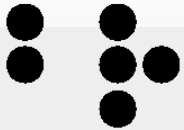
Tradução de texto em Braille a partir de imagens

Introdução ao Processamento de Imagens Digitais e Visão Computacional

Professora:
Luciana Ribeiro Veloso
Monitoria:
Leo de Lima Araújo

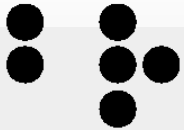
Alunos:
Alysson Machado de Oliveira Barbosa
Iury Chagas da Silva Caetano
Francinildo Barbosa Figueiredo

Introdução



- Conseguir traduzir textos em Braille para o alfabeto tradicional a partir de imagens digitais.
- Incentivar a disseminação do conhecimento produzido por parte de deficientes visuais.
- Divisão do Trabalho:
  a. Pré-processamento em imagens;
  b. Detecção de caracteres em braille;
  c. Classificação dos caracteres detectados;
  d. Tradução do texto em braille para o alfabeto tradicional;

Pré-Processamento dos Dados de Entrada

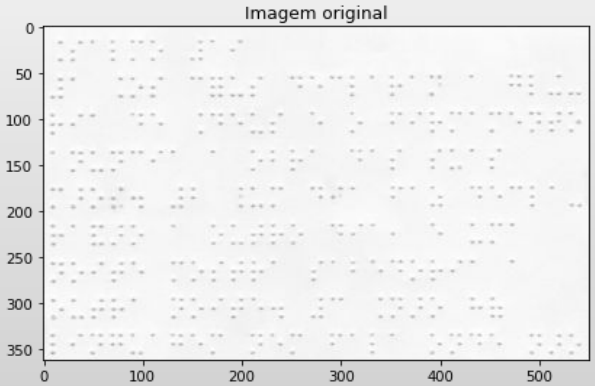


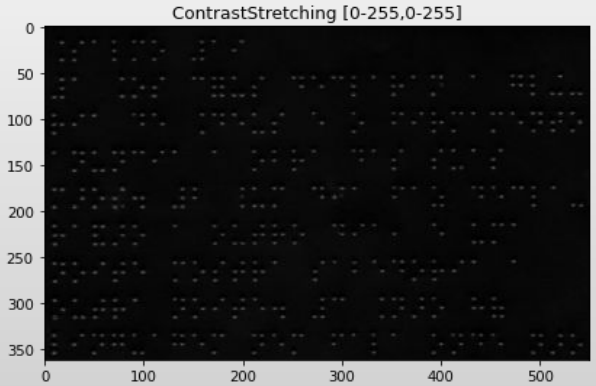
- Um dos grandes desafios de um projeto que visa classificar algo por imagens é o pré-processamento. Pensando nisso, temos essa como a parte inicial do projeto, onde poderemos identificar desafios a serem superados e também podemos identificar novos tipos de abordagens e claro, aprender com esses desafios.
- Ao longo dessa apresentação iremos expor alguns desafios para chegarmos nos resultados que temos até então. Portanto, poderemos **Classificar e Traduzir imagens do Braille para o Português convencional**.

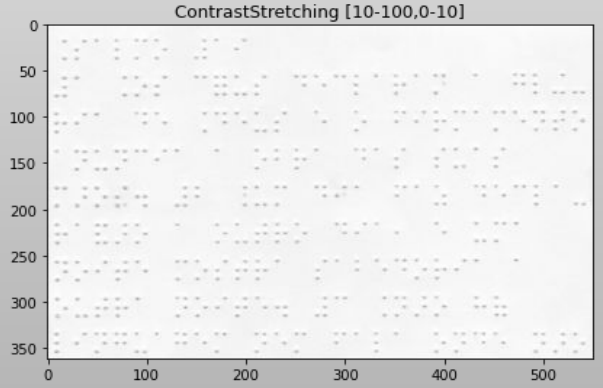
Resultados Iniciais

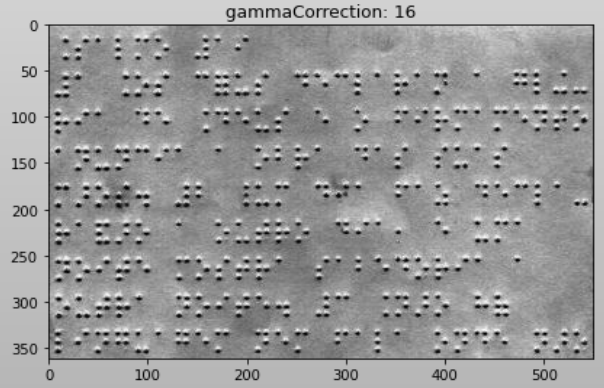
- Abordagens Iniciais:
- De primeiro entendimento do problema foi realizado uma sequência de tratamentos na imagem como: **Gamma Correction** e **Contrast Stretching**.
- Observamos alguns problemas com esse tipo de abordagem e optamos pela mudança para a **Binarização**.

Resultados Obtidos

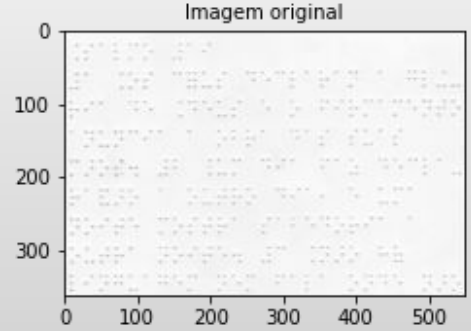


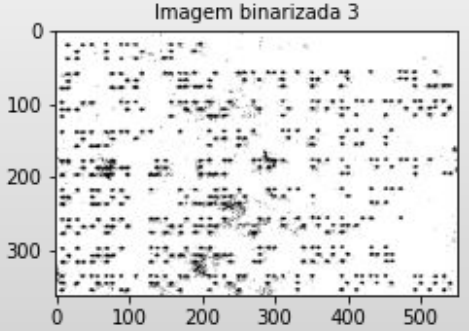


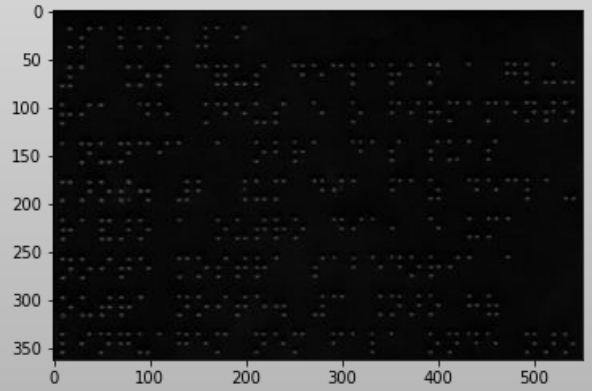




Aplicando a Binarização





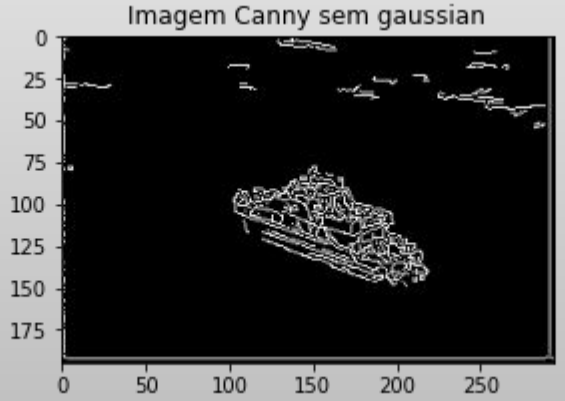


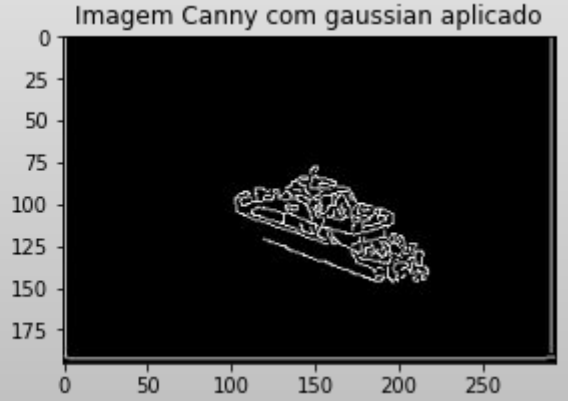
Aplicação do Filtro Gaussiano

- Foi observado a utilidade do **Filtro Gaussiano** quando aplicado a uma binarização ou outro método de segmentação, como o **Detector de Canny**.
- Com esse tipo de uso do filtro, podemos selecionar somente as partes mais relevantes da imagem, fazendo com que ocorra uma filtragem em alguns ruídos ou pixels provenientes de uma baixa resolução da imagem ou qualidade da câmera que a imagem foi capturada.

