Relatório Processamento de Imagem

Grupo: Matheus Willian e Vitor Pinheiro

Dividimos o código em funções de maneira a facilitar a leitura e utilização do mesmo.

Como executar o código

Para executar o código é necessário ter uma imagem qualquer que se deseja trabalhar, o código fonte disponível no github e então setar algumas variáveis globais no código se o que for desejado de executar for algo diferente do padrão.

Para executar utilize o comando:
python ./get_letters.py <caminho relativo ou absoluto para a imagem>

por padrão o algoritmo está executando os seguinte algoritmos com os seguintes parâmetros:

1. executa um threshold com um limite inferior de 100 para transformar a imagem em uma imagem binária
2. executa o algoritmo de dilatação com uma vizinhança de 3x3 quadrada
3. executa o algoritmo de erosão com uma vizinhança de 3x3 quadrada
4. segmenta a imagem para obter as letras
5. exibe o resultado delimitando com retângulos vermelhos os grupos encontrados.

Para alterar o threshold utilizado no passo 1 é necessário alterar a variável "THRESHOLD" para um número entre 1 e 254

Se as letras estiverem em branco, é necessário alterar a variável "USE_DEFAULT_COLOR" para False, dessa forma o algoritmo irá realizar os algoritmos dos passos, 2 e 3 de forma correta.

se não for necessário executar a dilatação é necessário alterar a variável "EXEC_DILATION" para False.

se não for necessário executar a erosão é necessário alterar a variável "EXEC_EROSION" para False.

se for desejado executar os algoritmos de erosão e dilatação com uma vizinhança maior é necessário alterar a variável "NEIGHBORHOOD" para o número desejado da seguinte forma:

- para executar com vizinhança 5x5, setar para 2
- para executar com vizinhança 7x7, setar para 3
- …

O algoritmo sempre será executado com uma vizinhança quadrada. Sendo essa uma possível melhoria para um trabalho futuro.

Por fim se for desejado alterar a ordem dos algoritmos 2 e 3 deve-se alterar a ordem de chamada das funções nas linhas 244 a 247, da seguinte forma:

- Pegar as linhas 244 e 255 e recortar seu conteúdo após o conteúdo da linha 247, dessa forma executando a erosão antes da dilatação.

Para separar palavras

Executar o algoritmo sem erosão e com uma dilatação que utilize uma vizinhança quadrada 5x5 ou 7x7. Dessa forma o algoritmo junta as letras e consegue separar as palavras em vez das letras.

Imports

Sys: Biblioteca necessária para passar a imagem como parâmetro na hora de chamar o código

Cv2: Biblioteca necessária para abrir a imagens, passar filtro cinza e outras coisas importantes

Numpy: Biblioteca necessária para percorrer a matriz da imagem, trabalhar com arrays de forma facilitada

Collections: Utilizamos o método de namedtuple para criar uma variável global chamada Pixel.

Matplotlib: Biblioteca necessária para plotar a imagem de uma maneira melhor

Variáveis Globais

RGB_RED = uma tupla com valor padrão de (255,0,0) que representa a cor vermelha para exibição dos resultados.

Black = Valor padrão de 0

White = Valor padrão de 255

Invalid = tem um valor padrão de 107 e é usada para a função de agregação reconhecer os pixels que já foram verificados.

Fit = valor padrão de 1 para fazer a verificação quando se vai realizar a erosão

Hit = valor padrão de 2 para fazer a verificação quando se vai realizar a dilatação

Hit_Rule = assumimos que um hit seria todo caso onde a verificação dos vizinhos encontrou pelo menos um pixel na verificação da regra (preto ou branco, depende do parâmetro passado)

Threshold = Valor padrão de 100 com intuito de ser um limiar como o nome diz.

Use_default_color = Variável booleana com intuito de facilitar o uso das funções onde é necessário passar uma cor como parâmetro porém o default está setado.

