RELATÓRIO SOBRE O SEGUNDO TRABALHO DE PROCESSAMENTO DIGITAL DE SINAIS E IMAGENS

June 23, 2019

Kevin Levrone Rodrigues Machado Silva
Christian Takashi Nakata
Universidade Estadual de Maringá
Departamento de Informática

Contents

0.1 Objetivo e descrição do problema

A correlação é uma técnica de realce espacial pertencente à área de processamento de imagens. Considerando uma imagem f(x) e uma máscara g(x), a correlação é o processo de mover g(x) por f(x) e calcular a soma dos produtos em cada posição de f(x). A soma dos produtos deve ser atribuída na posição de f(x) que corresponde ao centro de g(x) no dado momento.

Este trabalho almeja comparar duas formas diferentes de implementação da correlação. A primeira forma se assemelha à descrição do parágrafo anterior, pois é preciso processar a imagem, pixel a pixel, posicionando a máscara centrada em cada pixel e realizando as multiplicações e a soma resultante para este pixel.

Já a segunda consiste da translação da imagem por vetores definidos pelas posições da máscara, relativas ao centro desta. Cada posição define um vetor que, ao ser invertido, será responsável pela translação da imagem. Após transladar a imagem pelo vetor associado à posição p da máscara, é necessário multiplicar todos os elementos desta imagem transladada pelo peso atribuído à máscara na posição p. Para cada posição da máscara, é preciso fazer uma translação conforme a descrição. Por fim, as matrizes transladadas resultantes são todas somadas.

O parâmetro de comparação entre as duas abordagens citadas será o tempo de processamento que cada uma delas leva para ser concluída.

0.2 Ferramentas e recursos utilizados

Para o desenvolvimento deste trabalho, foram utilizadas as seguintes ferramentas:

- A linguagem de programação Python (Versão 3);
- A biblioteca Numpy (Versão 1.16.3);
- A biblioteca Time para a medição do tempo de processamento para cada abordagem de correlação;
- O programa OpenCV (versão 4.1.0) para carregar, salvar e mostrar as imagens utilizadas neste projeto.

0.3 Código-Fonte

O código a seguir foi implementado em Python com o intuito de carregar uma imagem, converte-lá para tonalidade de cinza e aplicar as duas operações de correlação. Ao executar o código, será pedido para que o usuário insira um número para selecionar o método de correlação desejada (1 para a correlação normal e 2 para a correlação por translação de imagem).

