UNIVERSIDADE ESTADUAL DE MARINGÁ

MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

PROCESSAMENTO DIGITAL DE IMAGENS

RELATÓRIO DE TRABALHOS DESENVOLVIDOS NA DISCIPLINA

Alunos:
Leonardo Gabiato Catharin
Rafael Biazus Mangolin

20 de Novembro, 2018

Leonardo Gabiato Catharin
Rafael Biazus Mangolin

RELATÓRIO DE TRABALHOS DESENVOLVIDOS NA DISCIPLINA

Implementação de técnicas de processamento digital de imagens aprendidas durante a disciplina.

20 de Novembro, 2018

1 Introdução

Como apresentado em Gonzalez and Woods [2008], a visão é o sentido mais avançado do ser humano. Por causa disso, as imagens exercem uma função muito importante para percepção humana. Isso motiva o estudo dessas imagens a fim de utiliza-las e processa-las. Mesmo que a visão humana faça as pessoas perceberem o mundo a sua volta, ela é limitada e não consegue captar tudo o que está a nossa volta. Partindo dessa limitação visual, os aparelhos de processamento de imagens auxiliam a capturar todos os tipos de imagens, variando de ondas gama a ondas de rádio. Nesse sentido área de processamento digital de imagem (PDI) tem aplicado seus esforços. Nessa área é possível encontrar diversas técnicas de processamento de imagens como filtros, técnicas de compressão, aplicações matemáticas.

Diante desse contexto, neste trabalho foram implementadas e testadas três técnicas muito populares na área de PDI, a saber: Chroma subsampling, transformações afins e coloração artificiais de imagens. Todas essas técnicas são explicadas contendo ainda o código desenvolvido ao longo do trabalho e um exemplo para facilitar o entendimento de suas aplicações. Todos os códigos fonte foram desenvolvidos utilizando a linguagem *python* e algumas de suas bibliotecas como a *opencv* e a *numpy*.

Na Seção 2 são apresentadas as técnicas abordadas neste trabalho. Ja na Seção 3 é apresentada uma sucinta conclusão do trabalho. Por fim, no final do trabalho são apresentadas as referências que serviram de base para o desenvolvimento do mesmo.

2 Desenvolvimento

Nesta seção são apresentados as técnicas de processamento de imagem digital aplicadas neste trabalho. São elas: Choma subsampling, tranformações afins e coloração artificiais de imagens.

2.1 *Chroma subsampling*

O *Chroma subsampling* é uma técnica de compressão de imagem muita utilizada. Ela aproveita que o olho humano é menos sensível à cor do que à luminosidade e reduz a quanidade de cores da imagem. Dessa maneira, a imagem se torna "mais leve", sem ter uma perda significativa de qualidade

da imagem perceptível a olho nu. O código abaixo foi desenvolvido neste trabalho para aplicar essa técnica.

```
import numpy as np
import cv2
from PIL import Image

imagem = 'exemplo.jpeg'
def chroma_subsampling(j,a,b):
        original = cv2.imread(imagem)
        img_1 = cv2.cvtColor(original, cv2.COLOR_BGR2YCrCb)
        Y, Cr, Cb = cv2.split(img_1)

        x = int(j/a)
        y = None
        if(int(b) == 0):
                y = 2
        else:
                y = 1

        cb = Cb[::y,::x]
        cb = np.repeat(cb,x,axis=1)
        cb = np.repeat(cb,y,axis=0)
        cb = cb[:len(Y),:len(Y[0])]

        cr = Cr[::y,::x]
        cr = np.repeat(cr,x,axis=1)
        cr = np.repeat(cr,y,axis=0)
        cr = cr[:len(Y),:len(Y[0])]

        img = cv2.merge([Y,cr,cb])
        img = cv2.cvtColor(img, cv2.COLOR_YCrCb2BGR)
        cv2.imwrite('{}:{}:{}_{}'.format(j,a,b,imagem),img)

chroma_subsampling(4,2,2)
```

Como pode ser visto no código acima, a função *chromasubsampling(j, a,*

b) aplica a técnica na imagem. O primeiro passo, é a conversão do modelo de cor da imagem original de RGB para YCrCb. Dessa maneira, é possível realizar operações na luminância (banda Y) e crominância (bandas Cb e Cr) da imagem separadamente. Para essa técnica, mantém-se a luminância intacta enquanto as operações são realizadas sobre a crominância. Após a conversão, é realizado a duplicação dos *pixels* no eixo horizontal e no eixo vertical respectivamente. É necessário realizar essa operação em ambas as bandas do modelo, Cb e Cr, já que as duas são a crominância da imagem. Em seguida, basta juntar as três bandas e fazer a conversão do modelo YCbCr para RGB. Na Figura 1 é apresentado um exemplo da aplicação dessa técnica. A Figura 1.a é a imagem original e a Figura 1.b é o resultado após a aplicação da técnica. Como já dito, a olho nu, não é possível perceber diferença. Utilizamos as metricas *Mean Square Error* (MSE) e *Peak Signal-to-Noise Ratio* (PSNR) para realizar a avaliação dos resultados conforme apresentado na Tabela 1. Na Tabela 1, podemos identificar que a variação do *chroma subsampling* 4:4:4 foi que apresentou menores taxas de erro, mas deve ser considerado que ele não aplica nenhuma modificação na imagem, assim o erro apresentado é devido a conversão de RGB para YCbCr. As outras variações apresentam um resultado similar, pode ser destacada a variação 4:2:0 que reduz o tamanho da imagem em aproximadamente 50% e apresenta um taxa de aceitável.

