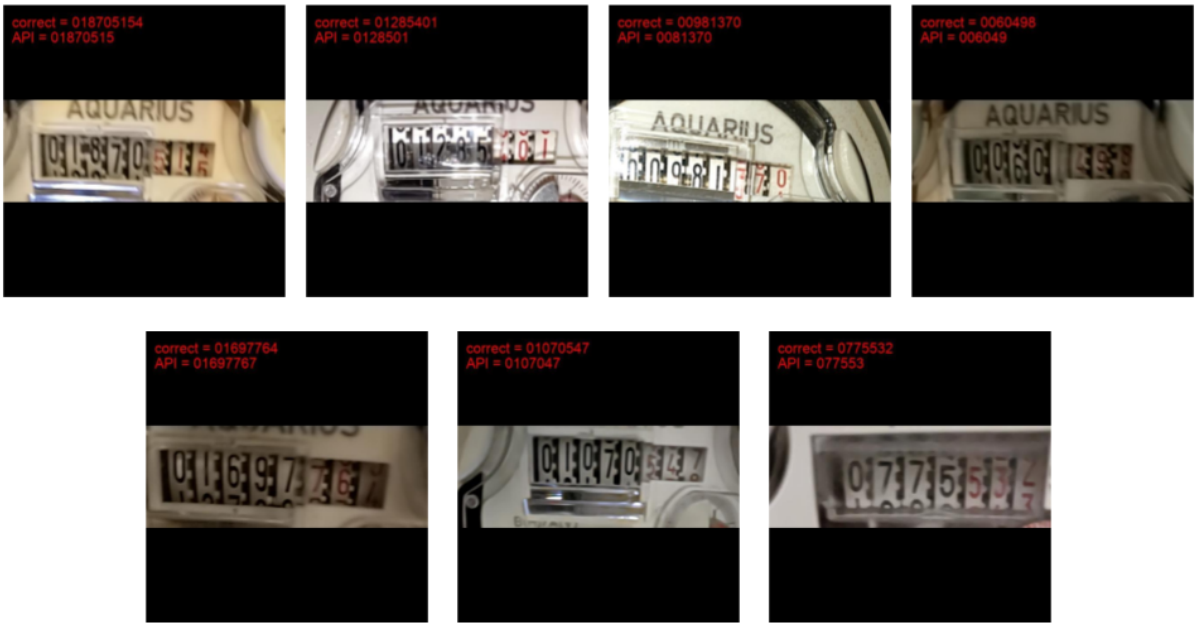


Figura 26: Resultados incorretos obtidos em testes para extração do valor do hidrômetro.

## 6.3 TESTES COM APLICATIVO

Após os testes de detecção e de construção de palavra, o modelo *resnet50_fpn_v2* foi embarcado na aplicação final e testado em 3 prédios e 5 modelos distintos de hidrômetros. O resultado final atingido foi satisfatório, ficando em 95% de assertividade, superando o valor base que era 90%. As principais dificuldades encontradas foram em cenários onde algum caractere estava parcialmente ocluso por algum elemento no vidro do hidrômetro, reforçando o que foi encontrado no teste de construção de palavras. Também foi verificado que os erros que aconteciam nos testes com OCRs listados não aconteceram nesta abordagem, diminuindo os casos onde o usuário precisaria fazer alguma correção. Na Figura 27 é possível ver a resposta recebida pelo aplicativo.

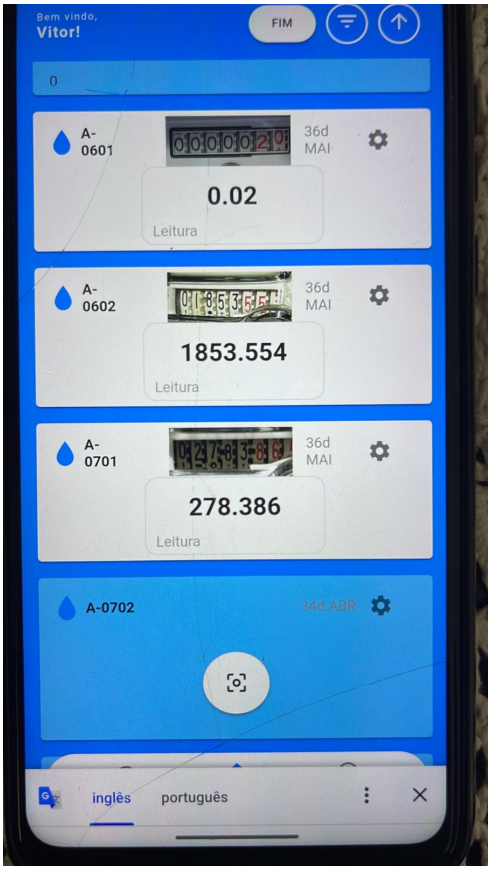


Figura 27: Resultados obtidos em testes fotografando hidrômetros em prédios que não foram contemplados no dataset.

