Relatório Final de Processamento Digital de Imagens

Universidade Federal do ABC (UFABC)

Professor Francisco de Assis Zampirolli

Marcelo de Souza Pena - RA: 11039314

marcelo.pena@aluno.ufabc.edu.br

1. Introdução

Minha intenção era fazer um algoritmo para reconhecer a minha caligrafia e deixá-lo disponível para que qualquer um pudesse utilizá-lo para a própria caligrafia. Para desenvolver o projeto me inspirei principalmente nos arquivos extras das aulas 3 e 4, além de utilizar diversos outros processamentos vistos nas aulas, como converter uma imagem colorida para tons de cinza, *blur*, *threshold*, redimensionamento, encontrar contornos, etc.

2. Dados

Para fazer o treinamento utilizei uma folha quadriculada em que escrevi manualmente o alfabeto vinte e duas vezes, metade com letras maiúsculas e metade com letras minúsculas, conforme é possível ver na Figura 1.

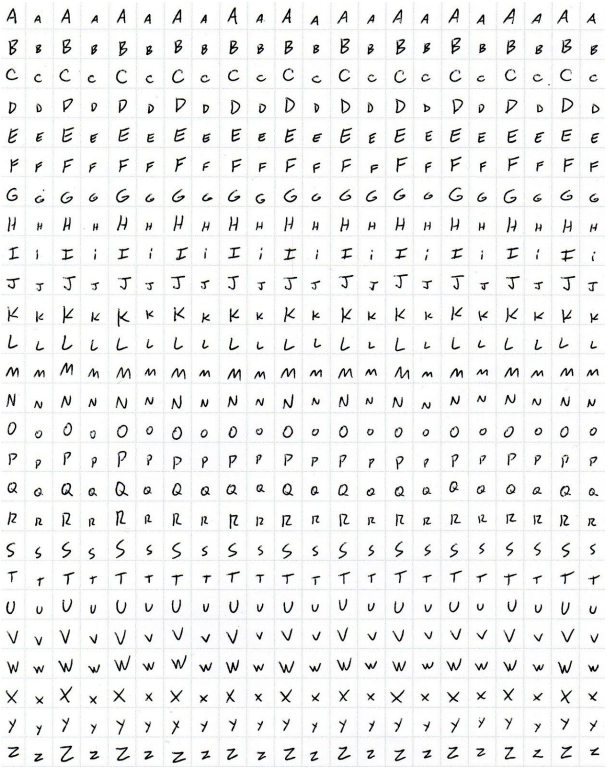

Figura 1 - Caracteres

Além dela também foi necessária uma imagem real para teste, como mostra a Figura 2. Escrevi frases e símbolos respeitando a limitação de utilizar somente as 26 letras do alfabeto, sem pontuação, acentos, etc., pois esses caracteres não foram treinados.

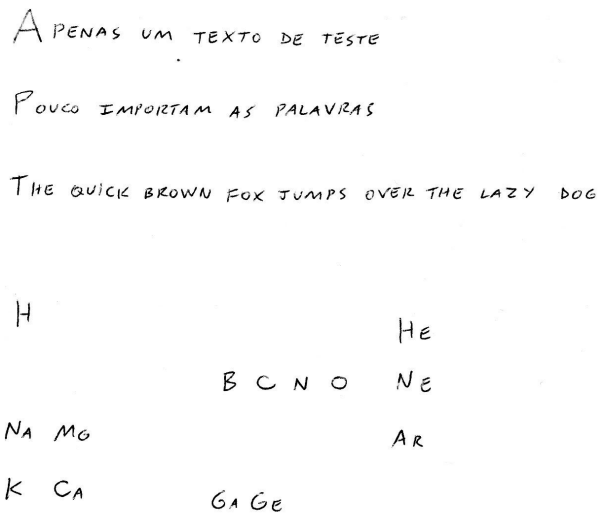

Figura 2 - Imagem de teste

3. Processamento

Durante o processamento foram feitas diversas transformações na imagem para adequá-la ao treinamento. A imagem original foi transformada de BGR para tons de cinza, então recebeu um filtro de *blur* a fim de eliminar algumas imperfeições e ajudar a eliminar as linhas de grade, então passou pelo *threshhold* método Otsu para deixá-la em preto e branco eliminando de vez imperfeições e deixando os caracteres mais destacados, conforme é possível ver na Figura 3.

