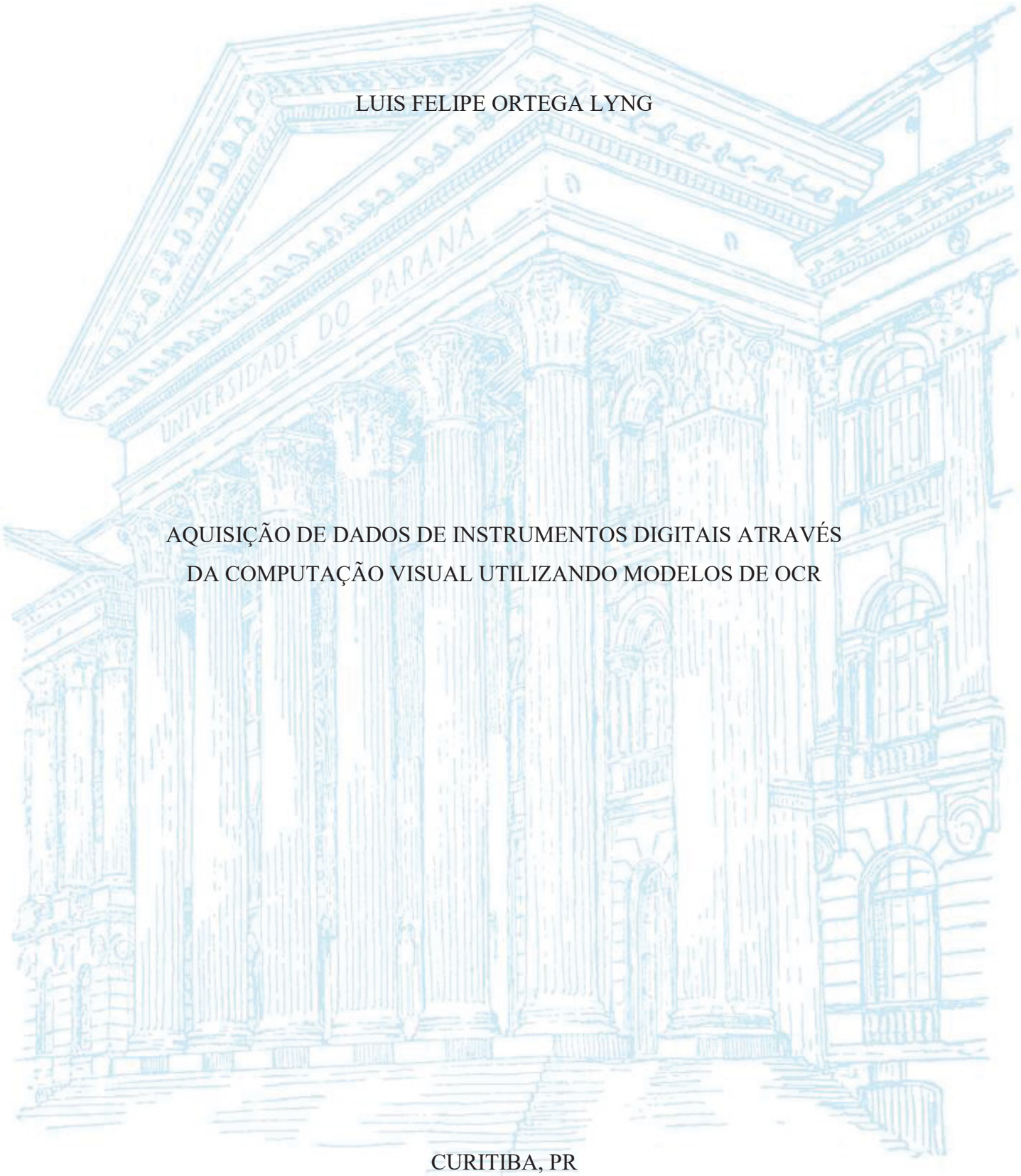
UNIVERSIDADE FEDERAL DO PARANÁ

LUIS FELIPE ORTEGA LYNG



AQUISIÇÃO DE DADOS DE INSTRUMENTOS DIGITAIS ATRAVÉS DA COMPUTAÇÃO VISUAL UTILIZANDO MODELOS DE OCR

CURITIBA, PR

2023

LUIS FELIPE ORTEGA LYNG

AQUISIÇÃO DE DADOS DE INSTRUMENTOS DIGITAIS ATRAVÉS DA COMPUTAÇÃO VISUAL UTILIZANDO MODELOS DE OCR

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Especialista em Inteligência Artificial Aplicada no Programa de Pós-Graduação em Inteligência Artificial Aplicada, Setor de Educação Profissional e Tecnológica, da Universidade Federal do Paraná.

Orientador: Prof. Dr. Razer Anthom Nizer Rojas Montaño.

CURITIBA, PR
2023

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação INTELIGÊNCIA ARTIFICIAL APLICADA da Universidade Federal do Paraná foram convocados para realizar a arguição da Monografia de Especialização de **LUIS FELIPE ORTEGA LYNG** intitulada: **AQUISICAO DE DADOS DE INSTRUMENTOS DIGITAIS ATRAVES DA COMPUTACAO VISUAL UTILIZANDO MODELOS DE OCR**, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua **APROVAÇÃO** no rito de defesa.

A outorga do título de especialista está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 14 de julho de 2023.

RAZER ANTHOM NIZER ROJAS MONTAÑO
Presidente da Banca Examinadora

JAIME WOJCIECHOWSKI
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Aquisição de dados de instrumentos digitais através da computação visual utilizando modelos de OCR

Luis Felipe Ortega Lyng
Setor de Educação Profissional e Tecnológica
Universidade Federal do Paraná
Curitiba, Brasil
felipeortega1984@gmail.com

Razer Anthom Nizer Rojas Montaño
Setor de Educação Profissionla e Tecnológica
Universidade Federal do Paraná
Curitiba, Brasil
razer@ufpr.com

***Resumo* — O presente estudo tem por objetivo apresentar os procedimentos para a realização do reconhecimento visual de números em *display* de equipamentos de medição eletrônica através da computação visual. Os modelos de OCR (*Optical Character Recognition*) utilizam redes neurais muito evoluídas que são capazes de reconhecer com muita precisão qualquer número. O método apresentado neste trabalho não é invasivo, não necessita de protocolos de comunicação serial, como também, não necessita de modificações no equipamento a ser mensurado. As imagens são capturadas por meio de 2 câmeras em formato HD 720p e 30 *frames* por segundo, que são instaladas em frente a cada *display* do equipamento. As imagens capturadas tratadas pelas bibliotecas *Opencv* em linguagem de programação *Python*, serão posteriormente inseridas em 3 modelos de OCR, *Pytesseract, EasyOCR* e *PaddleOCR*. Os modelos reconhecem os números e geram um *dataframe* no qual os dados coletados serão tratados e refinados pela biblioteca *Pandas*. O objetivo é gerar um relatório final no formato de gráficos com auxílio da biblioteca *Matplotlib*. Em vista disso, será realizado um estudo comparativo entre *Paddle OCR, Pytesseract e Easy OCR* apresentando os níveis de confiança de 99.79%, 93.07% e 85.09% respectivamente.**

Palavras-chaves* — *Computação visual, OpenCV, Pytesseract, EasyOCR, PaddleOCR, reconhecimento de dígitos.

Abstract — The present study aims to present the procedures for performing the visual recognition of numbers on the display of electronic measuring equipment through visual computing. OCR (Optical Character Recognition) models use very evolved neural networks that are capable of very accurately recognizing any number. The method presented in this work is non-invasive, does not require serial communication protocols, and also does not require changes in the equipment to be measured. Images are captured using 2 cameras in HD 720p format and 30 frames per second, which are installed in front of each equipment display. The captured images treated by the Opencv libraries in Python programming language will be later inserted into 3 OCR models, Pytesseract, EasyOCR and PaddleOCR. The models recognize numbers and generate a dataframe in which the collected data will be treated and refined by the Pandas library. The objective is to generate a final report in graph format with the help of the Matplotlib library. In view of this, a comparative study will be carried out between Paddle OCR, Pytesseract and Easy OCR, presenting confidence levels of 99.79%, 93.07% and 85.09% respectively.

Index terms* — *Computer vision, OpenCV, Pytesseract, EasyOCR, PaddleOCR, number recognition.

I. DESENVOLVIMENTO

Este trabalho foi motivado devido à necessidade em adquirir dados de equipamentos de teste de carga em baterias de LIPO ou Polímero de Lítio de *drone*. Os dados de equipamentos são importantes para a geração de relatório e gráficos que serão usados para mensurar a saúde da bateria e o seu desempenho

Os equipamentos utilizados neste trabalho são da linha *Hobby* e não dispõem de nenhuma saída para aquisição de dados em tempo real. Somente equipamentos da linha profissional possuem este recurso a custo bem mais elevado. Em visto disso, foi proposto um novo método não invasivo, no qual não houvesse a necessidade de uma conectividade serial. A ideia consiste em capturar a sequência de imagens com os dados apresentados no próprio display, para serem tratados e processados por algoritmos de OCR. Os algoritmos serão capazes de reconhecer todos os campos numéricos dos displays para armazenamento em um banco de dados e possibilitar a geração de diversos gráficos de desempenho, como por exemplo, a variação da tensão em *Volts* em função da corrente consumida pela bateria em *mili Amper Hora*. A Figura 1 apresenta um exemplo de ensaio realizado em bateria de LIPO, e de como esperar o resultado das curvas.

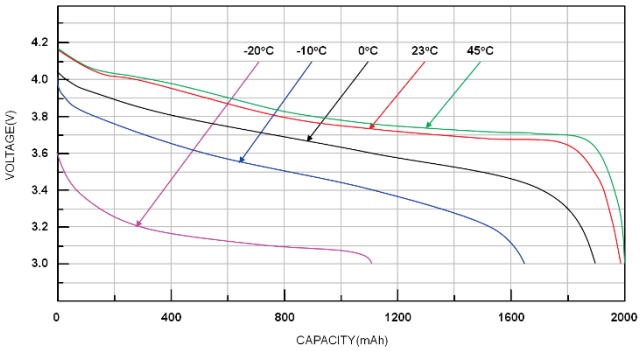


Figura 1: Exemplo de aquisição de dados em teste de carga em bateria.