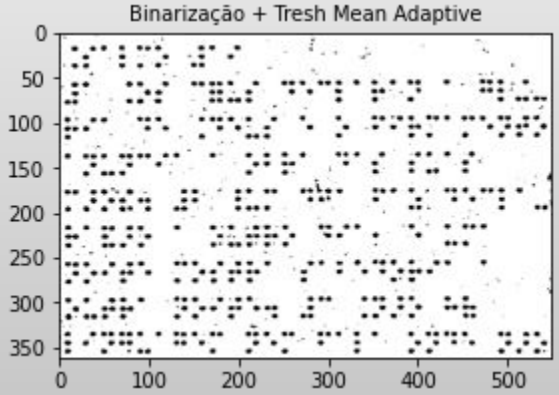
Resultados do Filtro Gaussiano

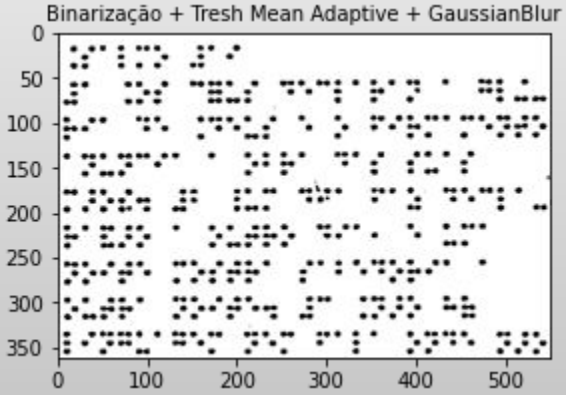
- Temos o **Filtro Gaussiano** aplicado a dois exemplos, um onde é possível observar mais claramente o seu uso e outro onde observa-se a remoção de ruídos da imagem.

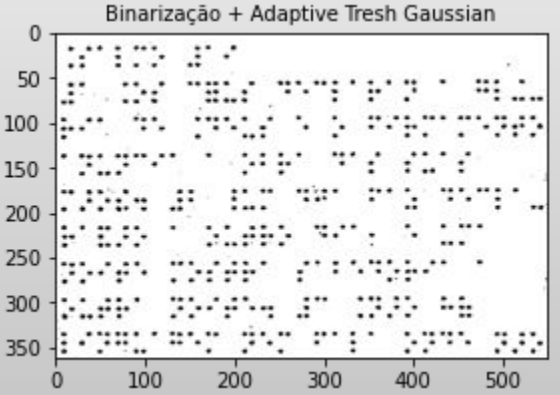




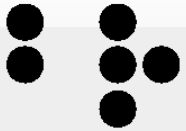
Segundo Exemplo do Filtro Gaussiano





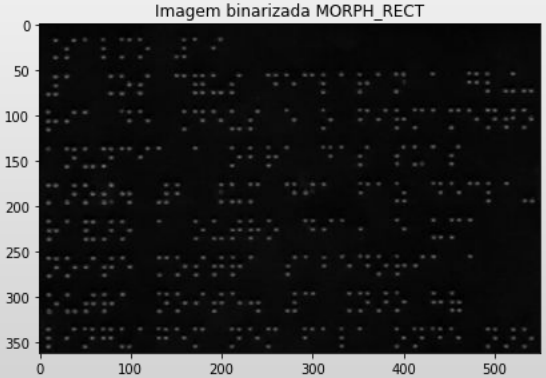


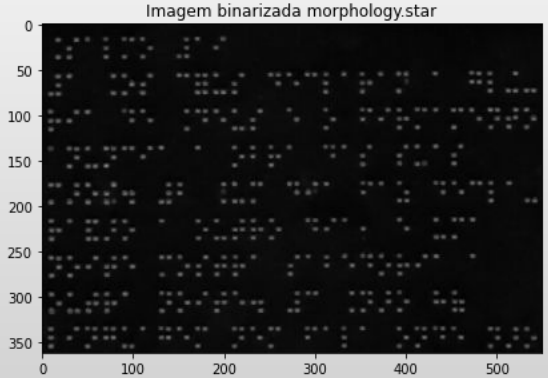
Filtros Usados para Binarização

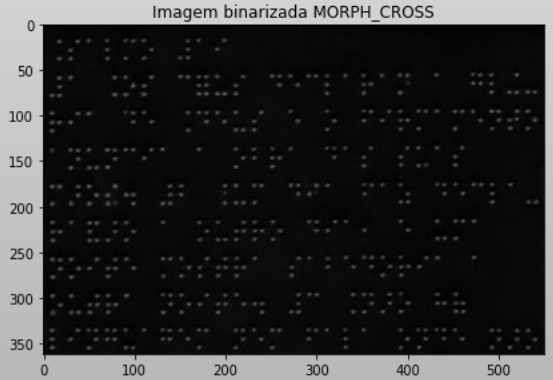


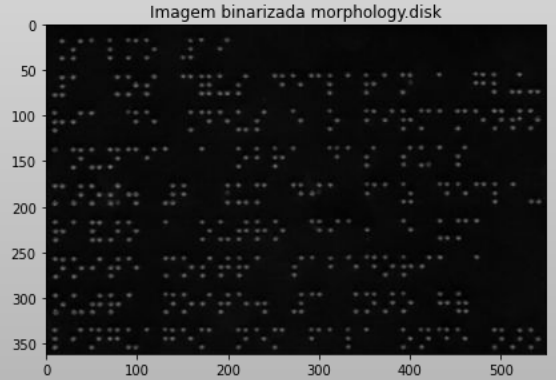
- **MORPH_RECT**
  - morphology.star
- **MORPH_CROSS**
  - morphology.disk
- Esses filtros são utilizados com Kernels definidos para que haja uma melhor segmentação por elementos estruturantes. Nesta aplicação, obtivemos resultados satisfatórios.

Resultado dos Filtros



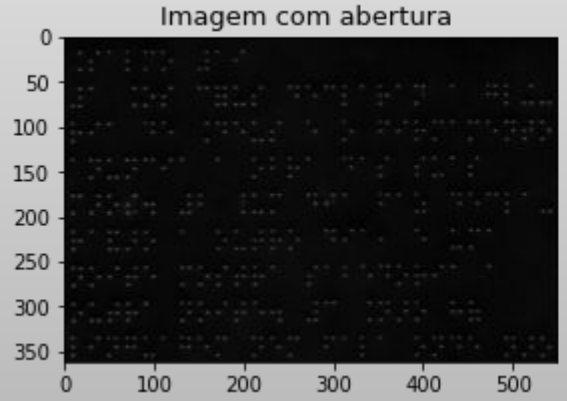


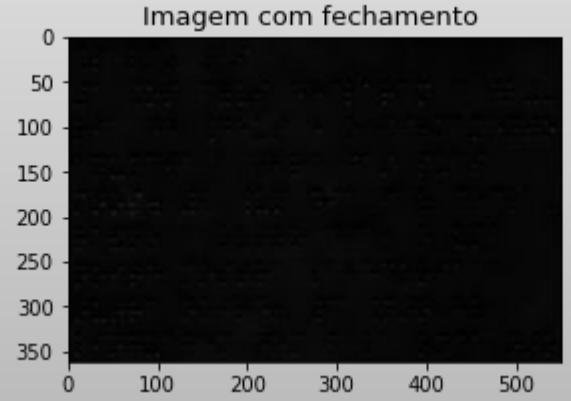


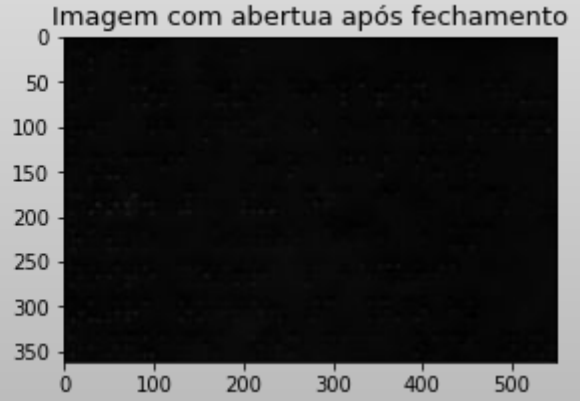


Abertura e Fechamento

- Outro processo que mostrou-se ser significativo foi o da **Abertura e Fechamento**, onde pudemos obter um segmentação próxima da que queríamos no início, mas ainda sim, nota-se que a imagem não fica da maneira mais ideal para ser segmentada posteriormente.