Figura 3 - Threshold Otsu

Para fazer a classificação manual na etapa de treinamento tive que diminuir a imagem para ⅓ do tamanho original, pois com o tamanho total não conseguia visualizar ela inteira no meu monitor.

Para o treinamento uma função da biblioteca *cv2* encontra os contornos na imagem e todos os caracteres que tiverem um tamanho mínimo escolhido experimentalmente por mim vão ser classificados manualmente. A classificação é feita com o algoritmo desenhando retângulos azuis ao redor dos contornos

definidos, então a área correspondente da imagem em preto e branco é salva e redimensionada para 10x10, então são salvas com dimensão 1x100, de forma que todos os *samples* salvos tenham o mesmo *shape*. O algoritmo então abre a imagem com o retângulo azul marcando o caractere que está sendo classificado e o usuário deve digitar a letra correspondente. Caracteres já classificados são marcados por um retângulo vermelho, como mostra a Figura 5. As respostas são salvas em uma lista e é possível utilizar o botão ESC para sair da classificação.

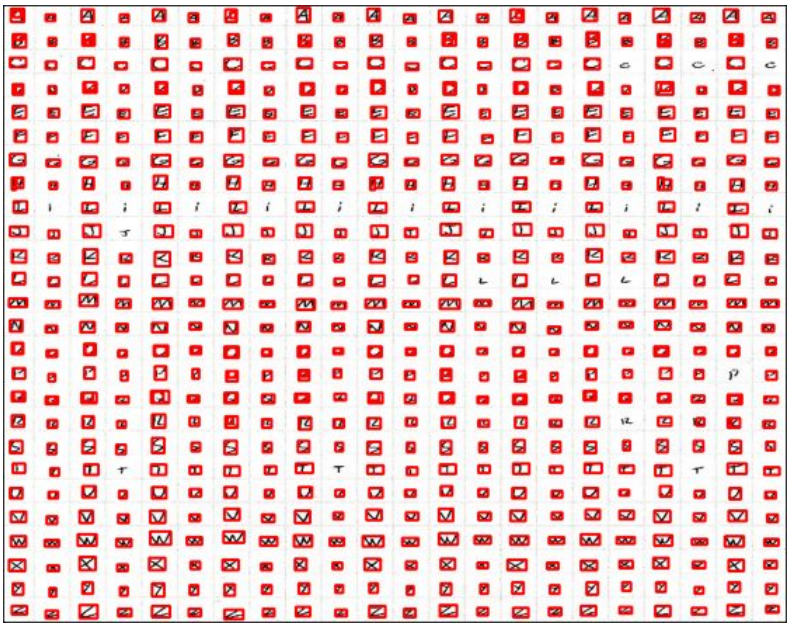

Figura 5 - Caracteres já classificados

Os *samples* salvos como 1x100 e as respostas digitadas pelo usuário são utilizadas para treinar o algoritmo KNN (K-vizinhos mais próximos), que deverá classificar novas amostras de acordo com a proximidade delas com as do treinamento.

A imagem de teste então passa pelos mesmos tratamentos: tons de cinza, *blur*, *threshold*, redimensionamento para metade do tamanho original e classificação manual (Figura 6).

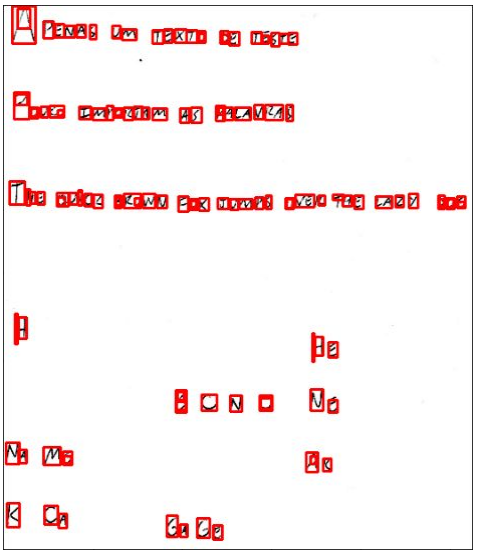

Figura 6 - Classificação manual da imagem de teste

No teste o algoritmo KNN classifica cada um dos caracteres encontrados de acordo com o vizinho mais próximo do conjunto de treino. Para visualizar o resultado, os caracteres atribuídos pelo KNN são escritos em uma imagem de fundo preto, a Figura 7.