Exec_dilation = Variável booleana com intuito de facilitar o uso da função genérica de dilatação/erosão

Exec_erosion = Variável booleana com intuito de facilitar o uso da função genérica de dilatação/erosão

NEIGHBORHOOD = Valor padrão de 2 a fim de facilitar o uso das funções que podem receber como parâmetro a quantidade de vizinhos.

Explicando as Funções

get_black_white_image: Nesta função utilizamos o método copy da biblioteca opencv-python para copiar a imagem passada por parâmetro. Fazemos um looping dentro de outro para percorrer a matriz de pixels da imagem passada por parâmetro de maneira a verificar se o valor deste pixel é menor que o limite inferior(threshold) recebido por parâmetro, quando a condição é verdadeira, o pixel em questão recebe o valor 255, cor branca, e quando a condição não é verdadeira recebe 0, cor preta.

get_image: Nesta função utilizamos o método imread da biblioteca opencv-python para abrir as imagens definida pelo comando e passando como parâmetro a cor cinza, desta forma, as imagens que são abertas já estão em tons de cinza.

get_next_color: Função utilizada para pegar a cor que o grupo deve ser renderizado.

create_group: Retorna um array cheio, do tamanho do parâmetro passado e preenchido pelo valor passado também por parâmetro através da função do numpy.

segment: Esta função tem como objetivo varrer a matriz da imagem para encontrar as letras e/ou palavras e armazenar a posição extrema esquerda em cima e a posição extrema direita embaixo para passar como parâmetro para a função plot_grouped_image. Para isso, ela percorre a matriz enquanto chama a função de encontrar os vizinhos para saber se os pontos ao redor da imagem continuam num mesmo grupo criado, caso sim, ela percorre até encontrar o final do grupo a fim de salvar suas posições. Os grupos são criados através do primeiro valor de pixel encontrado no reconhecimento e vão sendo appendados à uma lista enquanto seus valores são constantes.

update_react: Esta função tem como objetivo atualizar nosso dicionário de posições utilizadas para desenhar o retângulo no reconhecimento de letra e/ou palavra.

neighborhood_array: Nesta função recebemos por parâmetro a imagem e a posição a qual precisamos olhar para retornar os vizinhos, assim como, também recebemos quantos vizinhos olhar. O default é retornar uma matriz 3x3 dos vizinhos.

check_neigh: Nesta função chamamos neighbourhood_array para olhar os vizinhos da posição passada e checamos se os mesmos validam um **FIT, HIT ou MISS**, ou seja, se todos os vizinhos possuem o valor esperado, podendo ser este preto ou branco, dependendo do parâmetro passado, ou se pelo menos 2 vizinhos possuem o valor esperado ou se nenhum vizinho possui o valor esperado.

erosion: Chamamos a função genérica __erode_or_dilate__, o propósito de erosion é receber por parâmetro se deseja aumentar a quantidade de vizinhos olhados, alterar a cor que erode e/ou a cor de quando se já miss.

dilate: Chamamos a função genérica __erode_or_dilate__, o propósito de dilate é o mesmo da erode, receber por parâmetro se deseja aumentar a quantidade de vizinhos olhados, alterar a cor que dilata e/ou a cor de quando se já miss.

__erode_or_dilate__: Nesta função recebemos por parâmetro se o modo que será calculado será o de erosão ou dilatação, com isso, chamamos a função de checar os vizinhos e com o retorno da mesma, verificamos se o modo passado foi de erosão ou dilatação para decidir se usaremos **FIT ou HIT** e assim alterar a imagem cópia que será retornada como parâmetro enquanto percorremos a imagem original verificando a cada ponto seus vizinhos para saber quando trocar a cor.

plot_grouped_image: Nesta função utilizamos a função do opencv rectangle a fim de formar um retângulo em volta da letra e/ou palavra reconhecida. Também usamos as funções do matplotlib para plotar a imagem de forma melhor.