Fonte: Google

A sigla ‘OCR’ provêm do inglês, *Optical Chacacter Recognition*. É uma tecnologia da computação visual que existe há muitos anos. Porém, com o avanço recente da inteligência artificial e das redes neurais, ampliou-se a possibilidade da aplicação em diversas áreas, gerando resultados com precisão.

Pytesseract

O Tesseract é uma poderosa ferramenta de OCR, que utiliza a LSTM (*Long Short-Term Memory*). Desenvolvida inicialmente pela *HP Labs* em 1985, sendo descontinuado no início dos anos 90. Em 2005, a *HP* decidiu abrir o código do *Tesseract* e lançá-lo como um projeto de código aberto sob a licença *Apache* 2.0, permitindo que desenvolvedores ao redor do mundo contribuíssem para o seu desenvolvimento. Desde 2006, a *Google* tem patrocinado o projeto, implementando significativas melhorias e expandindo o suporte para uma ampla gama de idiomas, tornando-o uma das melhores ferramentas OCR de código aberto disponíveis, utilizada para digitalização de documentos impressos até a extração de texto de imagens na *web* [1].”

O *Tesseract* é capaz de reconhecer mais de 100 idiomas do mundo e pode ser treinado para reconhecer outros idiomas ou fontes específicas. O *Tesseract* funciona lendo a imagem, identificando caracteres e palavras, e em seguida, produzindo uma saída de texto.

EasyOCR

EasyOCR é uma biblioteca de código aberto para o *Python*. A biblioteca foi desenvolvida pela *Jaided AI*, uma empresa chinesa focada em tecnologias de visão computacional.

O *EasyOCR* é capaz de lidar com mais de 80 línguas, incluindo muitos *scripts* não latinos. Além disso, o *EasyOCR* suporta a leitura de textos em imagens com orientação vertical, um recurso que não é suportado por todas as bibliotecas OCR.

Toda a execução de *Deep Learning* é baseada em *Pytorch*. A execução de detecção utiliza o algoritmo *CRAFT*. O modelo de reconhecimento é um CRNN, e é composto por 3 componentes principais: extração de características que utiliza o *Resnet* e VGG, rotulação de sequência LSTM e decodificação CTC. O pipeline de treinamento para execução de reconhecimento é uma versão modificada do framework *deep-text-recognition-benchmark* [2].

PaddleOCR

PaddleOCR é uma biblioteca OCR baseada em *deep learning* desenvolvida pela equipe *PaddlePaddle*, que é a plataforma de aprendizado de máquina de código aberto da *Baidu*, na China, que suporta mais de 80 idiomas.

Uma das vantagens do *PaddleOCR* é que ele não apenas oferece modelos pré-treinados para tarefas de OCR com desempenho no estado da arte, mas também suporta a personalização e treinamento de modelos em novos conjuntos de dados, o que o torna flexível para uma variedade de tarefas e aplicações.

O *PaddleOCR* é capaz de realizar a detecção de texto em imagens, *layout* de documentos e reconhecimento de textos e dígitos [3].

A. Descrição dos equipamentos

A seguir é apresentado um pouco mais sobre os equipamentos utilizados neste experimento.

O testador de carga eletrônica *East Tester* ET-5010, mostrado na Figura 2, é responsável por drenar a carga da bateria em corrente contínua [4].

O ET5410 monitora as seguintes funções:

- Tensão em *Volts*;
- Corrente de descarga em *Amper*;
- Potência dissipada em *Watts*;
- Capacidade de corrente em *Amper hora*;
- Energia dissipada em *Watts hora*;

O balanceador de carga *ToolkitRC* M6D, mostrado na Figura 2, é responsável por carregar as células da bateria de forma balanceada antes mesmo do teste de carga, como também é responsável por monitorar o nível de tensão individual de cada célula individualmente [5].

Figura 2: Equipamentos eletrônicos para aquisição de dados
Fonte: O autor (2023)

A bateria utilizada é do modelo de drone DJI *Phantom* 4 Pro e possui as seguintes características [6].

- Possui 4 células de 4,35 V cada;
- Tensão máxima de 17.4 V;
- Capacidade de corrente 5.870 mAh;
- Energia 89,2 Wh;
- Autonomia de voo de 30 minutos;

Figura 3: Bateria do experimento
Fonte: SZ DJI *Technology* Co

A Figura 4 apresenta a montagem do experimento em bancada com todos os equipamentos descritos anteriormente.

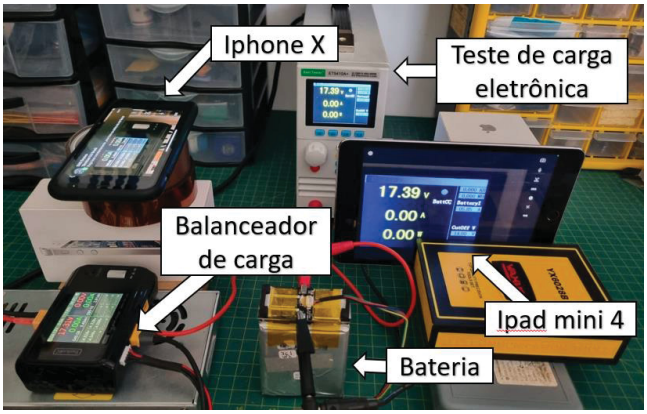


Figura 4: *Setup* dos equipamentos
Fonte: O autor (2023)

B. Métodos

A aquisição de dados, tratamento e criação de modelos de aprendizado de máquina, são técnicas muito utilizadas na ciência de dados, e existem duas abordagens:

Model-Centric (Centrada no Modelo): Nesta abordagem, a ênfase é colocada no desenvolvimento e ajuste do modelo. Isto envolve a escolha do tipo de modelo a ser usado (por exemplo, redes neurais, árvores de decisão etc.), ajuste dos hiperparâmetros, seleção de recursos, etc. A ideia subjacente é de que, com o modelo e os ajustes certos, o sistema poderá aprender efetivamente a partir dos dados disponíveis. Esta abordagem é comum em muitas competições de aprendizado de máquina, nas quais o objetivo é obter o melhor desempenho possível em um conjunto de dados específico. [7].

Data-Centric (Centrada nos Dados): Nesta abordagem, a ênfase é colocada na coleta, limpeza, anotação e engenharia de dados. A ideia é que, com dados suficientemente bons e relevantes, modelos relativamente simples podem produzir resultados eficazes. Esta abordagem é especialmente útil em situações do mundo real, onde os dados podem estar desorganizados, incompletos ou não representativos do problema que se deseja resolver [7].

A abordagem ***Data-Centric***, será a escolha para esse experimento.

Para facilitar o entendimento deste experimento, foram criados quadros com seus atributos que estarão disponíveis nos scripts em linguagem *Python* e nos fluxogramas de cada processo.

Primeiramente, deve-se seguir a ordem correta das instalações dos pacotes e bibliotecas *Python*, independentemente do Sistema operacional, *Windows*, *MAC* ou *Linux*.

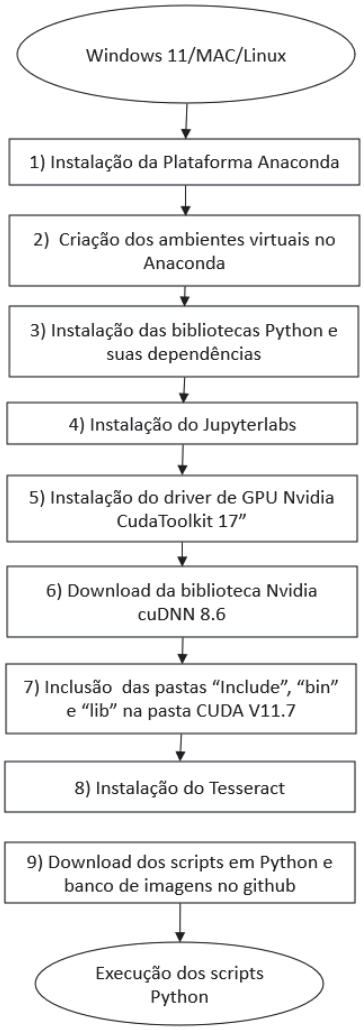


Figura 5: Fluxograma para instalação dos pacotes *Python* e suas dependências.
Fonte: O autor (2023)

Os ambientes virtuais são criados dentro da Plataforma *Anaconda*, e têm por finalidade isolar as bibliotecas *Python* dos outros ambientes, isso evita problemas com incompatibilidades durante a execução dos *scritps*. Na sequência, criam-se 4 ambientes virtuais dedicados a executar

cada *script* (*Cameras, Pytesseract, EasyOCR, PaddleOCR)* conforme listado no Quadro I:

Quadro I
INSTALAÇÃO DAS BIBLIOTECAS PYTHON

Biblioteca	***Cameras***	***Tesseract***	***EasyOCR***	***PaddleOCR***
Python + pacote	*Sim*	*Sim*	*Sim*	*Sim*
Numpy	*Sim*	*Sim*	*Sim*	*Sim*
Pandas	*Não*	*Sim*	*Sim*	*Sim*
Matplotlib	*Não*	*Sim*	*Sim*	*Sim*
Scikit-Learn	*Não*	*Sim*	*Sim*	*Sim*
Pytorch	*Não*	*Não*	*Sim*	*Não*
CUDA Toolkit	*Não*	*Não*	*Sim*	*Sim*

Algumas bibliotecas específicas que não foram citadas no Quadro I, foram inseridas no cabeçalho de cada *script*.

A instalação do driver da GPU (*Graphics Processing Unit*) CUDA *Toolkit* deve ser feita através do site oficial da *Nvidia* [8] e deve-se seguir o mesmo procedimento para baixar a biblioteca cuDNN [9]. Para isso é necessária uma versão correta do *driver*, conforme o modelo da placa de vídeo listada no site da *Nvidia*.[10].

Neste experimento foi utilizado uma GPU *Nvidia* RTX 3060 compatível com o *driver* CUDA *Toolkit* V11.7 para o *Windows* 11. A biblioteca de redes neurais para computação paralela recomendada é a cuDNN 8.6, neste caso conforme o requisito do fabricante da placa de vídeo [10].

C. *Captura das imagens*

Inicialmente houve a intenção em utilizar 2 *webcams* devido ao baixo custo. Mas na testagem percebeu-se que a distância focal era bem maior e que não atendia a esse requisito no experimento. Portanto optou-se pelo uso de um *smartphone* e um *Tablet* visto que estavam disponíveis.

