UNIVERSIDADE DE SÃO PAULO
SISTEMAS DA INFORMAÇÃO

Anderson Giuseppe Saraiva Patriarca Fantin - 9016981
Larissa Fabião da Fonseca - 11208367
Lucas Imamura Alves - 11208221
Maria Fernanda Basso Santos - 11208197
Mateus Alex - 11207964
Rafael Silva de Lima - 11295790

COMPUTAÇÃO GRÁFICA — EXERCÍCIO PROGRAMA DE PROCESSAMENTO DE IMAGENS

SÃO PAULO
2022

1 INTRODUÇÃO

O presente relatório tem como objetivo descrever o objetivo e processo de desenvolvimento do projeto de identificação do resultado de testes rápidos para identificar a contaminação por covid-19 desenvolvido para a disciplina de Computação Gráfica da Escola de Artes, Ciências e Humanidades da Universidade de São Paulo.

A proposta inicial do projeto foi desenvolver um algoritmo que utiliza se técnicas de processamento de imagens para identificar o número de riscos vermelhos presentes no cartucho registrado na imagem para assim determinar um resultado positivo (dois riscos) não negativo (um risco). Para isso, 73 imagens foram coletadas para criar uma base de dados levando em consideração uma variedade de testes positivos e negativos, angulação e iluminação do objeto de interesse de modo a garantir que os testes contemplassem uma diversidade de cenários.

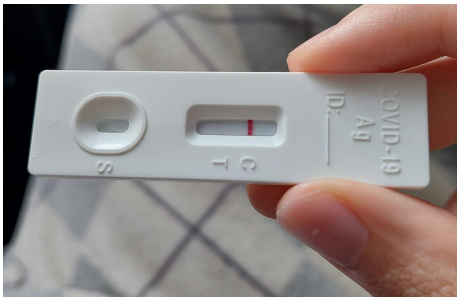
Além disso, a linguagem de programação python foi utilizada junto da versão compatível com esta linguagem da biblioteca OpenCV. A instalação dos pré requisitos e a execução do projeto serão discutidos posteriormente na seção 3 deste relatório.

2 DESENVOLVIMENTO

O primeiro passo para construir o projeto de identificação dos testes rápidos de covid-19 constituiu na construção de base de dados com fotos de testes em diversas condições de luminosidade, posição e também clareza dos resultados. Uma vez que não foram encontradas bases de dados com estas características, as imagens foram coletadas manualmente. Para isso, todos os integrantes do grupo responsável pelo presente projeto coletaram imagens de testes que já haviam realizados e também contamos com a colaboração de outros colegas de classe para enviarem fotos dos testes sempre que precisassem realizar um.

Após coletar as imagens, todas as imagens foram classificadas de acordo com o resultado do teste presente na mesma. Ao fim deste processo, 73 imagens foram coletadas e classificadas de acordo com os requisitos apresentados previamente.

Imagem 1: Exemplo de uma imagem coletada contendo um resultado negativo.



Fonte: imagem do banco de dados criado no presente trabalho

Imagem 2: Exemplo de uma imagem coletada contendo um resultado positivo.

Fonte: imagem do banco de dados criado no presente trabalho

Em seguida, nossa primeira tentativa foi a de converter a imagem para escala de cinza através do método *cvtColor* da biblioteca OpenCV, aplicar o algoritmo de Canny para detecção de bordas e por fim utilizar uma máscara para remover todos os objetos que não são vermelhos. Com a imagem resultante desse processo, um algoritmo de reconhecimento de retângulos através dos contornos encontrados era utilizado a fim de buscar pelas linhas que indicam o resultado do teste. No entanto, não foi muito eficiente, dado que para os testes positivos tinha um baixo acerto, isso porque a segunda linha de teste (a que significa o positivo) normalmente é bem mais fraca e o algoritmo de Canny acabava não a reconhecendo como uma borda.

Também tentamos utilizar a Transformada de Hough para detectar retas, mas não foi muito eficiente dado que ainda havia muito vermelho e então supostas linhas na imagem. Além disso, para identificar o que seriam as linhas de interesse, as áreas dos retângulos também são calculadas e as maiores ou muito menores áreas eram descartadas quando mais de dois retângulos eram encontrados. Concluindo, esta primeira abordagem mostrou-se falha dado o grande número de objetos que estavam sendo considerados como linhas de teste. Isto ocorreu devido a interferência de outros objetos que eram confundidos com os traços vermelhos

desejados e da grande variação no tamanho do teste em relação à totalidade da imagem (e consequentemente de suas linhas).

Normalmente, o fundo das imagens cuja análise falhava continham um grande número de objetos vermelhos que acabam sendo confundidos com as duas pequenas linhas vermelhas buscadas. Desta maneira, a alternativa encontrada para finalizar a análise foi a de recortar as imagens de forma que somente o objeto de interesse ficasse em destaque e assim eliminar grande parte dos objetos que poderiam ser confundidos com o resultado do teste.

Imagem 2: resultado da Imagem 1 após o recorte manual do objeto de interesse.