Assim que uma das abordagens for selecionada, uma função da biblioteca Time será executada para iniciar a contagem do tempo de processamento. Assim que a execução do método for finalizada, será retornado ao terminal o tempo total de processamento que a abordagem levou para realizar a correlação na imagem de entrada.

```
import numpy as np
import cv2 as cv
import time

def trataBordas(matriz):
    return np.pad(matriz, pad_width=1, mode='constant', constant_values=0)

def soma_e_multiplicacao(matriz, mascara, i, j):
    return (matriz[i-1][j-1] * mascara.item(0)) + (matriz[i-1][j] * mascara.item(1))



def convolucaoNormal(matriz, mascara):

    altura, largura = matriz.shape
    matriz_tratada = trataBordas(matriz)
    altura_tratada, largura_tratada = matriz_tratada.shape
    matriz_convolucao = np.empty([altura_tratada, largura_tratada])

    print("Matriz preenchida com zeros:")
    print(matriz_tratada)

    for i in range(1, altura_tratada - 1):
        for j in range(1, largura_tratada - 1):
            matriz_convolucao[i][j] = soma_e_multiplicacao(matriz_tratada, mascara, i
```

```
    return matriz_convolucao

def transladaMatriz(matriz, valor_pad_vertical, valor_pad_horizontal, elem_mascara):

    if(valor_pad_vertical == 1):
        matriz = np.roll(matriz, 1, axis=0)[1:, :]  # shift down e apaga primeira li
        matriz = np.pad(matriz, ((1,0),(0,0)), mode='constant', constant_values=0) #

    elif(valor_pad_vertical == -1):
        matriz = np.roll(matriz, -1, axis=0)[:-1, :]  # shift up e apaga última linh
        matriz = np.pad(matriz, ((0,1),(0,0)), mode='constant', constant_values=0) #

    if(valor_pad_horizontal == 1):
        matriz = np.roll(matriz, 1, axis=1)[:, 1:]  # shift right e apaga coluna esq
        matriz = np.pad(matriz, ((0,0),(1,0)), mode='constant', constant_values=0) #

    elif(valor_pad_horizontal == -1):
        matriz = np.roll(matriz, -1, axis = 1)[:, :-1] # shift left e apaga coluna d
        matriz = np.pad(matriz, ((0,0),(0,1)), mode='constant', constant_values=0) #

    altura, largura = matriz.shape

    matriz = matriz * elem_mascara

    return matriz

def convolucaoTranslação(matriz, mascara):
```

```
    altura, largura = matriz.shape

    matrix_resultante = np.empty([altura, largura])

    matrix_resultante = transladaMatriz(matriz, 1, 1, mascara.item(0))
    matrix_resultante += transladaMatriz(matriz, 1, 0, mascara.item(1))
    matrix_resultante += transladaMatriz(matriz, 1, -1, mascara.item(2))
    matrix_resultante += transladaMatriz(matriz, 0, 1, mascara.item(3))
    matrix_resultante += transladaMatriz(matriz, 0, 0, mascara.item(4))
    matrix_resultante += transladaMatriz(matriz, 0, -1, mascara.item(5))
    matrix_resultante += transladaMatriz(matriz, -1, 1, mascara.item(6))
    matrix_resultante += transladaMatriz(matriz, -1, 0, mascara.item(7))
    matrix_resultante += transladaMatriz(matriz, -1, -1, mascara.item(8))

    print("Matriz Convolução: ")
    print(matrix_resultante)

    return matrix_resultante


def main():

    teste = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

    imgLena = cv.imread('./images/lena_original.png', 0)
    # imgLena = cv.imread('./images/cat.png', 0)


    cv.imshow('Leninha', imgLena)
    cv.waitKey(0)
    cv.destroyAllWindows()

    print("Matriz da imagem:\n")
    print(imgLena)
```

```
mascara = np.matrix([[-1,-1,-1], [2,2,2], [-1,-1,-1]])    # detecção de linha (h

print("\nMáscara 3x3:\n")
print(mascara)
print()

imagem_convolucao = [[]]

print("Digite o número da convolução desejada:")
op = int(input("1 - Normal\n2 -Translação de imagem\n"))


while(op != 1 and op != 2):
    print("Escolha um valor entre 0 e 1:")
    op = input("1 - Normal\n2 -Translação de imagem\n")

if(op == 1):
    print("Convolução Normal selecionada!")
    start = time.time()
    imagem_convolucao = convolucaoNormal(imgLena, mascara)

elif(op == 2):
    print("Convolução por Translação de imagem selecionada!")
    start = time.time()
    imagem_convolucao = convolucaoTranslação(imgLena, mascara)


alt, lar = imagem_convolucao.shape
maior_tonalidade = np.amax(imagem_convolucao)
img_conv2 = np.empty([alt, lar])

for i in range(alt):
    for j in range(lar):
```

```
            img_conv2[i][j] = (imagem_convolucao[i][j]/maior_tonalidade) * 255

    nova_img = img_conv2.astype(np.int8)
    end = time.time()

    print("Imagem Convolução Normalizada: ")
    print(nova_img)

    print("Tempo total de execução: " + str(end - start))

    cv.imshow('lena convolucao', nova_img)
    cv.waitKey(0)
    cv.destroyAllWindows()

    cv.imwrite('./images/lena grayscale.png', imgLena)
    cv.imwrite('./images/lena convolucao.png', nova_img)

main()
```

0.4 Imagens Resultantes

Como pode ser visto no código-fonte da seção anterior, a imagem carregada para os testes com as duas abordagens é a "lena_original.png", cuja resolução é de 420x440. Originalmente, a imagem é colorida (RGB), mas foi convertida para *grayscale* no momento em que foi carregada, como mostra a Figura 3. O mesmo procedimento foi aplicado na Figura 5, que possui resolução de 733x490.

A máscara utilizada nos experimentos é representada na Figura 1. Quando aplicada na correlação, é gerada a mesma imagem, mas com suas linhas horizontais detectadas e em evidência. Os experimentos foram executados em uma distribuição Ubuntu 16.04 do sistema operacional operacional Linux. As especificações da máquina consistem em um processador i5 8ª geração, placa de vídeo MX150 com 2GB dedicados e memória RAM de 8GB.

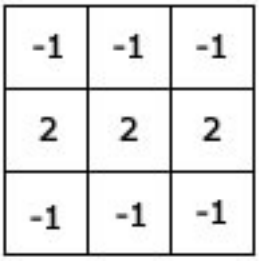

Figure 1: Máscara para detecção de linhas horizontais

Table 1: Tempo de processamento para cada abordagem de convolução para a Figura 3

Abordagem	**Tempo de Processamento (segundos)**
Convolução Normal	8.07
Convolução por Translação de Imagem	0.17

Table 2: Tempo de processamento para cada abordagem de convolução para a Figura 6

Abordagem	**Tempo de Processamento (segundos)**
Convolução Normal	16.41
Convolução por Translação de Imagem	0.35

Como pode ser observado na Tabela 1 e na Tabela 2, a convolução por translação de imagem possui um desempenho superior ao método Normal de convolução. Ambas geraram o mesmo tipo de imagem com as linhas horizontais em destaque para a Lena (Figura 4) e para o gato (Figura 7.