A captura da imagem pode ser feita por qualquer câmera digital desde que atenda os seguintes requisitos:

- Resolução mínima em HD 720p e 30 FPS;
- Distância focal mínima a partir de 10 cm;
- Controle de foco e velocidade do obturador.

As câmeras precisam estar configuradas na resolução 720p e 30 FPS *(Frames* por segundo), no modo manual, com controle de foco e balanço de branco automático desabilitados. Foi ajustado o filtro ISO para 400, velocidade do obturador em 1/60. A transmissão de imagem ocorreu simultaneamente entre o computador e as duas câmeras pelo Wifi 5G através do aplicativo *Iriun.*

Iriun é um aplicativo que converte o s*martphone* Android ou IOS em *Webcam*. O sinal é transmitido pela rede WIFI. A latência do sinal é muito baixa, praticamente inexistente, desde que seja garantido estabilidade de conexão de rede. Este aplicativo é gratuito e pode ser instalado em *Windows* ou *MAC* ou *Linux* [11].

A composição da imagem exige que as câmeras estejam posicionadas de forma ortogonal ao *display* do equipamento eletrônico. Além de garantir a distância mínima de 10 cm do *display* para não haver perda de foco. O *script* `Cameras` possui uma função criada para gerar linhas de grade 5x5 na cor branca, isso auxilia no processo de alinhamento das câmeras, basta pressionar a tecla "G" para habilitar ou desabilitar essa função.

Deve-se tomar certos cuidados durante a captura das imagens, e evitar ao máximo a incidência de luz solar ou artificial diretamente sobre os *displays*, pois isso gera reflexo e prejudicará a qualidade das imagens capturadas. Além disso, o ambiente precisa ter luminosidade controlada, pois as câmeras operarão no modo manual sem auto compensação de exposição de luz.

Para otimizar a captura das imagens, foi delimitado por regiões de interesse (ROI) de cada *display*. Isso é definido através da criação de coordenadas retangulares desenhadas diretamente sobre a janela de captura de vídeo, com algumas funções da biblioteca *OpenCV* que foram incorporadas no *script* `Cameras`. Esse método aumenta a precisão e reduz drasticamente a captação de ruídos indesejados.

Figura 6: Captura de imagem do display balanceador de carga M6D
Fonte: O Autor (2023)

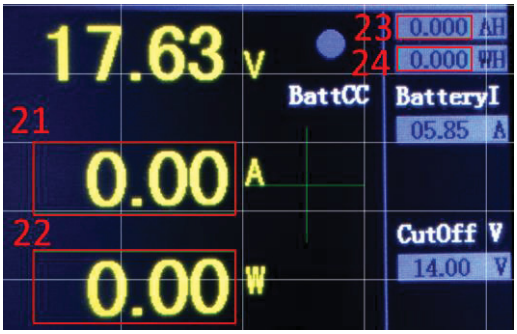


Figura 7: Captura de imagem do *display* do ET-5410
Fonte: O Autor (2023)

D. Tratamento das imagens com OpenCV

Cada ROI foi representado na Figura 6 e 7 através de retângulos em vermelho, estes podem ser verificados através dos índices no Quadro II.

Quadro II

ATRIBUTOS DA BASE DE DADOS INPUT

Atributo	Descrição Tratamento das imagens script Cameras
ROI[xy]	Região de interesse da imagem
ROI[11]	Camera 1, campo 1 “Voltage [V]”
ROI[12]	Camera 1, campo 2 “Cell_1 [V]”
ROI[13]	Camera 1, campo 3 “Cell_3 [V]”
ROI[14]	Camera 1, campo 4 “Cell_2 [V]”
ROI[15]	Camera 1, campo 5 “Cell_4 [V]”
ROI[21]	Camera 2, campo 1 “Current [A]”
ROI[22]	Camera 2, campo 2 “Power [W]”
ROI[23]	Camera 2, campo 3 “Capacity [Ah]”
ROI[24]	Camera 2, campo 4 “Energy [Wh]”
[x,y,w,h]	Coordenadas dos retângulos dos ROIs
frame	Leitura dos *frames* com imagem e mostrada em janela
RGB	Imagem em 3 canais de cores (vermelho, verde e azul)
gray	Conversão do frame de RGB para escala de cinza
clahe	Equalização do histograma em escala de cinza
dst	Redução de ruido da leitura com filtro Clahe
cam	Resultado da imagem recortada pelas coordenadas do ROI
tresh	Binarização da imagem recortada pelo ROI
tresh_inv	Binarização inversa da imagem recortada pelo ROI
blur	Filtro *blur*, desfoque na imagem binarizada
resize	Redimensionamento do filtro da imagem binarizada
border	Acréscimo de bordas brancas de todas as imagens

Fonte: O autor (2023)

Imagens são matrizes de pixels, com uma tupla de 3 canais RGB (vermelho, verde e azul) variando em uma escala de cores em 8 bits, de 0 a 255 em números inteiros.

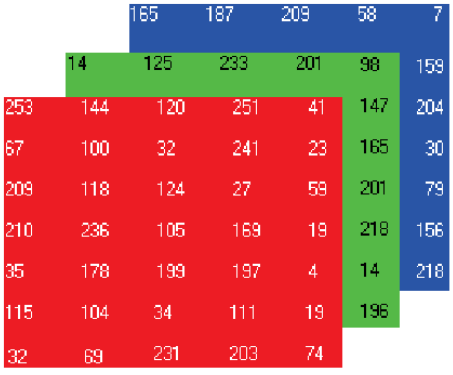


Figura 8: Matriz de imagem colorida
Fonte: Opencv.org

Uma imagem em escala de cinza possui uma matriz única variando em tons de cinza de 0 a 255, ou seja, varia do preto até o branco.

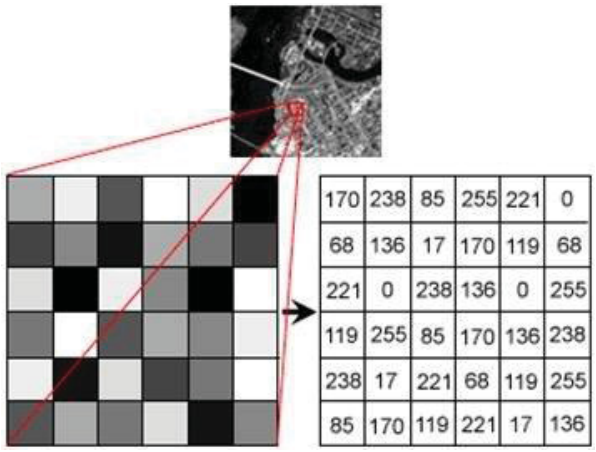


Figura 9: Matriz de imagem em escala de cinza
Fonte: Opencv.org

A seguir é mostrado como funciona o processo do tratamento de imagem deste experimento.

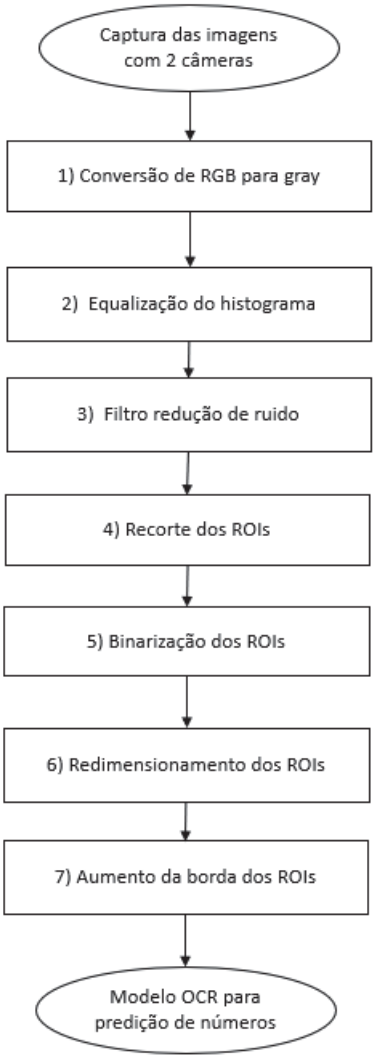


Figura 10: Fluxograma para tratamento de imagem.
Fonte: O autor (2023)

A função `cv2.VideoCapture` é usada para capturar imagem de vídeo das câmeras representado na etapa de entrada da Figura 10, também é utilizado para abrir arquivos de vídeo em formato MP4, AVI ou MOV.

```
Cap1 = cv2.VideoCapture(0) #Captura imagem câmera 1
Cap2 = cv2.VideoCapture(1) #Captura imagem câmera 2
```

O índice "0" se refere ao canal de captura de vídeo, neste caso utilizamos o índice 0 e 1 para capturar imagem das duas câmeras simultaneamente.

A função `cv2.COLOR_RGB2GRAY` representado na primeira etapa da Figura 10, é usada para converter uma imagem colorida RGB em uma imagem em escala de cinza. Reduzimos a matriz composta de 3 canais de cores para apenas um canal de 256 tons de cinza [12].

A conversão dos 3 canais de cores ocorre seguindo a equação do código abaixo:

```
Y = 0.299*R + 0.587*G + 0.114*B
```

Y é o valor da escala de cinza de cada pixel da matriz da imagem colorida RGB [12].

```
Clahe =
cv2.createCLAHE(clipLimit=2,tileGridSize=(5, 5))

equalized = clahe.apply(gray)
```

A função `cv2.createCLAHE` representada na segunda etapa da Figura 10, é usada para implementar uma técnica chamada CLAHE (*Contrast Limited Adaptive Histogram Equalization*), que é um método para melhorar o contraste de uma imagem [13].

A equalização do histograma é uma técnica que busca redistribuir os valores dos *pixels* de uma imagem de maneira que seu histograma (um gráfico que mostra a distribuição de intensidade dos *pixels*) seja aproximadamente uniforme, ou seja, que haja aproximadamente o mesmo número de *pixels* de cada intensidade de cinza. Isso tem o efeito de maximizar o contraste da imagem, um exemplo é mostrado na Figura 11.