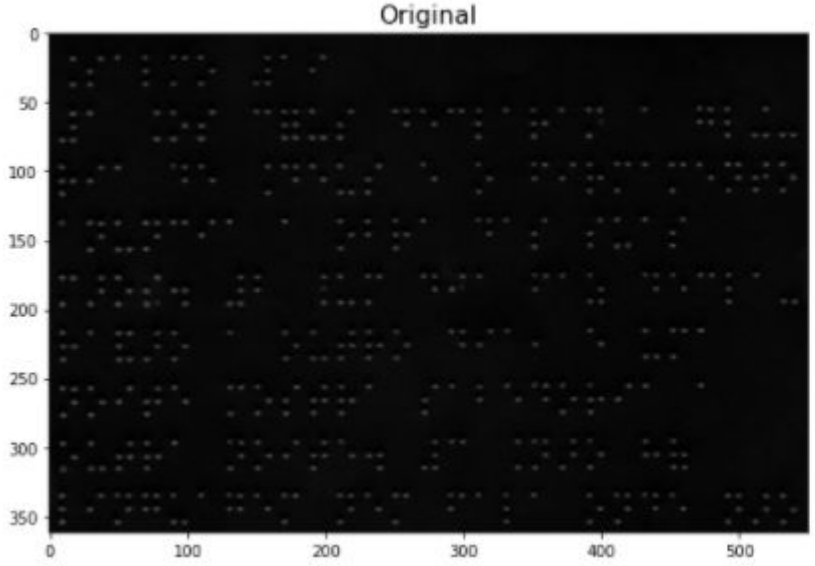


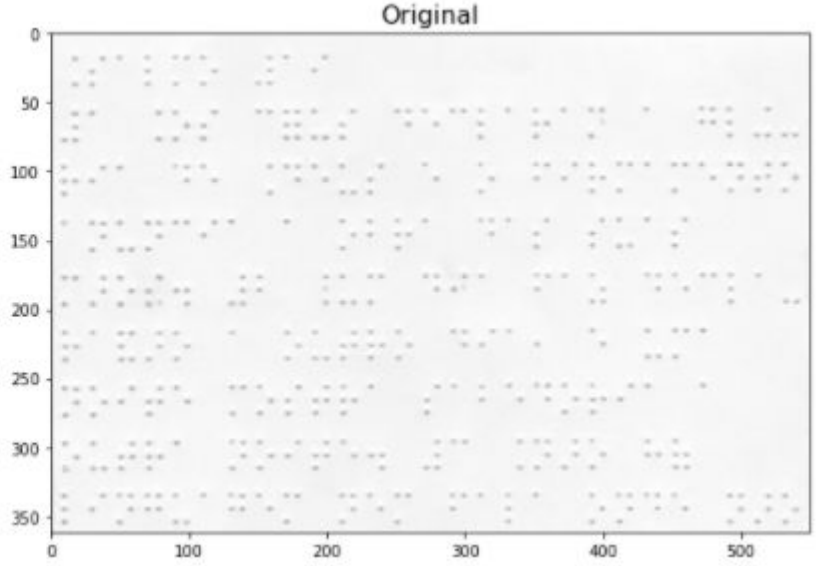


Detector Canny e Detector Sobel

- Como falado anteriormente, o **Detector de Canny e Sobel** mostra-se útil quando o assunto é contorno e segmentação de objetos, logo, o seu próprio conceito já nos deixa brechas para o uso em nosso projeto.
- Vejamos alguns exemplos do **Canny** e o **Sobel** utilizando técnicas de Dilatação para aprimorar seus resultados.

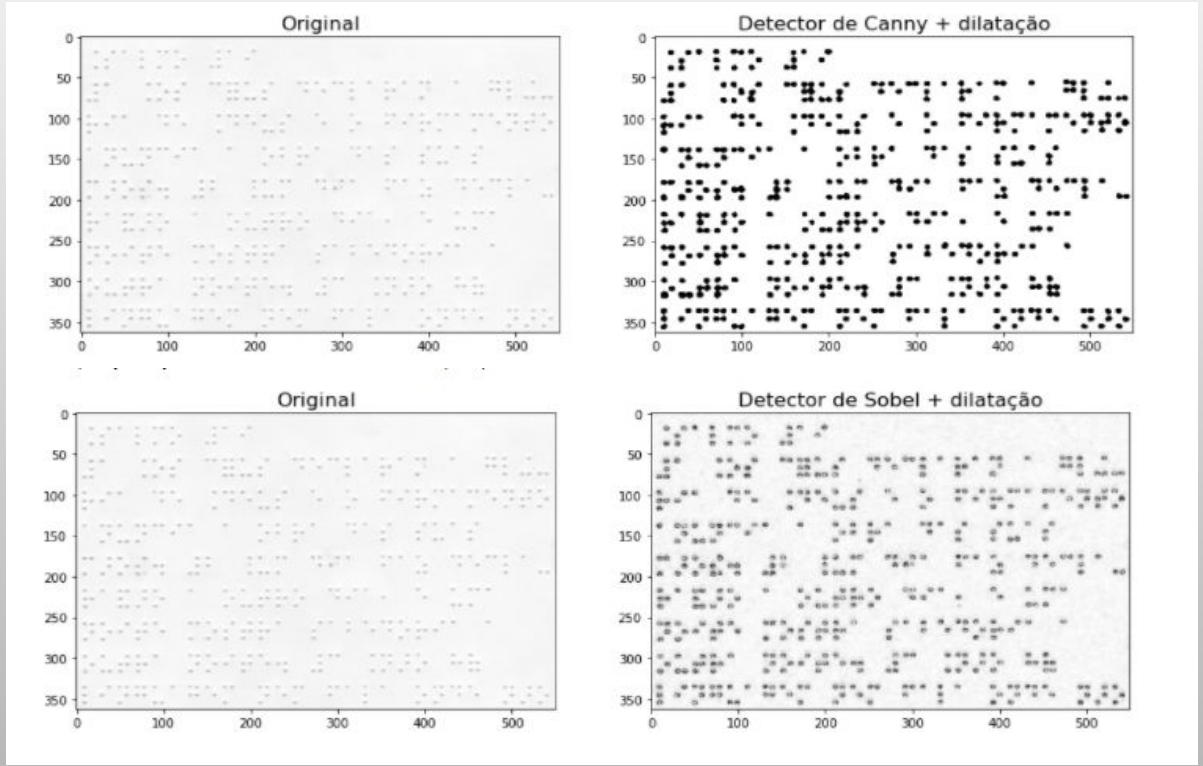
Imagens Originais - Canny e Sobel





Resultados - Canny e Sobel

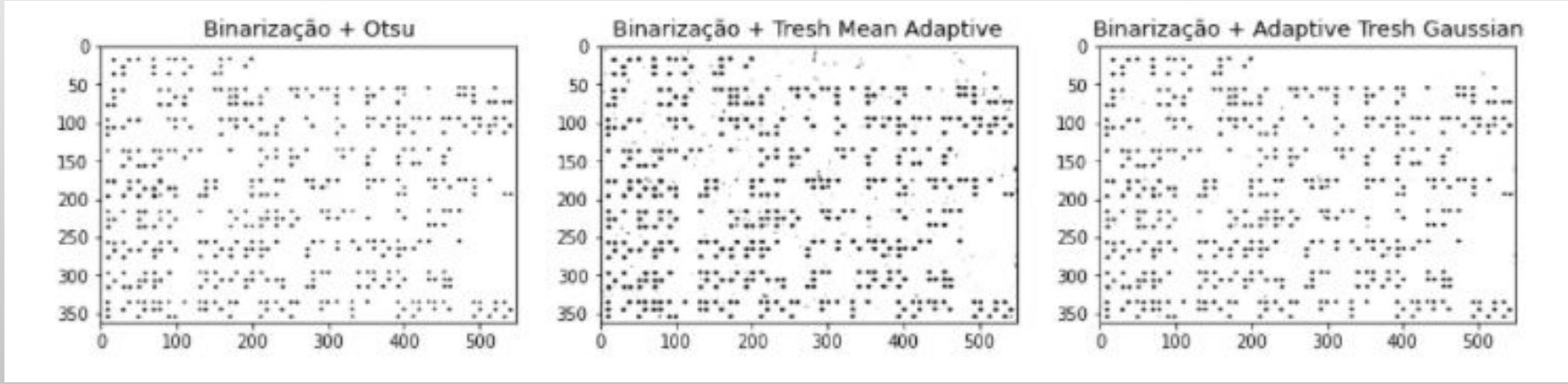
Resultado: Canny + Dilatação



Binarização Adaptativa

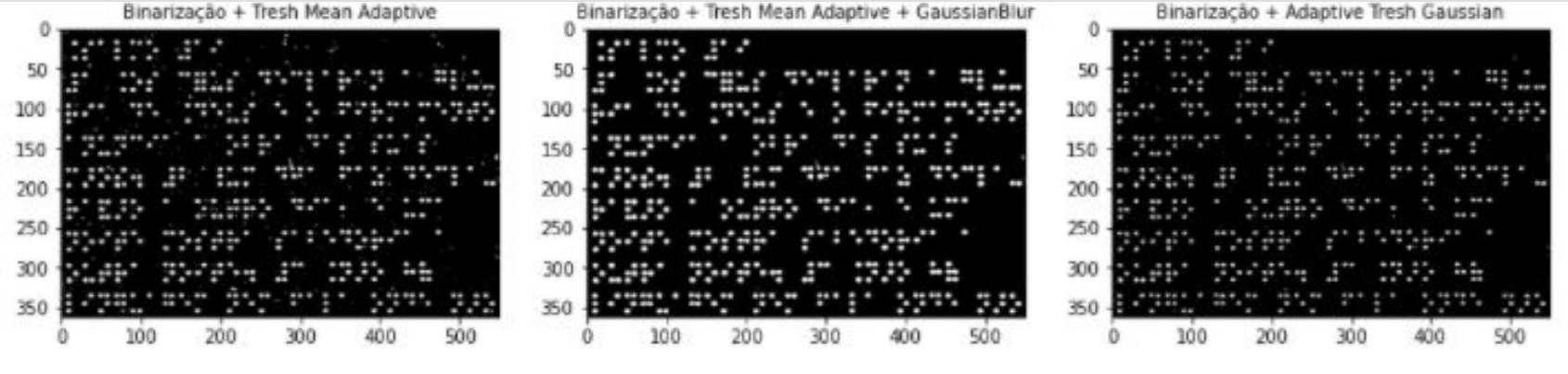
- Foi usado a **Binarização Adaptativa** juntamente com a **OTSU** com a finalidade de encontrar um resultado superior ao Canny, que se mostrava até então mais eficiente.
- Um dos principais desafios desse método é a alta captação de ruído da imagem, pensando nisso, sua função foi criada com um filtro gaussiano para reduzir a maior parte desse ruído. Entretanto, mesmo com esse tratamento, há imagens que a Binarização Adaptativa é ineficiente quando comparado ao Canny.