Figure 2: Lena RGB

Figure 3: Lena em *grayscale*

Figure 4: Lena pós-convolução

Figure 5: Cat RGB

Figure 6: Cat em *grayscale*

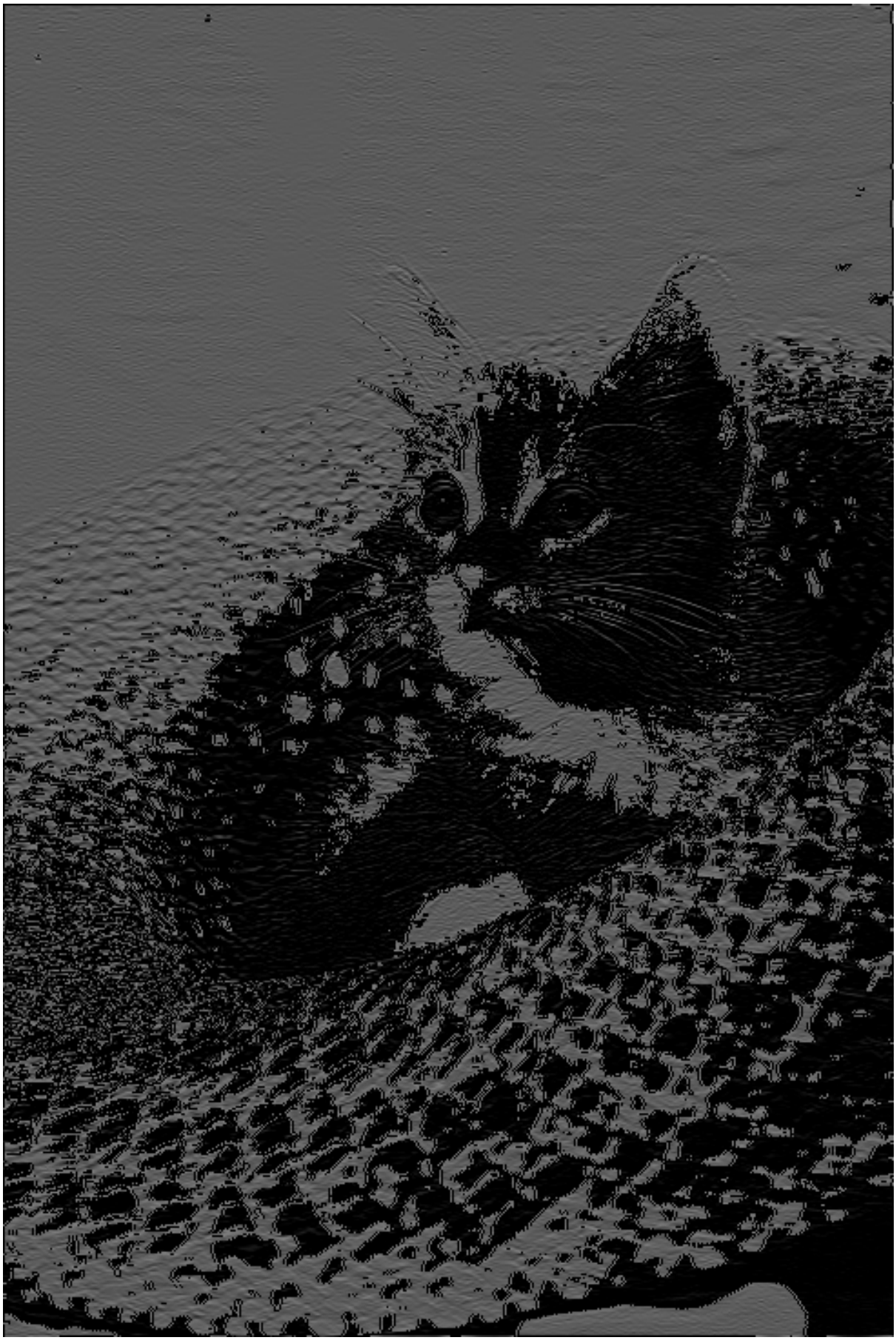

Figure 7: Cat pós-convolução