No entanto, a equalização do histograma tradicional aplica a mesma transformação a toda a imagem, o que pode aumentar o ruído e alterar os contrastes locais. A CLAHE resolve isso dividindo a imagem em pequenos blocos ou `tileGridSize` (neste exemplo 5x5), e aplicando a equalização do histograma a cada um deles separadamente. Isso garante que o contraste seja aumentado localmente, preservando melhor os detalhes e reduzindo o ruído, na Figura 12 nota-se bem a diferença na equalização.

Além disso, a limitação de contraste, `clipLimit` na CLAHE refere-se a uma intensidade que é aplicada durante a equalização do histograma para evitar que as diferenças de contraste se tornem muito grandes. Se algum histograma de bloco tiver um pico muito grande (indicando muitos *pixels* de uma certa cor), a limitação de contraste reduzirá esse pico, limitando a quantidade de contraste que pode ser aplicada.

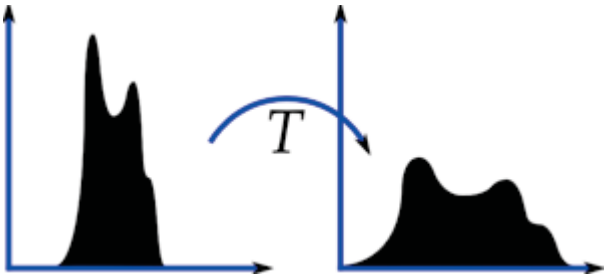


Figura: 11: Transformação do histograma de cinza CLAHE
Fonte: Opencv.org

Figura 12: Comparação do resultado do filtro CLAHE
Fonte: Opencv.org

A função `cv2.fastNlMeansDenoising`, representada na terceira etapa do fluxograma é usada para remover o ruído de uma imagem usando o método *Non-local Means Denoising*. Este método é uma técnica eficaz para eliminar o ruído, especialmente o ruído gaussiano, da imagem [14].

O método *Non-local Means Denoising* funciona comparando todos os *pixels* na imagem, em vez comparar apenas os vizinhos mais próximos como na maioria das outras técnicas de *denoising*. Para cada *pixel*, uma vizinhança é definida em torno dele. Em seguida, essa vizinhança é comparada com todas as outras vizinhanças na imagem e uma média ponderada é calculada para obter o novo valor do *pixel*. Os pesos para a média são calculados com base na similaridade entre as vizinhanças [14].

A versão rápida deste método, implementada em `cv2.fastNlMeansDenoising`, utiliza um algoritmo otimizado que é significativamente mais rápido do que o original, especialmente para imagens maiores, mas ainda oferece resultados semelhantes [14].

```
dst = cv2.fastNlMeansDenoising(img, None, 10, 7, 21)
```

Os parâmetros passados controlam a força da *denoising* (10 neste exemplo), o tamanho da vizinhança (7 neste exemplo), e o tamanho da janela de pesquisa (21 neste exemplo). Esses valores podem ser ajustados dependendo das características específicas da sua imagem e do quanto de ruído você precisa remover.

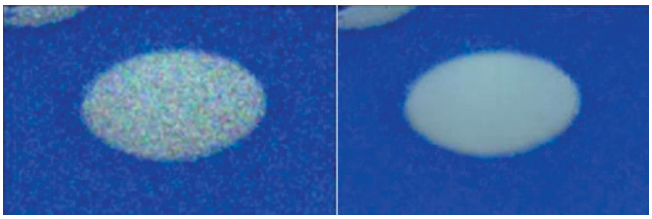


Figura 13: Resultado da aplicação do filtro *Fast Denoising*
Fonte: Opencv.org

A captura das regiões de interesse ROI, representada na quarta etapa do fluxograma, é feita de forma simples pela biblioteca OpenCV, bastando indicar as coordenadas.

```
Cam11 = dst1[y0:y0+h0, x0:x0+w0]
```

Sendo `Cam11`, o resultado da imagem capturada pelas coordenadas de cada retângulo na imagem `dst1` (resultado do filtro *Fast Denoising*).

A função `cv2.threshold`, representada na etapa 5 do fluxograma, é usada para aplicar um limiar, *threshold* em uma imagem binarizada, o que é útil para operações como segmentação de imagem e detecção de bordas [15].

A segmentação de imagem é um passo comum no processamento de imagens, onde uma imagem é dividida em múltiplas regiões, e cada uma das quais contendo *pixels* com propriedades similares. Um dos métodos mais simples de segmentação é a binarização de imagem, que é onde `cv2.threshold` é frequentemente usado [15].

A função `cv2.threshold` recebe uma imagem em escala de cinza e um valor de limite como entrada, juntamente com um valor máximo e um tipo de *thresholding*. Cada *pixel* na imagem é comparado com o valor do limite.

Se o valor do *pixel* for maior que o valor do limite, o *pixel* é definido para o valor máximo (geralmente 255, indicando branco).

```
ret,thresh=cv2.threshold(img,127,255,cv2.THRESH_BIN
ARY)
```

Se o valor do *pixel* for menor ou igual ao valor do limite, o *pixel* é definido para 0 (indicando preto). O tipo de *thresholding* pode ser definido para realizar diferentes tipos (como *thresholding* direta e inversa.) [15].

```
ret,threshinv=cv2.threshold(img,127,255,cv2.THRESH_
BINARY_INV)
```

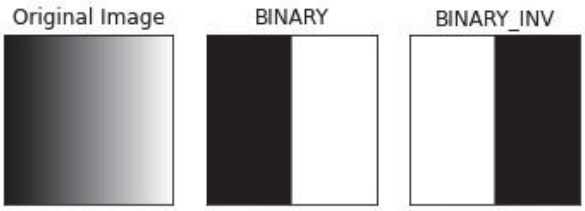


Figura 14. Aplicação do *tresholding* numa imagem em escala de cinza
Fonte: Opencv.com

A função `cv2.resize`, representada na etapa 6 do fluxograma, é usada para redimensionar imagens. Essa função pode aumentar ou diminuir o tamanho de uma imagem, alterando o número de *pixels*. Pode utilizar diferentes métodos de interpolação para preencher os valores dos pixels quando a imagem é redimensionada. A interpolação é usada para estimar os valores dos *pixels* em um novo tamanho de imagem com base nos valores dos pixels na imagem original [16].

A seguir está um exemplo de uso do `cv2.resize`:

```
resize11 = cv2.resize(treshinv12, None, fx = 2.0,
fy = 2.0, interpolation = cv2.INTER_LINEAR)
```

onde `fx` e `fy` se refere ao fator de escala na ampliação do frame, neste exemplo é proporcional ao dobro do tamanho.

O parâmetro `cv2.INTER_LINEAR` é um método de interpolação bilinear (usado por padrão). Ele usa os quatro *pixels* mais próximos para estimar o valor do *pixel* interpolado [16].

A função `cv2.BORDER_CONSTANT`, representada na sétima etapa do fluxograma, é um dos modos que você pode especificar na função de preenchimento de borda da imagem, como `cv2.copyMakeBorder()` ou `cv2.resize()`. Este modo adiciona uma borda constante à imagem, o que significa que os *pixels* adicionados têm um valor de cor específico que você define, no caso foi escolhido branco. A finalidade é compensar a margens das capturas dos ROIs que ficaram com muito estreitas [17].

Por fim, a última etapa e função do fluxograma, `cv2.imwrite` salva os *frames* numa pasta destino `root_path`.

```
cv2.imwrite(root_path+"/img11/img11_{}.jpg".format(
count), border11)
```

A função `Count`, executa a contagem dos *frames* salvos da captura num intervalo de tempo estipulado de 5 segundos. Cada *frame* é salvo numa pasta `root_path` seguindo uma sequência numérica iniciando a contagem do zero até o último *frame* [18].

Para resumir todos esses processos descritos, o resultado obtido do tratamento de imagem ficou similar a Figura 15.

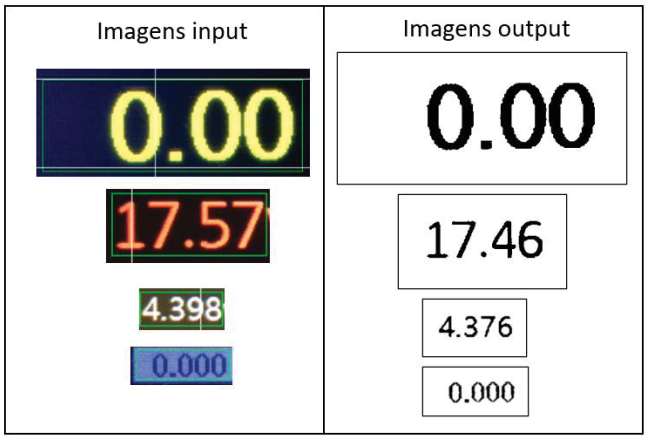


Figura 15. Resultado do tratamento de imagem pelo OpenCV
Fonte: O autor (2023)

Após ajustado todos os parâmetros no tratamento de imagem, deu-se início a captura das imagens do experimento, que teve uma duração de 29 minutos.

A partir das capturas de 1 *frame* a cada 5 segundos, foi gerado um banco de imagens composto por 9 pastas contendo 351 imagens em cada pasta, totalizando 3.159 imagens e em 20,14 Mb de espaço.

Nota-se que as imagens da Figura 15 possuem variações de tamanho e qualidade de imagem. A qualidade das imagens varia entre alta, média e baixa definição, o que torna uma base de dados com uma boa variância para o experimento.

E. *Processamento dos modelos de OCR*

Os modelos de OCR necessitam de parametrização para funcionar corretamente, pois apresentam peculiaridades no processamento dos dados de saída.

PYTESSERACT

A função `pytesseract.image_to_data()` da biblioteca *PyTesseract* é usada para extrair informações detalhadas sobre a estrutura da imagem. Ela não apenas reconhece o texto da imagem, mas também fornece informações como o número de palavras, a localização de cada palavra na imagem e o nível de confiança do reconhecimento de cada palavras.

```
pytesseract.image_to_data(img, config='--psm 6
digits', output_type=Output.DICT)
```

O primeiro parâmetro para `image_to_data()` é a imagem na qual o OCR deve ser realizado e `img` representa o caminho.

O segundo parâmetro, `config`, é usado para passar as configurações para o motor *Tesseract*. No exemplo fornecido, `--psm 6`, define o modo de segmentação da página para assumir um único bloco de texto uniforme. `Digits` é uma configuração adicional que informa ao *Tesseract* para reconhecer apenas dígitos.