Resultados - Binarização Adaptativa e OTSU



Captação de ruído e tratamento

- Foi observado um ruído na imagem de treinamento e com o filtro colocado na função de processamento pela binarização adaptativa ele foi contornado.

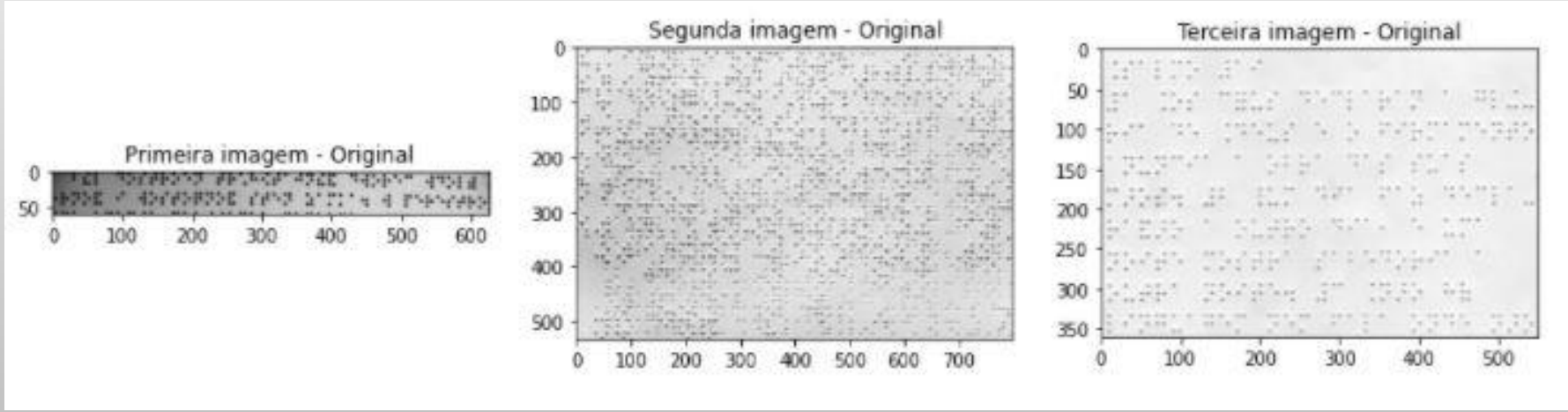


Teste do pré-processamento

- Uma das etapas do pré-processamento é justamente o teste em diversas imagens, seja em boa qualidade ou ruim, temos que saber o que nosso algoritmo consegue lidar e o que não consegue.
- Portanto, com todas as abordagens concluídas, chega ao fim do Pré-Processamento de imagens de entrada.

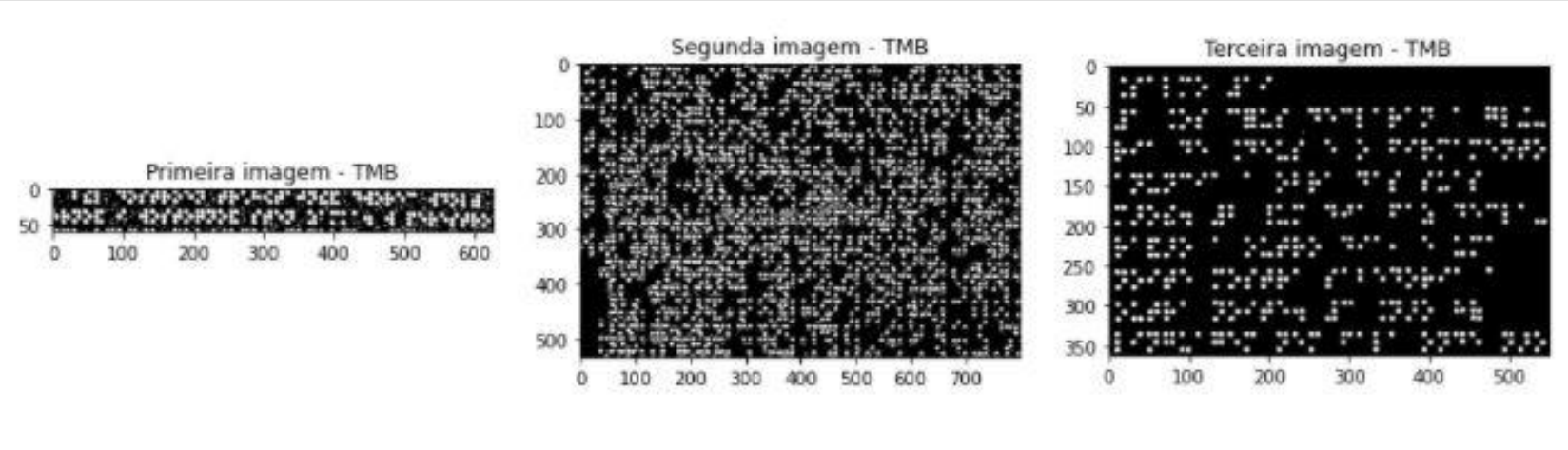
Resultado dos Algoritmos

- Imagens Originais:



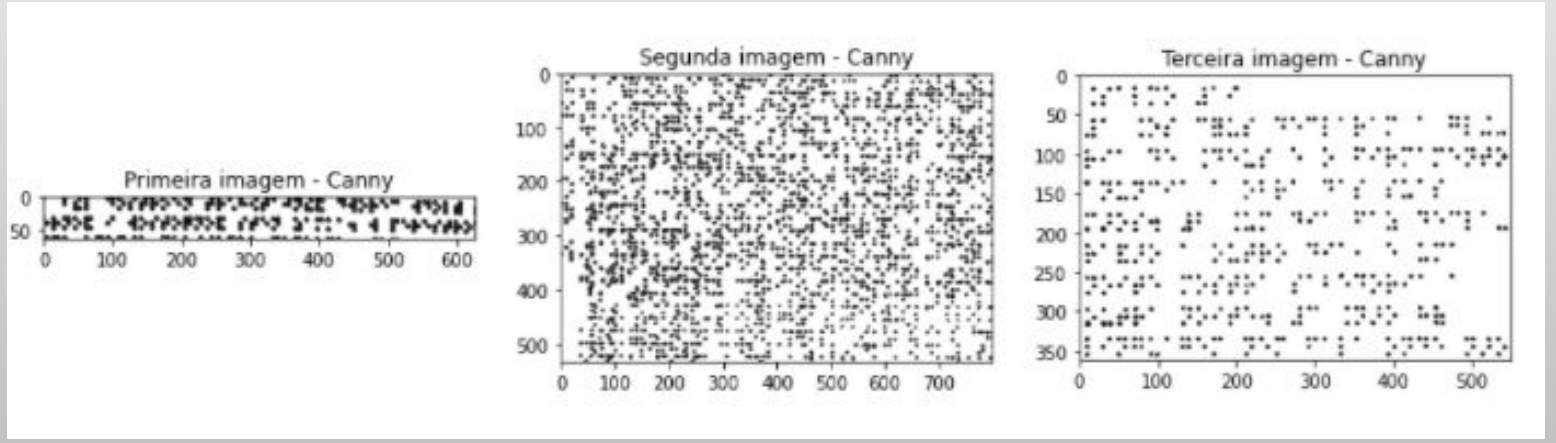
Resultado dos Algoritmos

- Imagens segmentadas pelo **Threshold Adaptativo**:



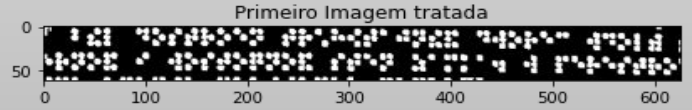
Resultado dos Algoritmos

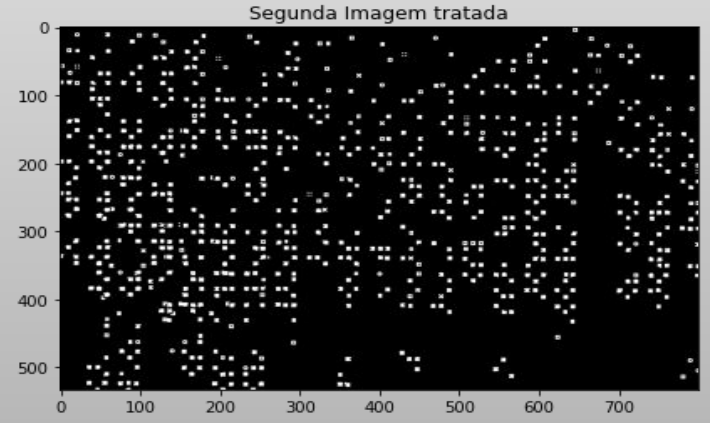
- Imagens segmentadas pelo Detector de Canny + Dilatação:



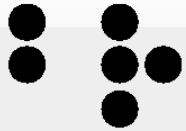
Comentários dos Resultados

- Notou-se que com a segmentação através da binarização deixa alguns ruídos que já estavam presentes na imagem ainda mais evidentes, pensando em um meio de solucionar isso, foi tratada a imagem com um filtro de contraste, para melhor aperfeiçoamento do filtro gaussiano. Assim, a segmentação foi totalmente aprimorada e esses foram os resultados:



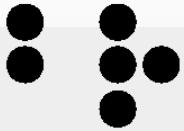


Conclusão do Pré-Processamento



- Com a finalização do Pré-Processamento, notou-se que para algumas imagens torna-se inviável segmentar os caracteres com as abordagens atuais. Portanto, é mais um desafio em que devemos nos atentar para próximas versões. Entretanto, para imagens com uma resolução maior e uma qualidade melhor, temos um ótimo resultado, principalmente com imagens digitalizadas em uma impressora, por exemplo.