Fonte: imagem do banco de dados criado no presente trabalho

Com esta nova abordagem, aplica-se uma conversão das cores da imagem para HSV a fim de facilitar a aplicação de uma máscara para filtrar somente os objetos vermelhos. Além disso, a aplicação de algoritmos de detecção de retângulos é feita para poder encontrar e contar o número de objetos identificados e assim chegar a conclusão do resultado da análise de imagem. Para finalizar os testes, realizamos uma comparação entre os resultados obtidos com o algoritmo de processamento de imagens e o esperado de acordo com a base de dados produzida. Desta maneira, o presente trabalho obteve 90,41% de acertos.

3 EXPLICAÇÃO DO ALGORITMO

Nesta seção, o algoritmo desenvolvido será explicado com mais detalhes a fim de explorar as técnicas e lógicas utilizadas. Devido a escolha de utilizar o python como linguagem de programação, o projeto possui apenas um arquivo, denominado main.py, que recebe a função main assim como seu nome. Além disso, o código foi dividido nos métodos *“__init__”*, “*list_files”, “find_red”, “find_lines”, “draw_found_rectangles”, “write_results” e “make_result_dataframe” que serão explicadas com mais detalhes a seguir.*

O método *“__init__”* centraliza as principais rotinas do algoritmo, realizando as chamadas dos demais métodos. Primeiro, o método “*list_files”* é chamado para listar todos os caminhos dos arquivos de teste (que começam com “teste“) que estão no mesmo diretório do arquivo main.py, além de também retornar qual é o caminho do diretório.

Então, cria um data frame (final) para juntar os data frames obtidos para cada imagem, e para cada caminho de imagem de teste, são realizadas as chamadas das demais funções auxiliares *“find_red”, “find_lines”, “draw_found_rectangles”,* aqui o algoritmo salva a imagem criada pelo método anterior e depois chama *“write_results”*, em seguida, a última coisa que acontece neste *for* é a junção do data frame que foi obtido a partir da chamada dessas funções (ou seja, para essa imagem) com o data frame externo (final). Por último, fora do *for*, com os dados unificados de todas as imagens nesse data frame final é chamado o último método, *“make_result_dataframe”*.

O método *“find_red”* é então chamado para encontrar os objetos vermelhos na imagem. Para isso, primeiro ele lê a imagem e converte as cores da imagem para HSV a fim de facilitar a manipulação dos valores. Em seguida, cria as máscaras mask1 e mask2 com os valores máximos e mínimos para vermelho em HSV e por fim realiza a junção das máscaras a partir de uma operação bitwise or.

Após obter a imagem resultante do filtro de objetos vermelhos, a função *“find_lines”* é chamada. Como o próprio nome do método sugere, o objetivo desse passo é identificar as linhas na imagem. Entretanto, antes de iniciar esse processo descarta-se 20% das imagens (nas extremidades), pois o resultado dos testes sempre estarão centralizados nos cartuchos e dificilmente serão encontrados próximos à borda da imagem. Após realizar esta operação, buscamos os contornos na imagem retornada por *“find_red”.*

Para cada contorno encontrado, calculamos a área aproximada do objeto formado por este contorno e criamos um retângulo equivalente do contorno a partir da posição dele na imagem e sua altura e largura. Com estes dados, o algoritmo ignora os retângulos que não tenham área maior que 3 e menor que 2000 ou que a área aproximada do contorno não ocupa mais de 20% da área do retângulo pois estes casos não se enquadram no perfil identificado pelas linhas do teste. Cada um dos retângulos encontrados na imagem é adicionado como uma linha dentro do

objeto da biblioteca pandas denominado data frame que agrega todos os dados obtidos do processamento das imagens.

Caso para uma mesma imagem existam mais de dois retângulos, ou linhas, no data frame, então significa que o algoritmo encontrou mais objetos vermelhos que o esperado e então os retângulos que contém as prováveis linhas tem que ter uma altura e largura maior que 5 e área aproximada de seu contorno maior que 50. Se após esse filtro ainda existir mais de um retângulo, identificamos se a altura do retângulo encontrado é maior que a largura para identificar a orientação do teste (vertical ou horizontal).

Para o caso em que a altura é maior que a largura, se o centro da altura da imagem está contido no retângulo em relação a sua altura então é armazenado no data frame que a distância entre o retângulo e o centro da imagem é zero (em relação a altura). Caso contrário, calcula a distância em módulo do ponto inicial (y) e final do centro da imagem em relação a altura e se a variável em relação ao início do retângulo é menor que a em relação ao final dele, então guarda nessa linha do data frame a distância entre a variável em relação ao início do retângulo e o centro da imagem (em relação a altura) ou guarda a distância entre a variável em relação ao final do retângulo e o centro da imagem (em relação a altura), caso contrário.