Em termos de tempo de resposta, em geral, o sistema conseguiu gerar resposta para o usuário em aproximadamente 6 segundos, que incluem o tempo de envio da imagem, carregamento do modelo, inferência, construção da palavra e resposta.

# 7

# CONCLUSÕES

Dado o cenário de aplicação e com os testes realizados, foi possível observar que nem sempre soluções de OCR amplamente distribuídas, comerciais ou open-source são as mais indicadas para fazer extração de caracteres quando o domínio de operação é muito especifico ou possui particularidades que influenciam diretamente no resultado. No caso posto, o desalinhamento dos caracteres e a presença de engrenagens que eram semelhantes a caracteres geravam informações errôneas. Assim, a extrapolação do problema para o domínio da detecção de objeto se mostrou uma solução robusta e permitiu a identificação correta dos valores presentes em hidrômetros de 3 prédios com modelos distintos.

O modelo Faster R-CNN com o backbone da Resnet50 se mostrou assertivo e acurado na predição das bounding boxes e nas classes em comparação com os backbones testados, atingindo cerca de 93% de assertividade quando usado para extrair os 5 primeiros caracteres do hidrômetro. O modelo também se mostrou muito eficaz em ignorar informações adjacentes que foram capazes de confundir os algoritmos de OCR testados. Além disso, outros modelos não pré-treinados no COCO também apresentaram resultados interessantes e também eram mais leves do que a opção escolhida para a implementação final.

Este resultado também aponta a eficiência do dataset construído em representar de forma balanceada e significativa cada caractere. Assim, foi capaz de diminuir o viés pela posição do dígito no relógio, distorções devido à presença de efeito de lente no vidro e até mesmo mitigar o impacto de variações no ângulo e na iluminação das imagens.

Como próximos passos deste desenvolvimento espera-se que novos backbones sejam testados no intuito de buscar maior eficiência ao diminuir o tamanho do modelo e ganhando velocidade ao passo que a assertividade final do valor extraído seja mantida. Também espera-se que a arquitetura seja expandida para permitir a extração de caracteres em relógios de energia elétrica e de medidores de consumo de gás. Por fim, faz parte do planejamento investigar o uso dos modelos mini_squeezenet1_1_small_head e mini_squeezenet1_1_tiny_head como uma possível solução para o processamento embarcado, diminuindo o consumo de pacote de dados do usuário.

# REFERÊNCIAS


[1] Álvares, A. J., Souza, A. C. A., & de Castro, M. F. (2020). Implementação de um aplicativo móvel (app) para leitura de medidores de água e energia baseado em visão computacional. *Revista Brasileira de Computação Aplicada*, 12(3):107–121.

[2] Brasil (2018). Portaria nº 295, de 29 de junho de 2018. *Diário Oficial da República Federativa do Brasil*.

[3] Brasil, I. T. (2021). Brasil chega aos 40% de perdas de água potável.

[4] Burkpalli, A. V., Joshi, A., Warad, A. B., & Patil, A. (2022). Automatic number plate recognition using tensorflow and easyocr. *International Research Journal of Modernization in Engineering Technology and Science*, 4(09):493–501.

[5] Cerman, M., Shalunts, G., & Albertini, D. (2016). A mobile recognition system for analog energy meter scanning. In Bebis, G., Boyle, R., Parvin, B., Koracin, D., Porikli, F., Skaff, S., Entezari, A., Min, J., Iwai, D., Sadagic, A., Scheidegger, C., & Isenberg, T., editors, *Advances in Visual Computing*, 247–256.

[6] Chin, J. (2021a). Yolov5-svhn dataset. `https://universe.roboflow.com/soumyadeep-dutta/yolov5-svhn`. visited on 2023-07-31.

[7] Chin, Z.-Y. (2021b). yolov5-schn-detection. visited on 2023-07-31.

[8] Chouiten, M. & Schaeffer, P. (2014). Vision based mobile gas-meter reading. In *Proceedings of Scientific Cooperations International Workshops on Electrical and Computer Engineering Subfields, Istanbul, Turkey*, 94–97.

[9] Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. *CoRR*, abs/1702.05373.

[10] das Nações Unidas, O. (2023a). 46% da população global vive sem acesso a saneamento básico.

[11] das Nações Unidas, O. (2023b). Objetivo para desenvolvimento sustentavel - agenda 2030.

[12] de Pernambuco, D. (2023). Compesa investe em projeto contra desperdício de água.

[13] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255.

[14] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142.

[15] Elrefaei, L. A., Bajaber, A., Natheir, S., AbuSanab, N., & Bazi, M. (2015). Automatic electricity meter reading based on image processing. In *2015 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*, 1–5.

[16] Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2015). The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136.

[17] Foret, P., Kleiner, A., Mobahi, H., & Neyshabur, B. (2020). Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*.

[18] Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, 1440–1448.

[19] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 580–587.