O terceiro parâmetro, `output_type`, especifica o tipo de saída que você deseja. O *PyTesseract* pode retornar a saída como uma *string*, como uma lista de caixas delimitadoras, semelhante a um dicionário. No exemplo fornecido, `Output.DICT` indica que você deseja que a saída seja um dicionário.

A função `Ouput.DICT` retorna um *dataframe* com coordenadas do *bounding box*, nível de confiança e valor numérico ou texto encontrado, conforme é mostrado o *datrame* na Tabela I.

A coluna `conf` do *dataframe* de saída do *Pytesseract* se refere ao nível de confiança do modelo, e varia de -1, quando não é encontrado nenhum valor, até 96 no nível de confiança.

Tabela I
Exemplo do output *image_to_data* do *Pytesseract*

	level	page_num	block_num	par_num	line_num	word_num	left	top	width	height	conf	text
0	1	1	0	0	0	0	0	0	500	320	-1	NaN
1	2	1	1	0	0	0	148	42	229	49	-1	NaN
2	3	1	1	1	0	0	148	42	229	49	-1	NaN
3	4	1	1	1	1	0	148	42	229	49	-1	NaN
4	5	1	1	1	1	1	148	42	229	49	96	KEEP
5	2	1	2	0	0	0	135	235	255	64	-1	NaN
6	3	1	2	1	0	0	135	235	255	64	-1	NaN
7	4	1	2	1	1	0	135	235	255	64	-1	NaN
8	5	1	2	1	1	1	135	235	255	64	96	RIGHT

Fonte: Google (2023)

EASYOCR

O método `readtext()` da biblioteca *EasyOCR* é usado para reconhecer texto em uma imagem.

```
reader = easyocr.Reader(['en'])
result = reader.readtext(img, detail=1)
```

O primeiro parâmetro para `readtext()` é a imagem na qual o OCR deve ser realizado. Pode-se passar a imagem como um *array numpy* (por exemplo, uma imagem carregada usando *OpenCV*, ou como uma *string* contendo o caminho para a imagem).

O segundo parâmetro, `detail` é usado para especificar o nível de detalhe da saída. Se `detail` for 1 (padrão), a função retornará uma lista de tuplas, cada uma contendo as coordenadas da caixa delimitadora do texto, o texto reconhecido e o nível de confiança do reconhecimento.

A seguir temos um exemplo de como é fornecido o output do modelo *EasyOCR*.

```
[([[216, 76], [326, 76], [326, 104], [216, 104]],
'EXAMPLE', 0.9990473)]
```

As primeiras 4 tuplas se referem as coordenadas do *bounding box* da palavra identificada, a quinta posição se refere a predição ou valor numérico e a última posição refere-se ao nível de confiança, ou *confidence score*.

PADDLEOCR

O método `ocr_model.ocr()` da biblioteca *PaddleOCR* é usado para reconhecer texto em uma imagem.

```
ocr_model = PaddleOCR(lang="en", use_gpu=True,
enable_mkldnn=True)

result = ocr_model.ocr(image_path)
```

O parâmetro `image_path` é o caminho para a imagem na qual se deseja realizar o OCR. A imagem pode ser um arquivo local no sistema de arquivos do computador ou uma URL de uma imagem na *Internet*.

Os parâmetros para a função `PaddleOCR()` especificam várias opções para o objeto OCR:

O parâmetro `lang=` especifica o idioma do texto a ser identificado na imagem. Por exemplo, `'en'` para inglês, `'ch'` para chinês. *PaddleOCR* suporta muitos idiomas, e você pode até passar vários idiomas de uma vez, como `lang='en|ch'` para reconhecer tanto o inglês quanto o chinês na mesma imagem.

O parâmetro `use_gpu` é um valor booleano que determina se o *PaddleOCR* deve usar a GPU para acelerar as operações de OCR, se disponível. Definir isso como `True` pode resultar em desempenho mais rápido, mas requer uma GPU compatível. Se nenhuma GPU estiver disponível ou se este parâmetro for `False`, o *PaddleOCR* usará a CPU (*Central Processing Unit*) para as operações de OCR.

O parâmetro `enable_mkldnn` é um valor booleano que, quando for `True`, habilita o uso do MKL-DNN (*Math Kernel Library for Deep Neural Networks*). O MKL-DNN é uma biblioteca de *software* da Intel que acelera o desempenho das operações de aprendizado de máquina em CPUs Intel. Isso pode melhorar significativamente a velocidade e a eficiência das operações de OCR ao usar a CPU.

A saída da função `ocr()` é uma lista de listas. Cada lista interna contém uma ou mais listas, onde cada lista representa uma palavra reconhecida na imagem. A primeira entrada na lista são as coordenadas da caixa delimitadora ao redor da palavra, a segunda entrada refere-se ao nível de confiança da detecção da palavra e a terceira entrada é a palavra detectada.

```
[[(140.0, 39.0), (242.0, 39.0), (242.0, 79.0),
(140.0, 79.0)], 0.98356, 'Test']
```

F. *Tratamento dos dados com Pandas*

Nesta sessão, serão tratados somente os dados e *dataframes* utilizados na biblioteca Pandas do Python. Considerando o Quadro II e o fluxograma da Figura 17 é possível situar-se bem em todo processo que será idêntico para os 3 modelos.

Quadro II
ATRIBUTOS DA BASE DE DADOS OUTPUT MODELOS

Atributo	Descrição dados Pytesseract, EasyOCR e PaddleOCR
idx[xy]	Indice do campo ROI
df[xy]	*Dataframe* output dos modelos OCR
df11	*Dataframe* predições e nível de confiânca “Voltage [V]”
df12	*Dataframe* predições e nível de confiânca “Cell_1 [V]”
df13	*Dataframe* predições e nível de confiânca “Cell_3 [V]”
df14	*Dataframe* predições e nível de confiânca “Cell_2 [V]”
df15	*Dataframe* predições e nível de confiânca “Cell_4 [V]”
df21	*Dataframe* predições e nível de confiânca “Current [A]”
df22	*Dataframe* predições e nível de confiânca “Power [W]”
df23	*Dataframe* predições e nível de confiânca “Capacity [Ah]”
df24	*Dataframe* predições e nível de confiânca “Energy [V]”
dfs	Concatenação de todos os dataframes no eixo x
df3	*Dataframe* tratado para geração de gráficos e relatórios

Fonte: O Autor (2023)

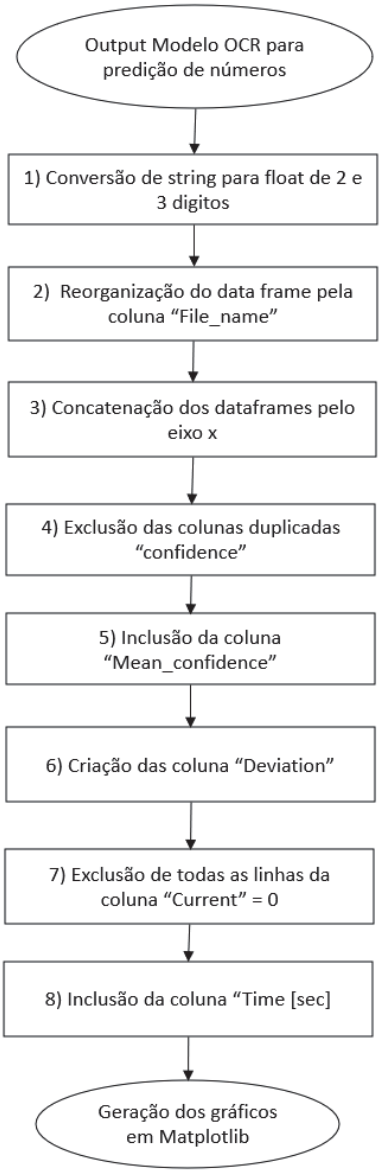


Figura 17: Fluxograma tratamento do *dataframe*.
Fonte: O autor (2023)

Neste ponto, é obtido um total de 9 *dataframes* no formato 351 x 3 resultantes dos *outpus* dos OCRs. A Tabela II mostra também as métricas e uma lista de imagens com números não reconhecidos.

Tabela II
OUTPUT DO DATAFRAME df1

	file_name	Voltage [V]	confidence
0	img11_0.jpg	17.46	96.0
1	img11_1.jpg	17.46	96.0
2	img11_2.jpg	17.46	96.0
3	img11_3.jpg	17.46	96.0
4	img11_4.jpg	17.46	96.0
...	...	...	...
346	img11_346.jpg	14.05	94.0
347	img11_347.jpg	14.03	96.0
348	img11_348.jpg	14.01	95.0
349	img11_349.jpg	13.99	95.0
350	img11_350.jpg	13.97	95.0

351 rows × 3 columns

Fonte: O autor (2023)

Conforme demonstrado na Figura 17, é necessário converter as saídas em *string* para *float* com 2 e 3 dígitos, uma vez que os dados de saída ainda não estão prontos para geração dos gráficos e relatórios.

Posteriormente, é preciso tratar os 9 *dataframes*, seguindo a sequência do fluxograma na segunda etapa, reorganizando a sequência da coluna `file_name` pelo seu sufixo numérico em ordem crescente. Iniciando na `img11_0` e finalizando na `img11_350`, depois o *dataframe* será reindexado.

```
# Extract the number from 'file_name' and convert
it to an integer

df11['file_num']=df11['file_name'].str.extract('(\d
+).jpg$', expand=False).astype(int)

# Sort the DataFrame by this new column

df11 = df11.sort_values('file_num')

# Drop the temporary column

df11 = df11.drop('file_num', axis=1)

# Reset the DataFrame's index

df11 = df11.reset_index(drop=True)
```

Na terceira etapa do fluxograma, é realizado uma concatenação pelo eixo x dos 9 *dataframes,* transformando em um único *dataframe* renomeado para `dfs`.

Seguindo para a quarta etapa, cria-se uma coluna para extrair o valor médio do nível de confiança `Mean confidence` de cada linha do `dfs` que será renomeado para `df3`.

```
# Drop columns that start with "confidence"

df3 = df3.drop(columns=[col for col in df3.columns
if col.startswith('confidence')])

# Round 'Mean confidence' column to 2 decimal places

df3['Mean_confidence']=df3['Mean_confidence'].round
(2)
```

Na quinta etapa da Figura 17, exclui-se as colunas `confidence_x` que foram iteradas para o cálculo da `Mean confidence`.