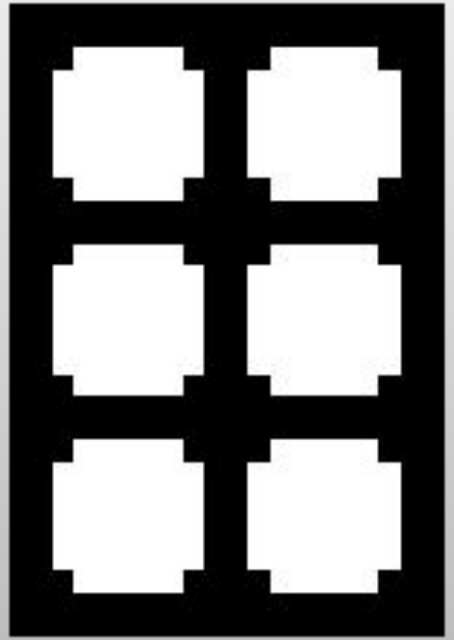
Detector de Caracteres Braille



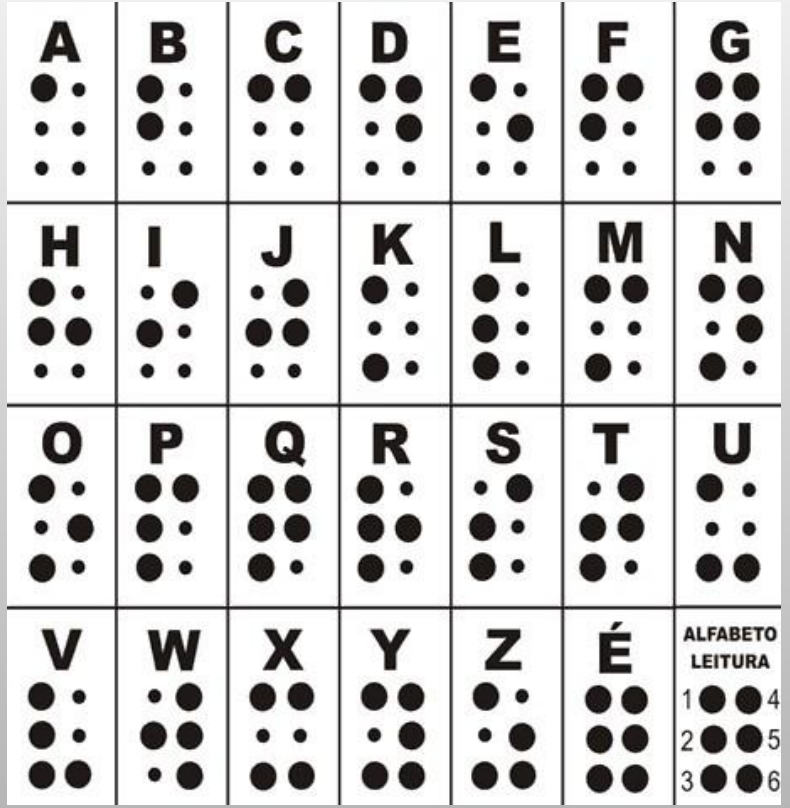
Passo a passo:

- Calcular o histograma de projeção horizontal da imagem;
- Determinar os delimitadores horizontais de cada linha de pontos;
- Determinar os delimitadores das linhas de caracteres;
- Obter sub imagens de cada linha de caracteres;
- Calcular o histograma de projeção vertical de cada sub imagem;
- Determinar os delimitadores verticais de cada coluna de pontos;
- Determinar os delimitadores de cada caractere;
- Obter sub imagens de cada caractere e organizar por palavras;

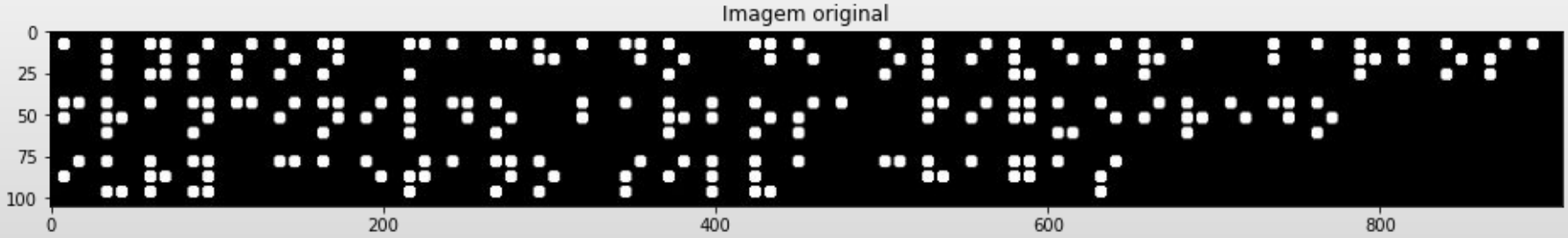
Caractere Braille

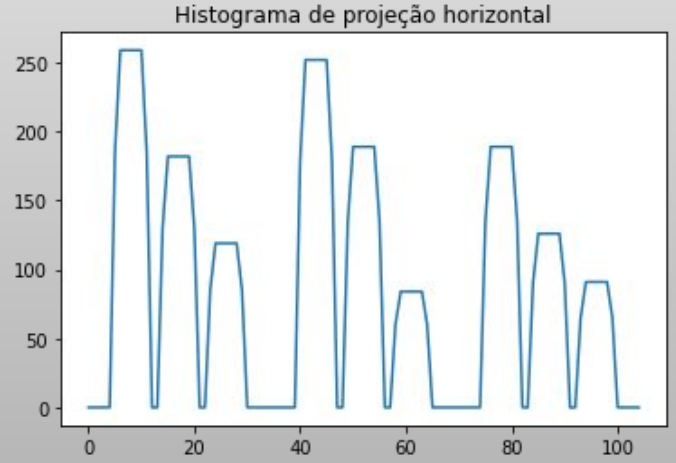


Alfabeto Braille

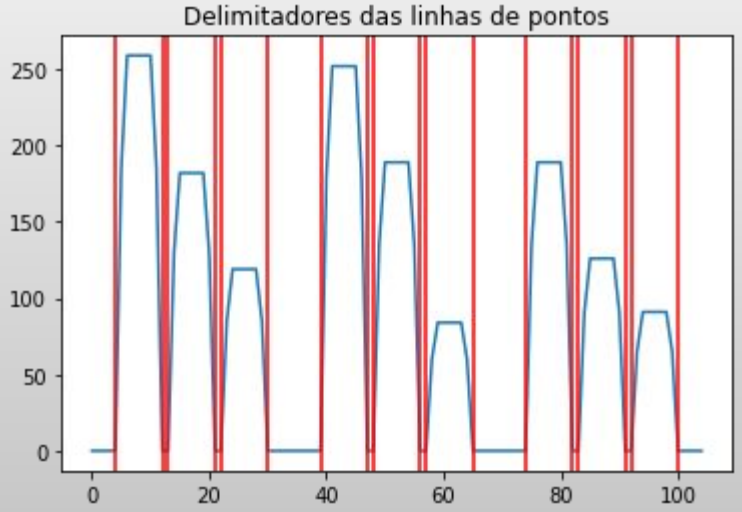


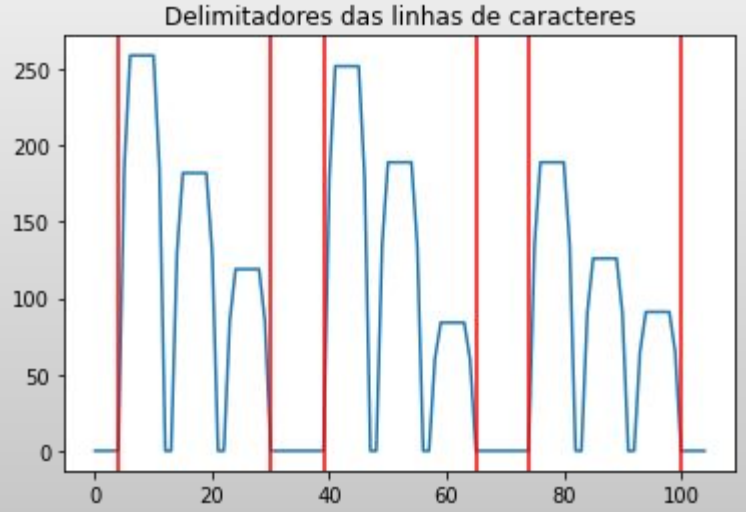
Cálculo do Histograma de Projeção Horizontal