[20] Graves, A., Fernández, S., Gomez, F., & Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, 369–376.

[21] Graves, A. & Graves, A. (2012). Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, 37–45.

[22] Hegghammer, T. (2022). Ocr with tesseract, amazon textract, and google document ai: a benchmarking experiment. *Journal of Computational Social Science*, 5(1):861–882.

[23] Hong, Q., Ding, Y., Lin, J., Wang, M., Wei, Q., Wang, X., & Zeng, M. (2021). Image-based automatic watermeter reading under challenging environments. *Sensors*, 21(2):434.

[24] Jayoma, J. M., Moyon, E. S., & Morales, E. M. O. (2020). Ocr based document archiving and indexing using pytesseract: A record management system for dswd caraga, philippines. In *2020 IEEE 12th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM)*, 1–6.

[25] Jeeva, C., Porselvi, T., Krithika, B., Shreya, R., Priyaa, G. S., & Sivasankari, K. (2022). Intelligent image text reader using easy ocr, nrclex & nltk. In *2022 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS)*, 1–6.

[26] Jocher, G., Chaurasia, A., & Qiu, J. (2023). YOLO by Ultralytics.

[27] Kusetogullari, H., Yavariabdi, A., Hall, J., & Lavesson, N. (2020). Digitnet: A deep handwritten digit detection and recognition method using a new historical handwritten digit dataset. *Big Data Research*.

[28] Liao, A. (2021). Andrew tensorpack. visited on 2023-07-31.

[29] Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312.

[30] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, 21–37.

[31] Martinelli, F., Mercaldo, F., & Santone, A. (2023). Water meter reading for smart grid monitoring. *Sensors*, 23(1):75.

[32] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., & Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning.

[33] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché Buc, F., Fox, E., & Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, 8024–8035.

[34] Rahim, M. S., Nguyen, K. A., Stewart, R. A., Giurco, D., & Blumenstein, M. (2020). Machine learning and data analytic techniques in digital water metering: A review. *Water*, 12(1):294.

[35] Ren, S., He, K., Girshick, R. B., & Sun, J. (2015). Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497.

[36] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

[37] Sai Rakesh Kamisetty, V. N., Sohan Chidvilas, B., Revathy, S., Jeyanthi, P., Anu, V. M., & Mary Gladence, L. (2022). Digitization of data from invoice using ocr. In *2022 6th International Conference on Computing Methodologies and Communication (ICCMC)*, 1–10.

[38] Saviolo, A. (2021). Retinanet-object-detector. visited on 2023-07-31.

[39] Smith, R. (2007). An overview of the tesseract ocr engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, 2:629–633.

[40] Smith, R. W. (2013). History of the Tesseract OCR engine: what worked and what didn't. In Zanibbi, R. & Coüasnon, B., editors, *Document Recognition and Retrieval XX*, 8658:865802.

[41] Spichkova, M., Van Zyl, J., Sachdev, S., Bhardwaj, A., & Desai, N. (2020). Comparison of computer vision approaches in application to the electricity and gas meter reading. In *Evaluation of Novel Approaches to Software Engineering: 14th International Conference, ENASE 2019, Heraklion, Crete, Greece, May 4–5, 2019, Revised Selected Papers 14*, 303–318.

[42] Sup, J. J. (2019a). retinanet-digit-detector. visited on 2023-07-31.

[43] Sup, J. J. (2019b). tf2-eager-yolo3. visited on 2023-07-31.

[44] Sup, J. J. (2019c). Yolo-digit-detector. visited on 2023-07-31.

[45] Thakare, S., Kamble, A., Thengne, V., & Kamble, U. (2018). Document segmentation and language translation using tesseract-ocr. In *2018 IEEE 13th International Conference on Industrial and Information Systems (ICIIS)*, 148–151.

[46] Uijlings, J. R., Van De Sande, K. E., Gevers, T., & Smeulders, A. W. (2013). Selective search for object recognition. *International journal of computer vision*, 104:154–171.

[47] Vedhaviyassh, D., Sudhan, R., Saranya, G., Safa, M., & Arun, D. (2022). Comparative analysis of easyocr and tesseractocr for automatic license plate recognition using deep learning algorithm. In *2022 6th International Conference on Electronics, Communication and Aerospace Technology*, 966–971.

[48] Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2021). Dive into deep learning. *arXiv preprint arXiv:2106.11342*.

[49] Zhang, G. (2019). digits-detector faster rcnn. visited on 2023-07-31.

[50] Zou, L., Wang, K., Wang, X., Zhang, J., Li, R., & Wu, Z. (2023). Automatic recognition reading method of pointer meter based on yolov5-mr model. *Sensors*, 23(14):6644.

[51] Čakić, S., Popović, T., Šandi, S., Krčo, S., & Gazivoda, A. (2020). The use of tesseract ocr number recognition for food tracking and tracing. In *2020 24th International Conference on Information Technology (IT)*, 1–4.