Os processos seguintes não são muito relevantes, mas serão mostrados apenas no resultado do *dataframe*. Na sexta etapa do fluxograma, será incluído a coluna `Deviation` que se refere ao valor máximo subtraído do mínimo entre as colunas `Cell_1`, `Cell_2`, `Cell_3`, e `Cell_4`. Multiplica-se o resultado por 1000 para converter para unidade em *milivolt* [mV]

```
# Assuming df3 is your original DataFrame

selected columns = df3.iloc[:, 1:5]
# columns 2,3,4,5 (0-indexed)

# Compute the max and min of each row

max_values = selected_columns.max(axis=1)

min_values = selected_columns.min(axis=1)

# Compute the deviation, multiply by 1000 and
convert to integer

deviation=((max_values-min_values)*1000)
.astype(int)

# Create a new column in df3

df3["Deviation [mV]"] = deviation
```

Inclusão da coluna `SOC [%]` (*State of charge*) significa nível de carga. Esta coluna utiliza o valor de `Voltage [V]` e realiza o seguinte cálculo:

```
# Constants for your equation

Y1 = 17.4   # Tensão máxima 100% carregada

Y0 = 13.8   # Tensão minima 0%

X1 = 100    # Nível máximo de carga

X0 = 0      # Nível mínimo de carga

# Calculate the slope and the intercept of the line

m = (X1 - X0) / (Y1 - Y0)

b = X0 - m*Y0

# Apply the function to the 'Voltage [V]' column

df3['SOC [%]'] = (m * df3['Voltage [V]'] +
b).astype(float).round(1)
```

Na sétima etapa, aplica-se um filtro em cada linha do `df3`, pois o valor da corrente de descarga é 0 *Amper*, e o nível de tensão é máximo, isso pode gerar uma falsa interpretação como *outlier* nos gráficos. Por fim, é incluído a coluna `Time [sec]` com um passo de 5 segundos para cada linha e todas as colunas foram reordenadas. E o *dataframe* `df3` final ficará conforme mostrado na Tabela III.

G. Tecnologias

Os *scripts* foram desenvolvidos e executados em um computador pessoal Dell G15 5520 com processador i7 de 14-núcleos a 4.7 GHz, placa de vídeo Nvidia RTX 3060,

32gb de RAM DD5 e 1 TB de SSD, com sistema operacional Windows 11 Home.

II. RESULTADOS E DISCUSÕES

A. Medida de qualidade dos modelos

Tabela III
Dataframe df3 resultante do modelo *PaddleOCR*

	Time [sec]	SOC [%]	Voltage [V]	Cell_1 [V]	Cell_2 [V]	Cell_3 [V]	Cell_4 [V]	Deviation [mV]	Current [A]	Power [W]	Capacity [Ah]	Energy [Wh]	Mean confidence
0	0	68.1	16.25	4.071	4.052	4.147	4.102	95	11.74	190.15	0.01	0.10	100.00
1	5	64.7	16.13	4.041	4.017	4.123	4.073	105	11.74	188.59	0.02	0.39	100.00
2	10	62.2	16.04	4.019	3.992	4.105	4.053	113	11.74	187.46	0.04	0.65	100.00
3	15	60.6	15.98	4.002	3.972	4.091	4.039	119	11.74	186.65	0.06	0.91	99.67
4	20	59.2	15.93	3.989	3.957	4.081	4.026	124	11.74	186.10	0.07	1.15	100.00

Fonte: O Autor (2023)

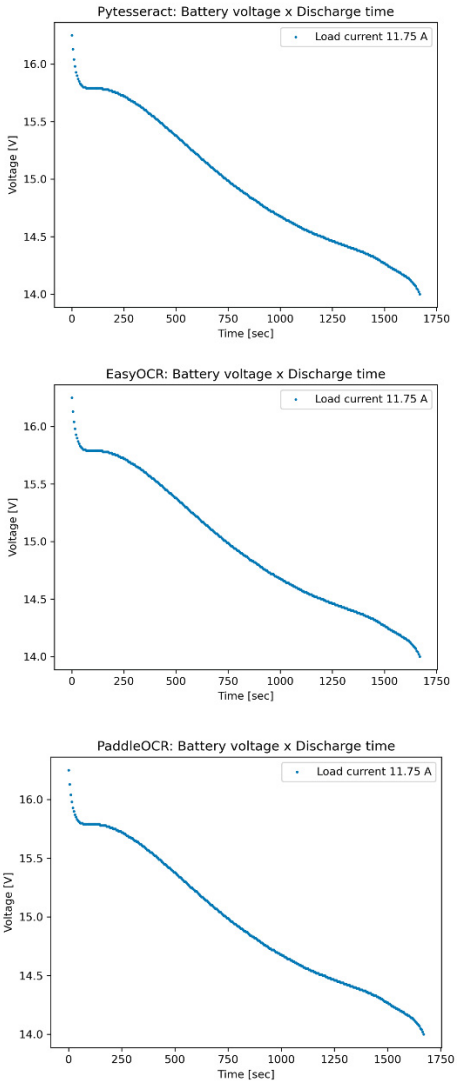


Figura 18: Gráficos Battery voltage x discharge time
Fonte: O autor (2023)

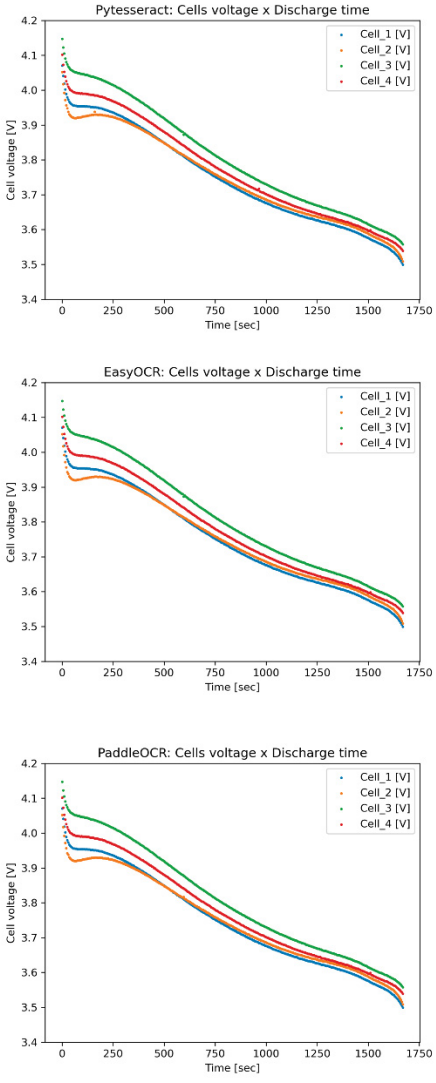


Figura 19: Gráficos Cells voltage x discharge time
Fonte: O autor (2023)

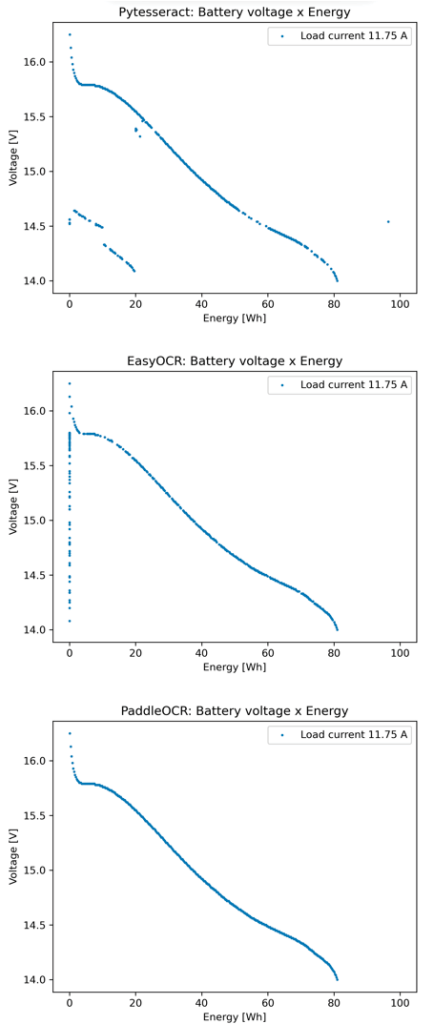


Figura 20: Gráficos Battery voltage x Energy
Fonte: O autor (2023)

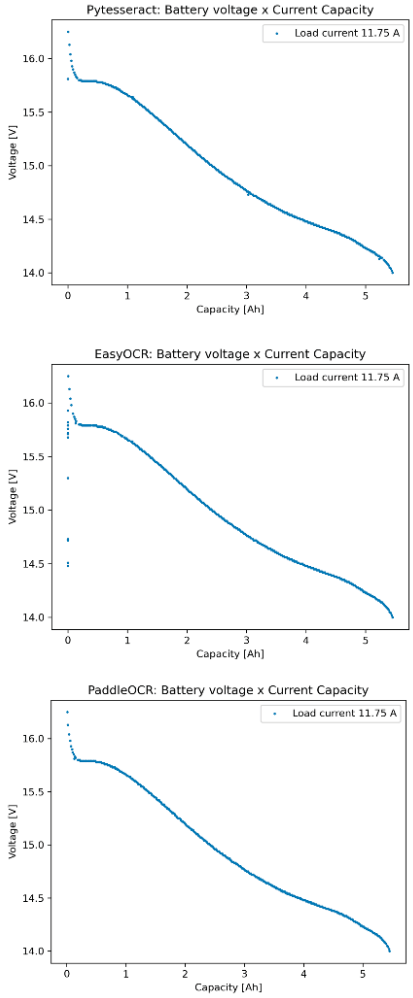


Figura 21: Gráficos Battery voltage x Capacity
Fonte: O autor (2023)

A Tabela III mostra o *dataframe* resultante depois do tratamento de dados com dimensão final de 338x13. Serão explicadas a seguir todas as colunas:

Time [sec]: Tempo de duração do experimento, com acréscimo de passo de 5 segundos por linha.

SOC [%}: é o nível de carga da bateria, valor máximo 17,4V para 100% e 14V para 0%.

Voltage [V]: é a tensão da bateria em *Volts*, iniciando com 16,25V e terminando com 14V.

Cell_1,2,3,4[V]: é a tensão em *Volts* de cada célula da bateria, possui um valor máximo de 4,35V e valor mínimo de 3,5V.