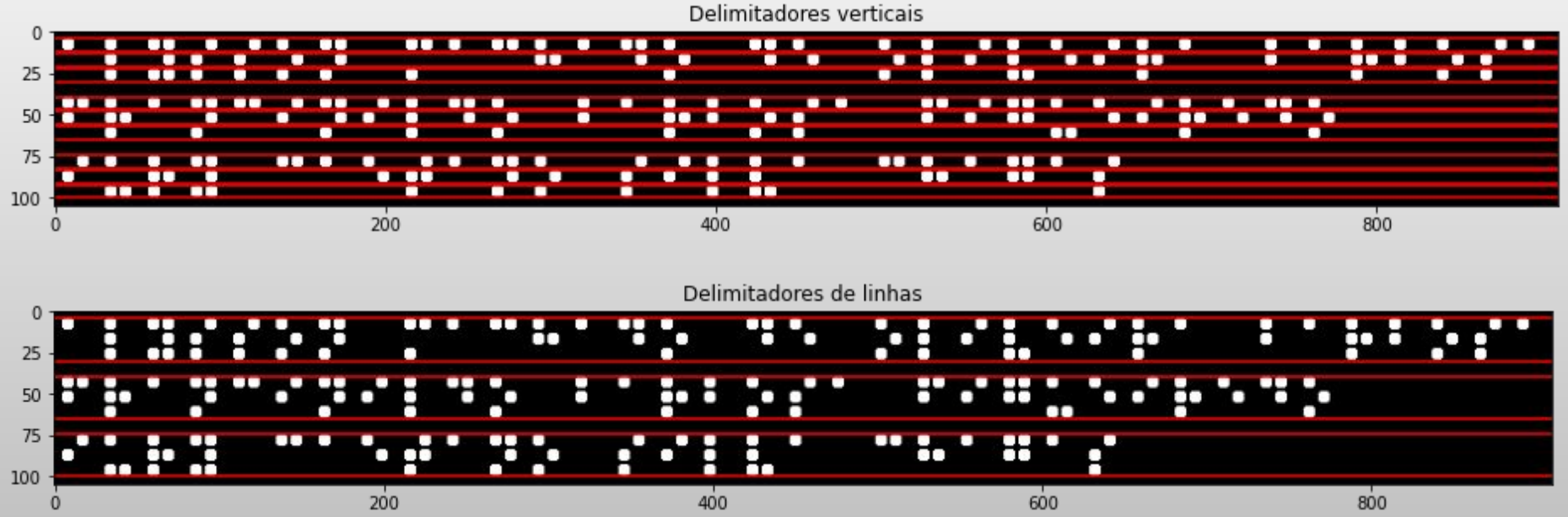


Determinando os Delimitadores Horizontais

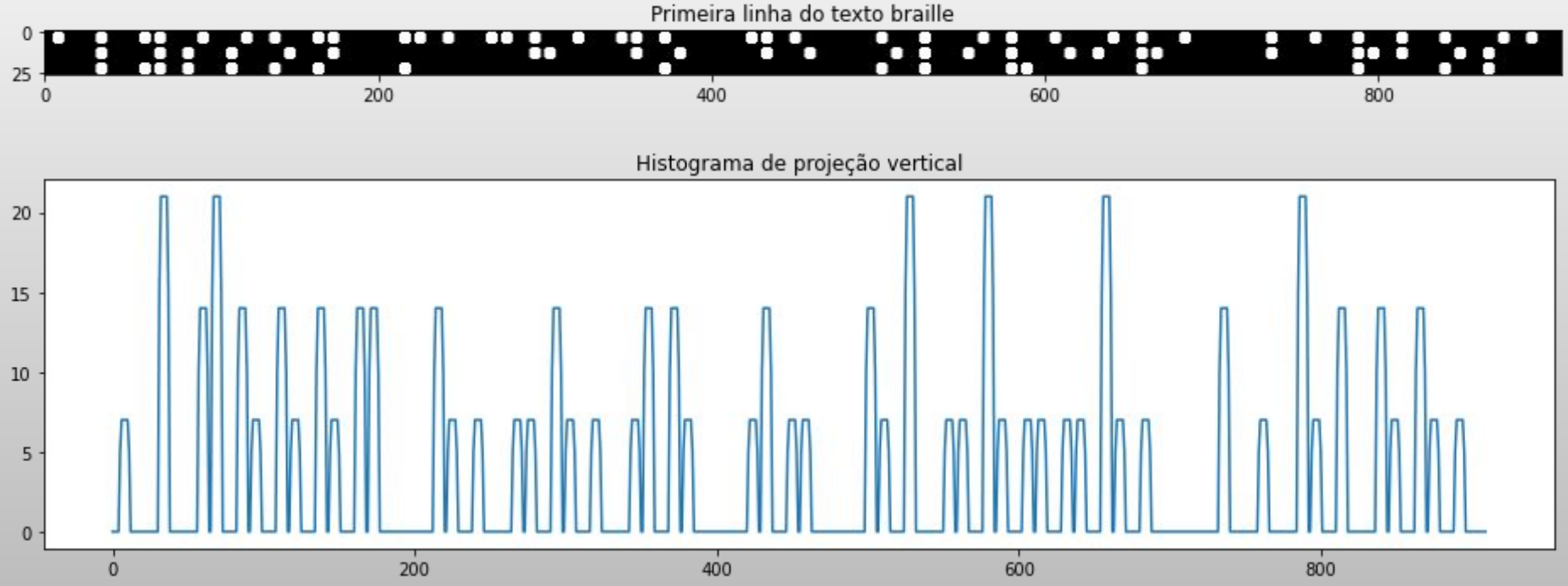




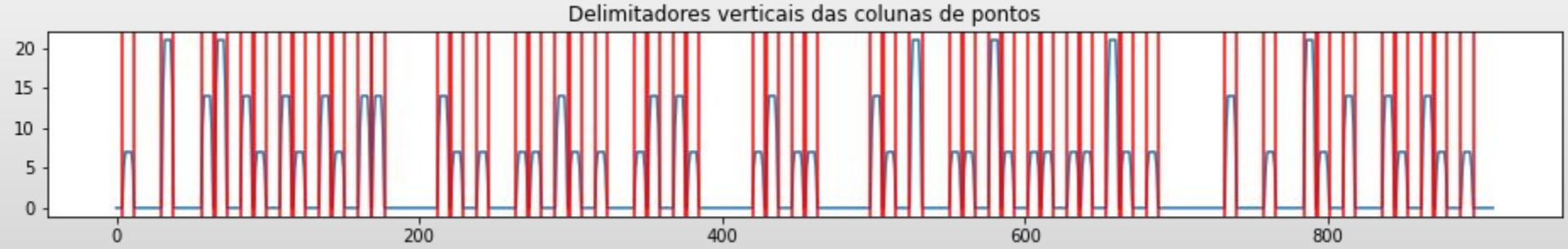
Determinando os Delimitadores Horizontais