No caso de a largura ser maior que a altura, se o centro da largura da imagem está contido no retângulo em relação a sua largura, então guarda no data frame a distância entre o retângulo e o centro da imagem como zero (em relação a largura). Na situação contrária, calcula a distância em módulo do ponto inicial (x) e final do centro da imagem em relação a largura e caso a variável em relação ao início do retângulo é menor que a em relação ao final dele, então guarda no data frame a distância entre a variável em relação ao início do retângulo e o centro da imagem (em relação a largura), senão, guarda nessa linha do data frame a distância entre a variável em relação ao final do retângulo e o centro da imagem (em relação a largura).

Ao fim deste processo, as linhas do data frame são ordenadas a partir da menor distância calculada anteriormente (ordem crescente) e se em alguma linha ela é maior que 50, essa linha é removida, uma vez que espera-se que as linhas estejam próximas ao centro da imagem. Caso após esse processo ainda existam mais de uma linha de teste, então comparamos os retângulos (linhas do data frame)

par a par, primeiro se no processo de cálculo citado anteriormente foi utilizado o mesmo parâmetro (altura ou largura).

Depois, se a altura foi utilizada, verifica-se a proximidade de seus pontos y, se eles não têm mais de 3 de diferença, logo eles devem ser as linhas positivas do teste (levando em conta também a ordenação citada anteriormente de menor distância do centro). Na situação de a largura ter sido utilizada, verifica-se a proximidade de suas alturas, se elas tiverem menos de 2 de diferença, então também podemos dizer que essas são as linhas de teste (ordenação também presente).

Agora, no método “draw_found_rectangles”, desenha-se na imagem original os retângulos encontrados para que facilitar a análise dos resultados pelo grupo e demonstrar o que foi encontrado como as linhas do teste e depois, como citado anteriormente, essa imagem é salva em uma pasta chamada resultados. Após isso, o método *“write_results”* foi usado para finalmente escrever no data frame o resultado encontrado pelo algoritmo: erro de leitura (nenhuma linha foi encontrada), negativo (uma linha foi encontrada), positivo (duas linhas foram encontradas) e inconclusivo (mais de duas linhas foram encontradas).

Por fim, no método *“make_result_dataframe”*, para comparar os resultados obtidos com os esperados, realizamos uma análise do data frame com a planilha criada pelo grupo com os resultados reais, então um arquivo denominado resultado.xlsx receberá esse data frame final que contém o nome do teste, a classificação dada pelo algoritmo, a classificação esperada, se o algoritmo acertou (1) ou não (0) e também o cálculo do percentual de acertos (com duas casas decimais) na última linha da planilha (na coluna da verificação de acertos).

4 EXECUÇÃO DO PROJETO

Como pré-requisito para a execução contamos com a prévia instalação do python 3.10 configurado e também do **pip** em sua versão mais recente. Uma vez que python não é uma linguagem compilada, o projeto inclui um shell script denominado ***exec.sh*** e um denominado ***exec_python2.sh*** que possui a rotina para instalar as dependências do projeto e em seguida executá-lo, para isso deve-se utilizar o primeiro arquivo caso o nome da instalação do python seja python3 (padrão do linux) e o segundo para caso o nome esteja somente como python

(normalmente para sistemas Windows). Dentro desse shell script temos os comandos para fazer o download dos pré-requisitos e rodar o código.

Caso prefira rodar manualmente, os seguintes comandos precisam ser executados e caso esteja utilizando uma versão anterior do python, utilize a opção 2.a ao invés da 2.

1. Instalar as dependências:
   pip install requirements.txt
2. Invocar o método main:
   python3 main.py
   a. Invocar o método main para um python anterior:
      python main.py

5 CONCLUSÃO

Após o desenvolvimento do presente trabalho, o algoritmo desenvolvido teve uma taxa de acerto de 90,41%, o que indica um bom resultado dada a complexidade das imagens analisadas. Identificou-se também que os casos de falha estão relacionados a imagens com uma baixa luminosidade, casos cujas linhas resultantes no teste ficaram muito fracas ou inexistentes e casos em que muitos objetos semelhantes foram encontrados na imagem. Desta maneira, o resultado obtido com o trabalho foi satisfatório uma vez que o objetivo inicial de criar um algoritmo capaz de classificar o resultado de um teste de infecção por covid-19 utilizando técnicas de processamento de imagens foi obtido com um bom percentual de acerto.

Ao fim do projeto, o algoritmo desenvolvido se mostrou capaz de identificar o resultado de imagens que contenham um único teste cujo resultado é apresentado por linhas vermelhas. Além disso, é capaz de identificar o resultado independente do formato do cartucho de testes. Por fim, com o percentual de acertos obtidos, é conhecido que caso as imagens sejam feitas com fundos que não contenham um grande número de objetos vermelhos, então a chance de acerto do resultado é ainda maior.

6 REFERÊNCIAS DE APOIO PARA O CÓDIGO

Para detectar o que é vermelho na imagem:

https://stackoverflow.com/questions/51225657/detect-whether-a-pixel-is-red-or-not/51228567#51228567

Para encontrar contornos e o retângulo referente a cada contorno (e desenhá-lo na imagem):

https://www.pythonpool.com/cv2-boundingrect/