Deviation [mV]: é a diferença entre o maior valor subtraído do menor valor encontrado em cada linha nas colunas *Cell_1,2,3,4*. A unidade é a milésima parte do *Volt*, ou *milivolt.*

Current [A]: é a corrente aplicada durante o teste expressa em *Amper*, o valor é de 11,75A.

Power [W]: é a potência dissipada em *Watts*, também pode ser calculada como o produto da multiplicação entre a tensão e corrente.

Capacity [Ah]: é a capacidade de corrente consumida, é também o produto da multiplicação da corrente pelo tempo do experimento. Inicia em 0 e termina com um valor próximo ao divulgado pelo fabricante [6] que seria valor próxima a 5.870 mAh.

***Energy [Wh]*:** é a capacidade da bateria entregar energia, é também o produto da multiplicação da potência dissipada pelo tempo, expressa em *Watt.hora* e segundo o fabricante [6], possui valor máximo de 89,2 Wh;

Mean Confidence: é o valor do nível de confiança médio das colunas 3 a 10.

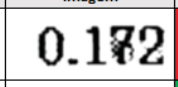
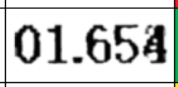
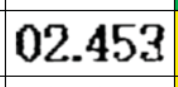
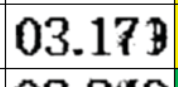
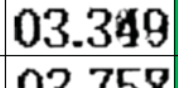
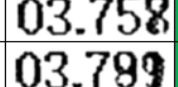
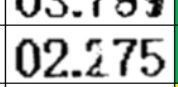
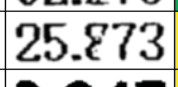
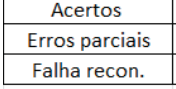
A Figura 18 apresenta o comparativo dos gráficos de cada OCR, registrando a queda de tensão total no decorrer do tempo, ao se aplicar uma carga de 11,75 A. Iniciou-se o teste com 16,25V e terminou com 14V num intervalo de aproximadamente 1.700 segundos.

A Figura 19 apresenta o comparativo dos gráficos de cada OCR, registrando a queda de tensão em cada célula da bateria no decorrer do tempo, ao se aplicar uma carga de 11,75 A. Iniciou-se o teste com 16,25V e terminou com 14V.

A Figura 20 apresenta o comparativo dos gráficos de cada OCR, registrando a queda de tensão em função da energia consumida em *Watt hora,* ao se aplicar uma carga de 11,75 A. Iniciou-se com 0 Wh e terminou com valor próximo a 82 Wh.

A Figura 21 apresenta o comparativo dos gráficos de cada OCR, registrando a queda de tensão em função da capacidade de corrente consumida em *Amper hora,* ao se aplicar uma carga de 11,75 A, iniciando em 0 Ah e terminando com valor próximo a 5.700 mAh.

Tabela IV
COMPARATIVO: IMAGEM COM RUÍDO x NÚMERO RECONHECIDO

item	File_name	imagem	Tesseract	EasyOCR	PaddleOCR
1	img23_24		0	0,132	0,132
2	img23_115		1,654	1,654	1,654
3	img23_164		2,452	2,453	2,453
4	img23_208		3,173	3,179	3,173
5	img23_219		3,349	3,349	3,349
6	img23_244		3,758	3,758	3,758
7	img23_246		3,799	3,799	3,799
8	img24_22		2,275	2,275	2,275
9	img24_115		20,073	0	25,873
10	img14_19		3,948	3,943	3,942
		Acertos	5	7	7
		Erros parciais	4	2	3
		Falha recon.	1	1	0

Fonte: O Autor (2023)

Tabela V
RESULTADOS DA QUALIDADE DOS MODELOS

	Pytesseract		EasyOCR		PaddleOCR	
	Qtde	Conf.	Qtde	Conf.	Qtde	Conf.
Voltage [V]	0	95,06	0	85,02	0	100,00
Cell_1 [V]	0	95,93	0	87,96	0	99,90
Cell_2 [V]	0	95,89	0	88,77	0	99,90
Cell_3 [V]	0	95,82	0	88,63	0	99,90
Cell_4 [V]	0	95,60	0	88,59	0	100,00
Current [A]	0	92,42	22	84,08	0	100,00
Power [W]	0	91,84	1	89,14	0	100.00
Capacity [Ah]	2	92,66	18	83,01	0	99,40
Energy [Wh]	13	82,44	69	70,65	0	99,00
Total	15	93,07	110	85,09	0	99,79

Fonte: O Autor (2023)

Na Tabela IV é possível verificar as imagens capturadas com ruído e comparado com o valor que cada modelo conseguiu reconhecer. Na última linha é apresentado a quantidade de acertos e erros de cada OCR.

Na Tabela V, a coluna "Qtde", diz respeito a quantidade de imagens com números não reconhecidos, quando o nível de confiança é igual a zero. Soma-se o valor total para cada OCR na última linha. A coluna "Conf" se refere ao nível de confiança de cada coluna do *dataframe df3* e na última linha tem-se o valor do nível de confiança global de cada modelo.

B. Desempenho dos modelos

O banco de imagem possui um total de 20,14 Mb de espaço ocupado em 3.159 arquivos e 9 pastas, e uma média de 6,375 Kb por imagem.

No final de cada *script*, foi criado um código para se obter as medidas de desempenho de cada modelo. Foi executado no total 5 experimentos, considerando 3 versões, a *Pytesseract*, *EasyOCR_CPU* e *PaddlOCR_CPU* estes foram executados no modo CPU. E mais duas versões, a *EasyOCR_GPU* e *PaddleOCR_GPU*, que foram executados no modo GPU. Com exceção do *Pytesseract* que não possui versão compatível para GPU. Os dados foram obtidos nas Tabelas VI e VII. Cada variável será explicada a seguir.

***Runtime*:** é o intervalo de tempo entre início e fim do processamento das imagens de cada OCR, expresso em segundos.

***Inferences*:** mede a velocidade em que cada imagem é processada pelo modelo, expresso em *frames* por segundo (FPS).

***Processing rate*:** é a razão entre o total de espaço ocupado pelas imagens, dividindo tempo de execução expresso em Mb/s.

Tabela VI

INDICADORES DE DESEMPENHO DOS OCR EM MODO CPU

	Tesseract	Easy_CPU	Paddle_CPU
Runtime [s]	294	1377	2279
Inferences [FPS]	10,7	2,3	1,4
Process. rate [Mb/s]	0,068	0,015	0,009

Fonte: O autor (2023)

Tabela VII

INDICADORES DE DESEMPENHO DOS OCR EM MODO GPU

	Easy_GPU	Paddle_GPU
Runtime [s]	130	426
Inferences [FPS]	24,3	7,4
Process. rate [Mb/s]	0,155	0,047

Fonte: O autor (2023)

Nas figuras 23, 24, 25 e 26, são plotados em forma de gráficos de barras os valores de cada variável para as 5 versões de modelos OCR comentado nas Tabela II, III e IV.

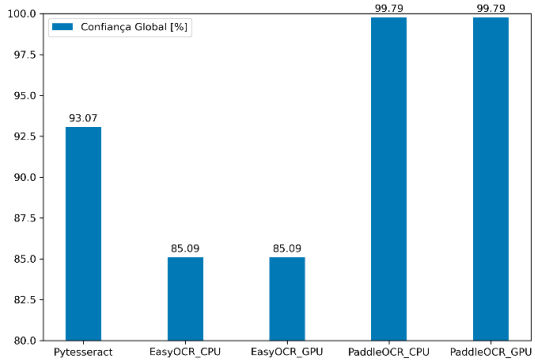


Figura 22: Comparativo entre o nível de confiança global. Fonte: O autor (2023)

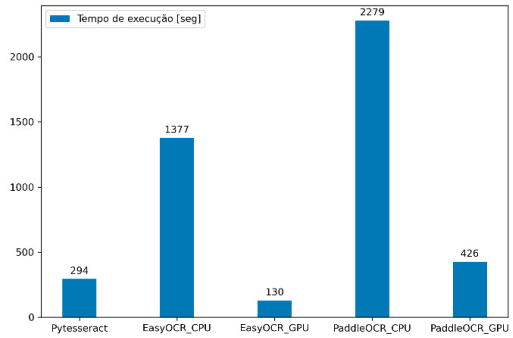


Figura 23: Comparativo entre *Runtime*, tempo de execução. Fonte: O autor (2023)

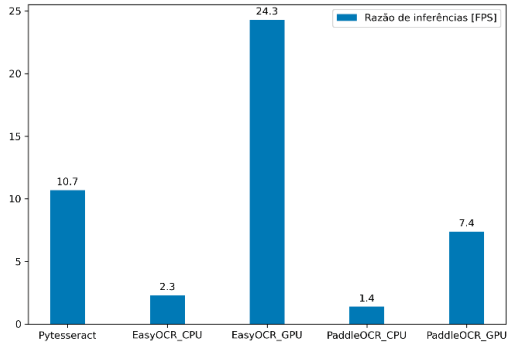


Figura 24: Comparativo entre a inferência. Fonte: O autor (2023

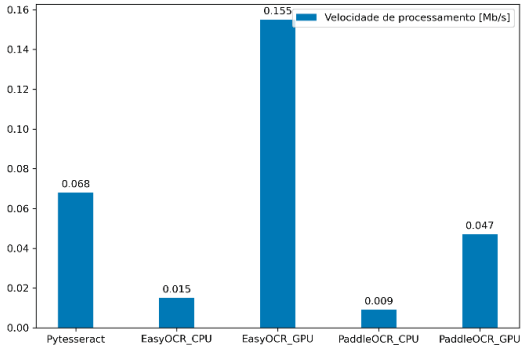


Figura 25: Comparativo velocidade de processamento. Fonte: O autor (2023)

C. *Discussões*

Antes de analisar cada caso, é necessário entender a interpretação dos *outliers* expostos nos resultados dos gráficos da Figura 27, onde existem duas situações. Os *outliers* marcados em amarelo, que representam um erro parcial no reconhecimento de algum algarismo do número, com um nível de confiança superior a zero, e também pode ser considerado como um falso positivo.

O segundo tipo de *outlier*, marcados em vermelho, são aqueles em que o modelo não consegue reconhecer nenhum número na imagem, estes assumem o valor zero no eixo x formando uma coluna no eixo y, totalizando 69 outliers, conforme mostra a Tabela IV.