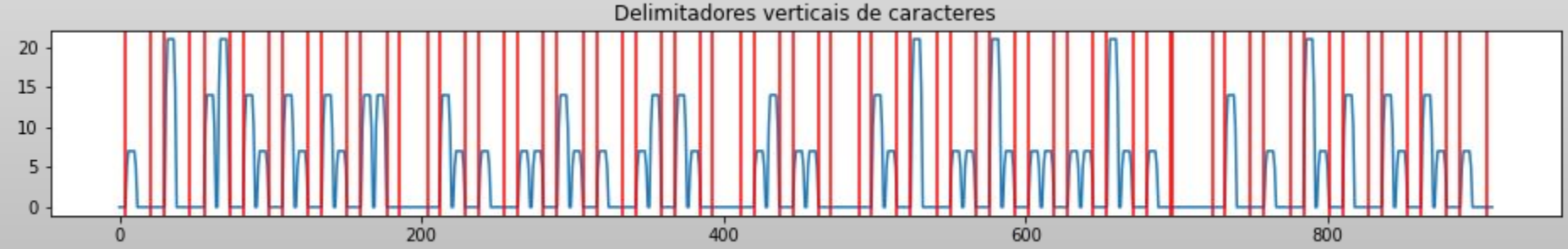


Histograma de Projeção Vertical na Linha

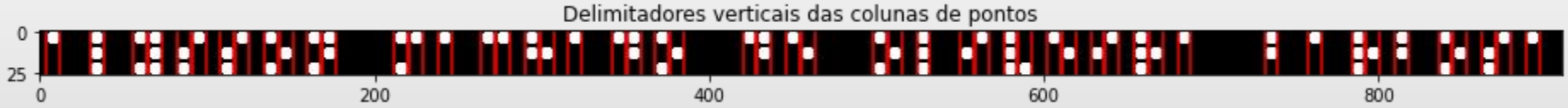


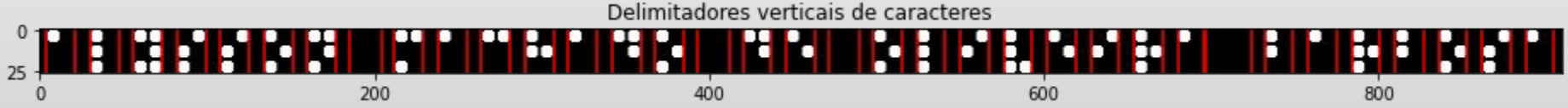
Delimitadores Verticais e de Caracteres



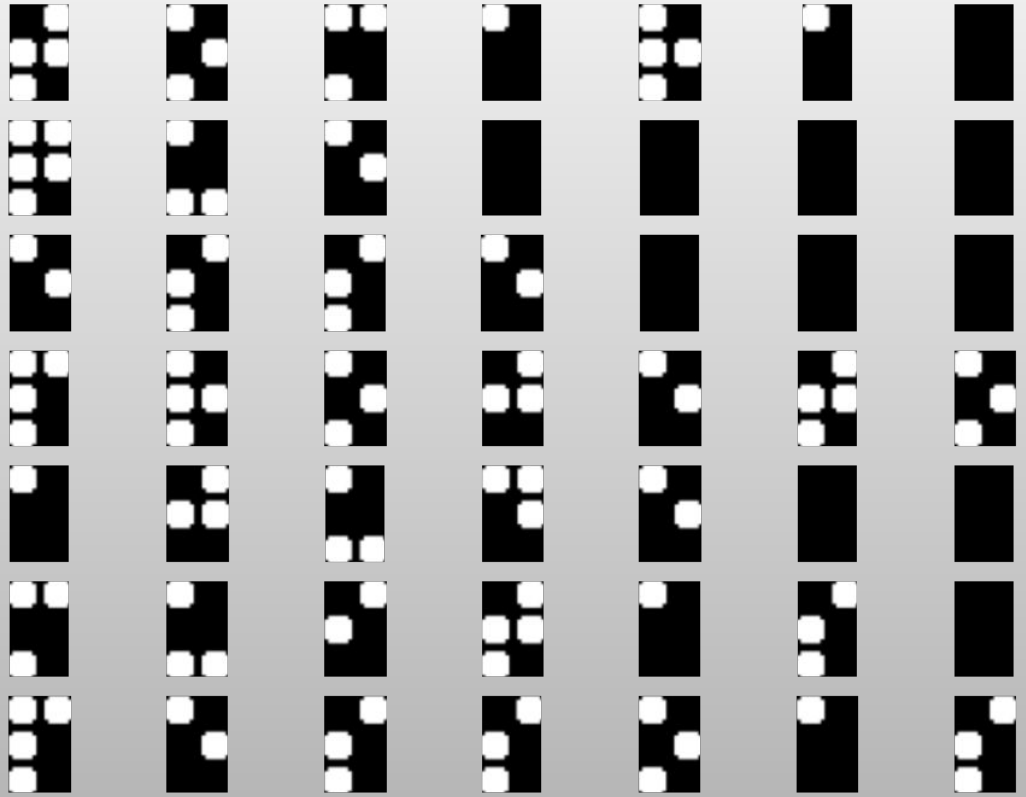


Delimitadores Verticais e de Caracteres



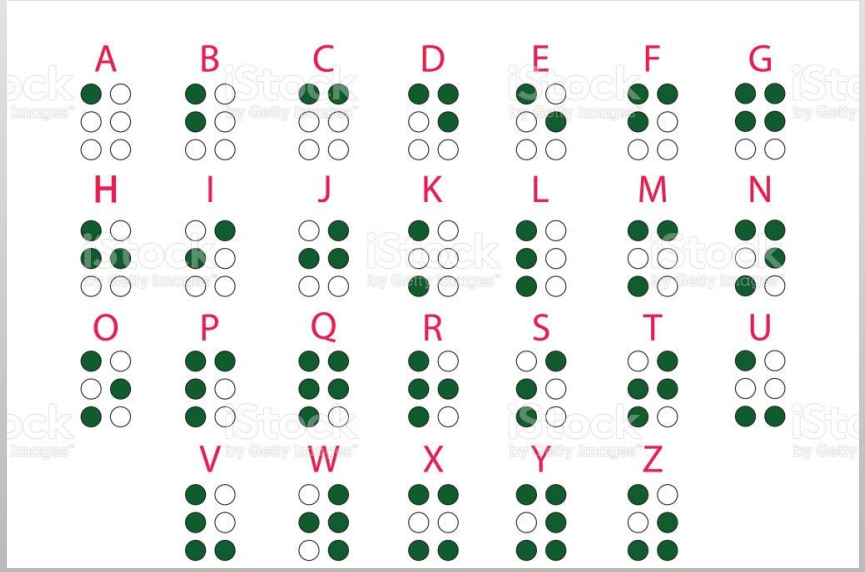


Resultado final

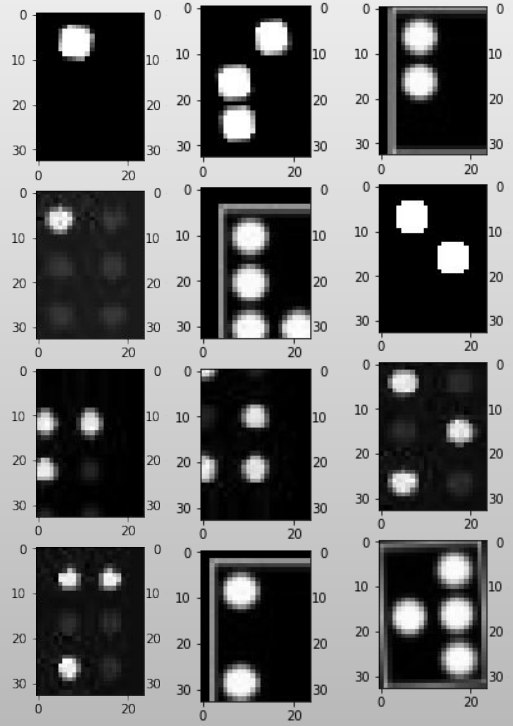


Classificador de Caracteres em Braille

- Uso de Redes Neurais Convolucionais para construção do classificador de caracteres escritos em braille.

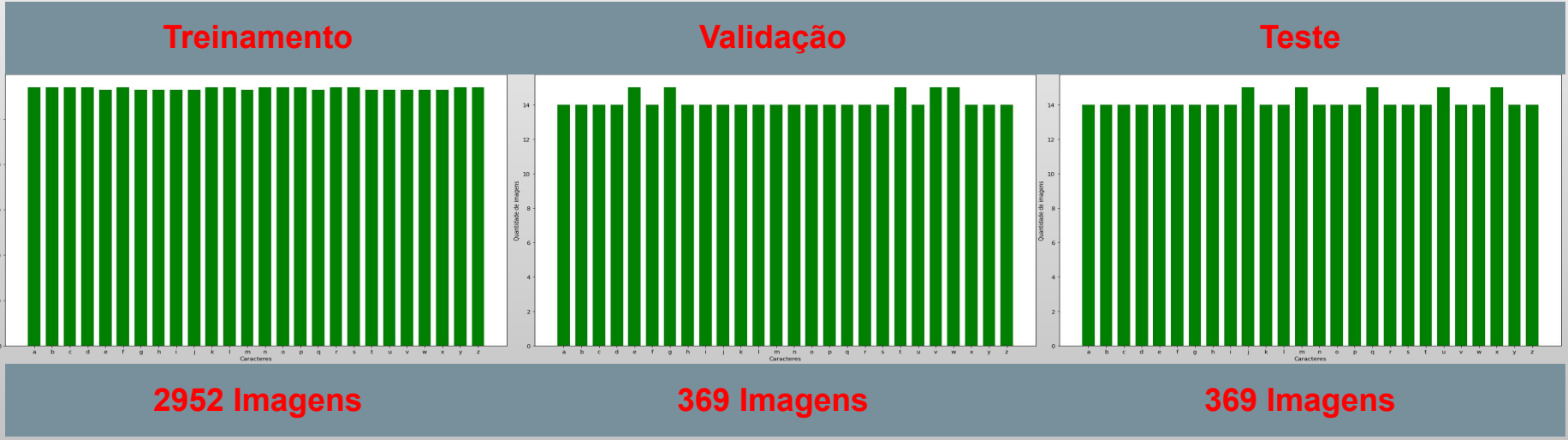


Organização da Base de Dados

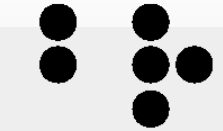


- **Fontes das Bases de Dados:**
  - Imagens geradas artificialmente;
  - Braille Character Dataset - Kaggle;
  - Braille Images for English Characters - Kaggle;
- **Aumento de Dados:**
  - Transladar a Imagem na vertical e horizontal
  - Limites de Translação: [-5 px, 5 px];
  - Rotações discretas na imagem no sentido horário e anti-horário;
  - Limites de Rotação [-10°, 10°];