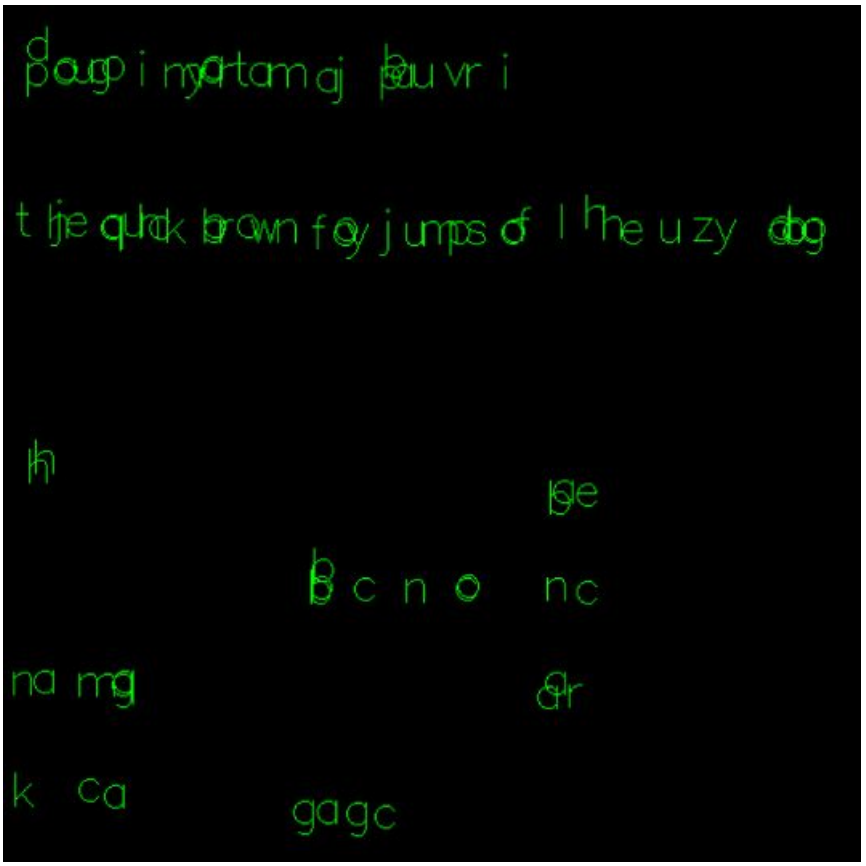

Figura 7 - Imagem com os caracteres atribuídos pelo KNN

4. Conclusão

A identificação dos caracteres na imagem se mostrou uma tarefa nada trivial. Ao mesmo tempo que alguns caracteres eram identificados mais de uma vez, como a letra B que frequentemente era identificada três vezes, sendo inteiro, parte de cima e parte de baixo, alguns caracteres não era identificados de jeito nenhum, mesmo eu diminuindo o tamanho necessário, o que também fazia com que o número de caracteres identificados mais de uma vez crescesse. Para resolver tive que escolher um meio termo.

O algoritmo teve dificuldade de diferenciar letras maiúsculas de minúsculas (não o culpo, por vezes nem eu mesmo consigo), alcançando uma acurácia de aproximadamente 0,415. Por isso optei por utilizar apenas caracteres minúsculos, resultando em uma acurácia de 0,797.

A princípio achei que seria mais fácil, mas o trabalho se mostrou bastante desafiador. Só queria ter tido a ideia antes da existência do Google Lens.

Obs: todo o conteúdo do projeto se encontra disponível neste repositório do Google Drive.

Referências Bibliográficas

RAHMAN, Abid K. **Simple Digit Recognition OCR in OpenCV-Python**. 2012. Disponível em: <https://stackoverflow.com/questions/9413216/simple-digit-recognition-ocr-in-opencv-python> Acesso em: 06 jun. 2020.