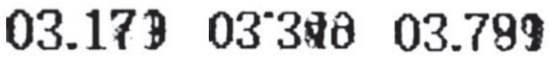


Figura 26: Exemplo de números com falhas na captura.
Fonte: O autor (2023)

O fato de haver falha na captura de alguns algarismos se deve ao sincronismo entre a captura do frame no exato momento, em que há uma mudança de um número para outro, resultando nas imagens da Figura 26.

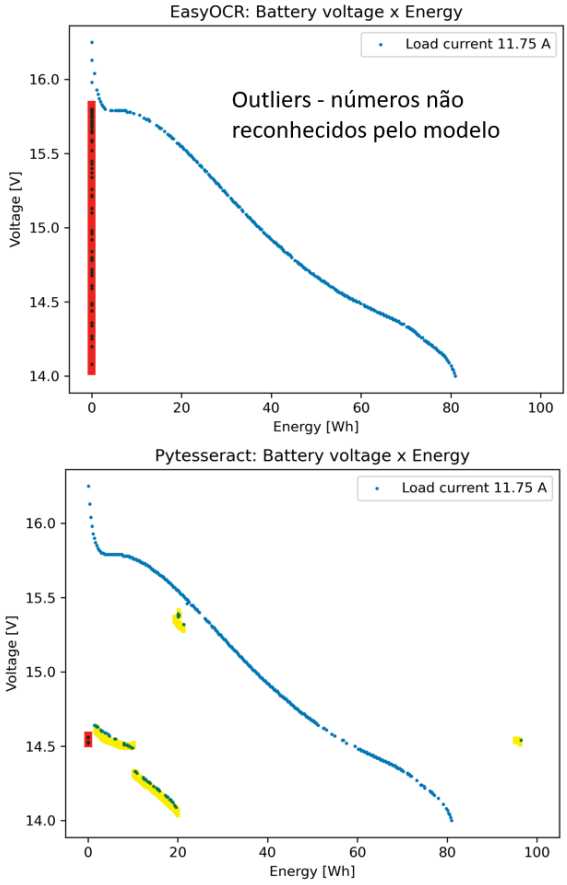


Figura 27: Gráficos com *outliers*
Fonte: O autor (2023)

Na primeira análise, não é possível afirmar que as imagens com falhas na captura reduzem o nível de confiança do OCR, pois houve mais acertos do que erros parciais, isso é demonstrado na Tabela IV. A maioria das imagens que não foram reconhecidas, foram aquelas que não apresentaram nenhuma falha na captura.

A segunda análise é sobre o desempenho dos modelos de OCR. Considerando o nível de confiança global de cada modelo, o *PaddleOCR* ficou na primeira posição com 99,79%, o *Tesseract* ficou na segunda posição com 93,07% e na terceira posição ficou *EasyOCR* com 85,09%.

Analisando a Tabela VI, em termos de performance com uso de CPU, o *Pytesseract* ficou em primeiro lugar, inferindo 10,7 FPS *frames* por segundo, a uma taxa de 0,068 Mb/s. O tempo total do processamento levou 294 segundos. Em segundo lugar ficou o *EasyOCR_CPU* inferindo 2,3 FPS com tempo de processamento de 1.377 segundos e em terceiro lugar ficou o *PaddleOCR_CPU* com 1,4 FPS e tempo de 2.279 segundos.

Quando opta-se pelo uso de GPU, *EasyOCR* ficou em primeiro lugar, apresentando um aumento considerável de performance, cerca de 10,5 vezes superior se comparado a versão com *EasyOCR* com CPU, reduzindo assim o tempo de processamento de 1.377 segundos para apenas 130 segundos. O *PaddleOCR* teve um aumento de performance de 5,3 vezes, reduzindo o seu tempo de processamento para 426 segundos.

D. Conclusão

Não existe um modelo vencedor para os quesitos apresentados. Entretanto, vale ressaltar que o nível de confiança é um fator primordial e que deve-se optar pelo modelo PaddleOCR, por garantir valores acima de 99% em nível de confiança, mesmo para imagens de baixa definição, este modelo é o estado da arte e independente se for para uso em CPU ou GPU.

Se a performance é o fator primordial, opta-se pelo *EasyOCR* com uso de GPU, assim serão obtidos ótimos resultados quando aplicado em imagens de alta e média definição.

Caso não haja disponibilidade de uma GPU, e a necessidade for de um meio termo entre nível de confiança e velocidade de processamento, deve-se optar pelo *Pytesseract*, pois este modelo reconhece imagens tanto de alta quanto de baixa definição com um nível de confiança superior a 90%.

Além disso, o *Pytesseract* é o modelo mais leve de todos, pois ocupa somente 240 Mb de espaço sem a necessidade de instalação de *driver* de vídeo, com isso se torna factível a portabilidade deste modelo de OCR para outras plataformas como *Android* e IOS.

E. Trabalhos futuros

Foram observados todos os desafios e dificuldades encontrados durante o desenvolvimento deste experimento. Podemos considerar as seguintes melhorias:

.

Mudança de abordagem de *Data Centric* para *Model Centric*, treinando o modelo *EasyOCR* e *Pytesseract* com banco de imagens de baixa definição. A otimização desses modelos seria estabelecer uma nova meta para melhorar o nível de confiança para valores superiores a 95%. Mesmo que o modelo *PaddleOCR* apresente o melhor resultado, este ainda precisa melhorar a sua performance no *Runtime,* neste caso se faz necessário otimizar os seus hiperparâmetros.

A aplicação de uma película anti-reflexo no display do equipamento, reduziria parcialmente alguns ruídos na captura das imagens em ambientes iluminados pelo sol.

O ajuste fino e dinâmico diretamente sobre a janela de cada captura de imagem para cada ROI, facilita a otimização do *script* `Cameras` ajustado pela função `cv2.treshold`.

A otimização do *dataframe* output dos modelos, com a inclusão de uma nova coluna, contendo tempo (hora, minuto e segundos) que pode ser usada na criação do arquivo imagem, ao invés utilizar a fórumla de passos de tempo constante na coluna Time [sec].

REFERÊNCIAS

[1] SMITH, Ray. An Overview of the Tesseract OCR Engine, Google Inc, 2007. Disponível em: < https://github.com/tesseract-ocr/docs/blob/main/tesseracticdar2007.pdf >. Acesso em: 20 de Junho de 2023.

[2] SHI, Baoguang. BAI, Xiang. YAO, Cong. An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition. Huazhong University of Science and Technology, Wuhan, China, 21 de Julho de 2015. Disponivel em < https://arxiv.org/pdf/1507.05717.pdf>. Acesso em: 20 de Junho de 2023.

[3] DU, Yuning. LI, Chenxia. GUO, Ruoyu. CUI, Cheng. LIU, Weiwei. ZHOU, Jun. LU, Bin. YANG, Yehua. LIU, Qiwen. HU, Xiaoguang. YU, Dianhai. MA, Yanjun. PP-OCRv2: Bag of Tricks for Ultra Lightweight OCR System. Baidu Inc, 12 de Outubro de 2021. Disponivel em < https://arxiv.org/pdf/2109.03144.pdf>. Acesso em: 20 de Junho de 2023.

[4] EAST TESTER, ET5410 、ET5411 Single-Channel ProgrammableDC Electronic Load User Manual. Hangzhou Zhongchuang Electronics Co, 2023,. Disponível em: <https://www.easttester-cn.com/uploads/ET5410-ET5411-Programmable-Single-DC-Electronic-Load-User-Manual.pdf>. Acesso em: 05, Julho e 2023.

[5] TOOLKITRC, M6DManual V1.1, Julho de 2020. Disponível em: <https://l1nq.com/aZycw>, Acesso em: 25 de Junho de 2023.

[6] DJI, Bateria de vôo inteligente 2023. Disponível em: <https://www.dji.com/br/phantom-4-pro-v2/specs>. Acesso em: 25 de Junho de 2023.

[7] DCAI.CSAIL.MIT.EDU, Data-Centric AI vs. Model-Centric AI, seminario IAP 2023. Disponível em: <https://dcai.csail.mit.edu/lectures/data-centric-model-centric/>. Acesso em: 28 de Junho de 2023.

[8] NVIDIA, CUDAToolkit driver, 2023. Disponível em: <https://developer.nvidia.com/cuda-11-7-0-download-archive>. Acesso em: 29 de Junho de 2023.

[9] NVIDIA, cuDNN download Library, 2023. Disponível em: <https://developer.nvidia.com/rdp/cudnn-archive>. Acesso em: 29 de Junho de 2023.

[10] NVIDIA, Support Matrix, 2023. Disponível em: <https://docs.nvidia.com/deeplearning/cudnn/support-matrix/index.html>. Acesso em: 29 de Junho de 2023.

[11] IRIUN, *Webcam* for Windows, 2023. Disponível em: <https://iriun.com/>, Acesso em: 05 de Julho de 2023

[12] OPENCV.ORG, Color converstions, ,manual do opencv.org, 2023. Disponível em: <https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html>. Acesso em: 29 de Junho de 2023.

[13] OPENCV.ORG, Histograms - 2: Histogram Equalization, 2023, Disponível em: <https://docs.opencv.org/4.x/d5/daf/tutorial_py_histogram_equalization.html>. Acesso em: 02 de Julho de 2023.

[14] OPENCV.ORG, Image Denoising, 2023, Disponível em: <https://docs.opencv.org/3.4/d5/d69/tutorial_py_non_local_means.html>. Acesso em: 02 de Julho de 2023.

[15] OPENCV.ORG, Image Thresholding, 2023. Disponível em: <https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html>. Acesso em: 02 de Julho de 2023.

[16] OPENCV.ORG, Geometric Transformations of Images, 2023. Disponível em: <https://docs.opencv.org/3.4/da/d6e/tutorial_py_geometric_transformations.html>. Acesso em: 02 de Julho de 2023

[17] OPENCV.ORG, Adding borders to your images, 2023. Disponível em: <https://docs.opencv.org/3.4/dc/da3/tutorial_copyMakeBorder.html>. Acesso em: 02 de Julho de 2023

[18] OPENCV.ORG, Image file reading and writing, 2023. Disponível em: <https://docs.opencv.org/3.4/d4/da8/group__imgcodecs.html> Acesso em: 02 de Julho de 2023