Imagens de Treino, Validação e Teste



Arquitetura e Detalhes da Rede

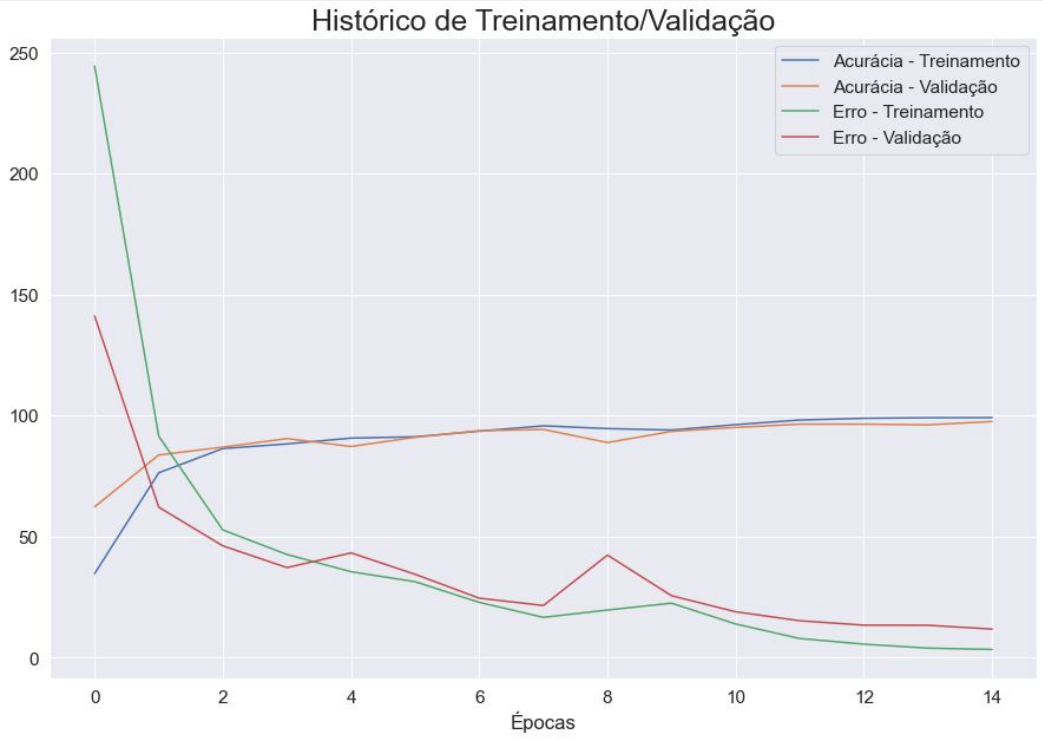


Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 33, 24, 1)]	0
Conv2D-1 (Conv2D)	(None, 32, 23, 32)	160
Pooling-1 (MaxPooling2D)	(None, 16, 11, 32)	0
Conv2D-2 (Conv2D)	(None, 15, 10, 64)	8256
Pooling-2 (MaxPooling2D)	(None, 7, 5, 64)	0
Flatten-1 (Flatten)	(None, 2240)	0
Dense-1 (Dense)	(None, 128)	286848
Dense-2 (Dense)	(None, 64)	8256
dense_1 (Dense)	(None, 26)	1690

Total params: 305,210
Trainable params: 305,210
Non-trainable params: 0

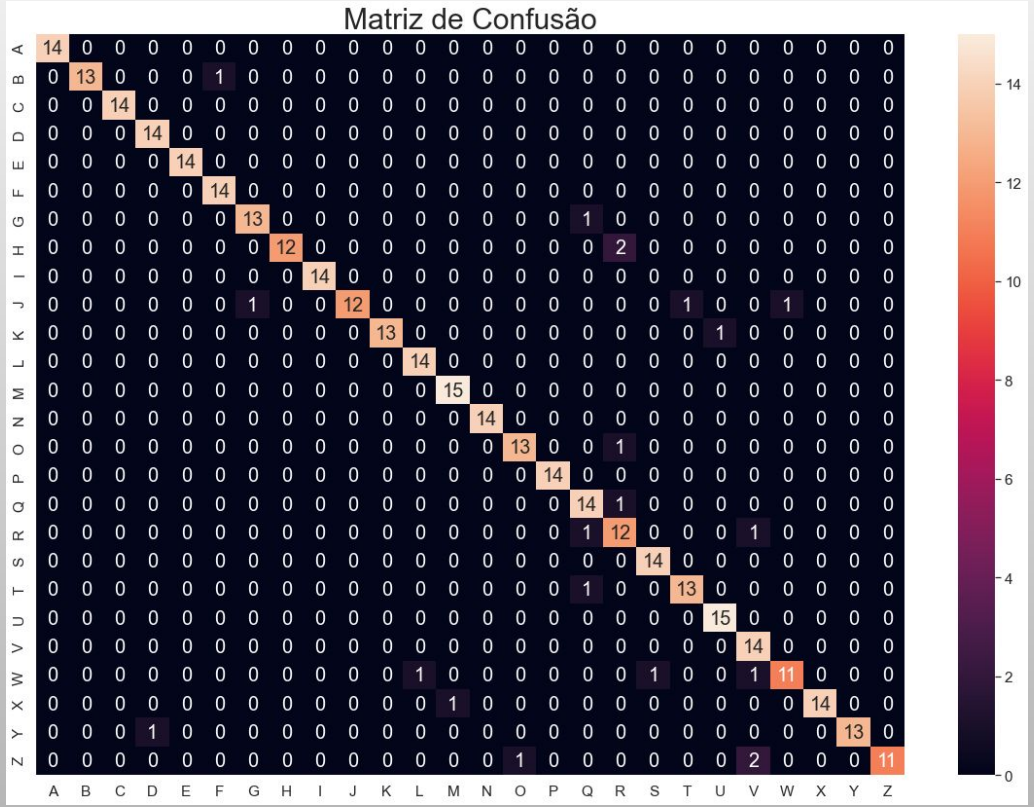
- **Otimizador:**
  - Adam;
- **Função de Custo:**
  - Categorical Crossentropy;
- **Callbacks:**
  - CSV History;
  - Model CheckPoint;
- **Épocas de Treinamento:**
  - 15 épocas;

Histórico e Resultados do Treinamento



Conjunto de Dados	Acurácia (%)
Treinamento	95,79
Validação	93,77
Teste	94,31

Matriz de Confusão



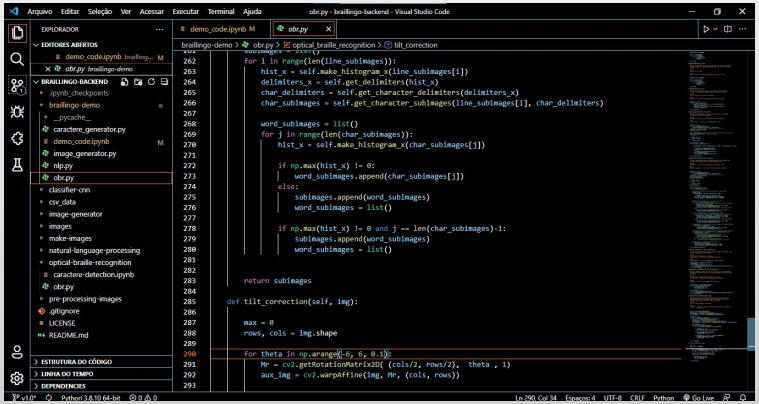
Algoritmo Completo

Pré-processamento da Imagem

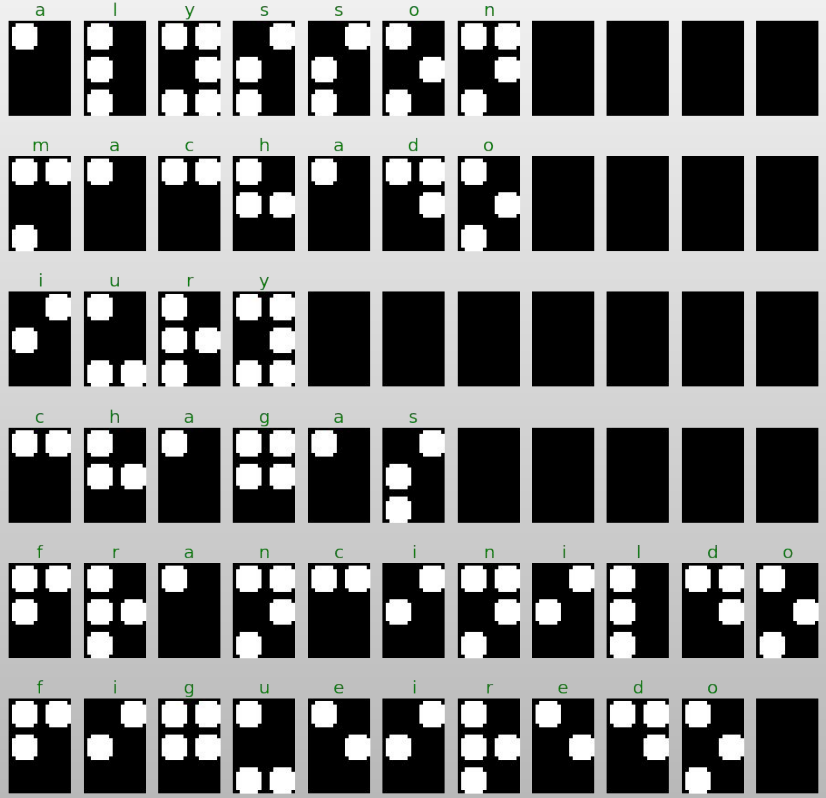
Detecção dos Caracteres

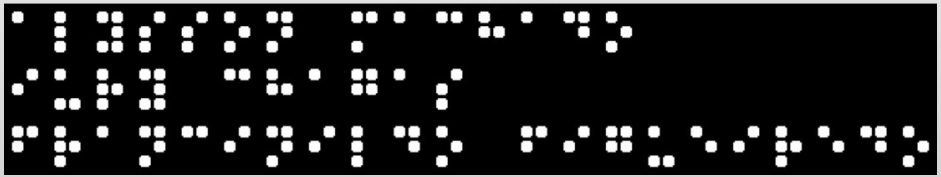
Classificação dos Caracteres

Tradução do Texto



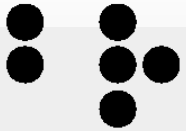
Demonstração do Projeto





Tradução do Classificador:
alysson machado iury chagas francinildo figueiredo

Obrigado Pela Atenção!



Alysson Machado de Oliveira Barbosa

alysson.barbosa@ee.ufcg.edu.br

Francinildo Barbosa Figueiredo

francinildo.figueiredo@ee.ufcg.edu.br

Iury Chagas da Silva Nascimento

iury.caetano@ee.ufcg.edu.br