Figura 1: Exemplo da aplicação da técnica Chroma Subsampling

(a) Imagem original

(b) Imagem pós aplicação do chroma subsampling 4:2:2

Tabela 1: Taxa de perda calculada utilizando MSE e PSNR.

	MSE	PSNR
4:1:1	37.333	32.409
4:2:0	23.937	34.340
4:2:2	16.097	36.063
4:4:4	0.3789	52.345

Fonte: Autoria própria

2.2 *Transformações Afins*

As transformações aplicadas em um objeto com o fim de modifica-ló podem ser chamadas de transformações afins Azevedo et al. [2018]. As principais transformações são: translação (T), rotação (R) e mudança de escala (E). Para aplicação dessas transformações no plano de duas dimensões podemos descreve-las pelas matrizes:

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}, R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} E = \begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Para realizar a translação deve ser informado para qual ponto (t_x, t_y) o objeto deve ser movido. Na rotação deve se informado o angulo θ que o objeto irá rotacionar. Já a mudança de escala deve ser informado a nova escala do objeto $c_x \times c_y$. Neste trabalho realizamos a operações de mudança de escala e rotação em imagens. O código abaixo apresenta as operações para se realizar a mudança de escala em imagens.

```
import cv2
import numpy as np
from math import cos, radians, sin


def multiplicar_ponto_matrix(x,y,mtr):
        ponto = np.array([x,y,1])
        return np.matmul(mtr,ponto).astype(int)

def matriz_escala(x,y):
        return [[x, 0, 0],
                [0, y, 0],
                [0, 0, 1]]

def escala(nome_imagem, formato):
        imagem  = cv2.imread(nome_imagem)
        altura  = len(imagem)
        largura = len(imagem[0])
        mtz_esc = matriz_escala(formato[0], formato[1])

        ponto = multiplicar_ponto_matrix(largura,altura,mtz_esc)
        x_max = ponto[0]
        y_max = ponto[1]

        new = np.zeros((y_max,x_max,3), np.uint8)
        for i in range(len(imagem)):
                for j in range(len(imagem[0])):
                        res = multiplicar_ponto_matrix(j,i,mtz_esc)
                        new[res[1],res[0]] = imagem[i,j]

        mtz_esc_inv = matriz_escala(1/formato[0],1/formato[1])
        for i in range(len(new)):
                for j in range(len(new[0])):
                        res = multiplicar_ponto_matrix(j,i,mtz_esc_inv)
                        new[i,j] = imagem[res[1],res[0]]

        cv2.imwrite('escala_{}x{}_{}'.format(formato[0],formato[1],nome_imagem),new)

if __name__ == '__main__':
        escala('lena.png',(1,.5))
        escala('lena.png',(.5,1))
```

Na função *escala* ocorre a operação para a mudança de escala, ela recebe como parâmetro o endereço da imagem e a nova escala. Após obter informações sobre a imagem fornecida (altura, largura) é obtida a matriz de mudança de escala no formato informado. Para gerar a nova imagem é necessário obter a localização do pixel mais distante, assim se tem as dimensões da nova imagem. O primeiro laço realiza o deslocamento dos *pixels* para os seus devidos lugares na nova escala. Essa operação quando aumenta a escala gera uma esparsidade dos *pixels*, para realizar a correção desse problema

deve ser aplicado um mapeamento da nova imagem para a imagem original utilizando a transformação de escala inversa. O resultado desses operações é ilustrado na Figura 2, em que foram aplicadas escalas 1×0.5 (Figura 1.a) e 0.5×1 (Figura 1.b) na imagem original representada na Figura 1.a.

Figura 2: Exemplo da aplicação da técnica de mudança de escala

(a) Imagem com escala 1×0.5.

(b) Imagem com escala 0.5×1.

A outra transformação afim implementada neste trabalho foi a rotação, o código abaixo foi implementado para realizar a rotação em imagens.

```
import cv2
import numpy as np
from math import cos, radians, sin

def rotacao(imagem_nome, ang):
        imagem    = cv2.imread(imagem_nome)
        angulo    = radians(ang)
        altura    = len(imagem)
        largura   = len(imagem[0])
        dy              = int(len(imagem)/2)
        dx              = int(len(imagem[0])/2)

        mul     = matriz_rotacao(angulo)
        ida     = matriz_translacao(-dx, -dy)
        volta   = matriz_translacao(dx, dy)
        mtz_rot = np.matmul(mul,ida)
        mtz_rot = np.matmul(volta,mtz_rot)

        alt_img_rot, lar_img_rot, sobra_y, sobra_x = parametros_rotacao(largura,altura,mtz_rot)
        imagem_nova = np.zeros((alt_img_rot+1,lar_img_rot+1,3), np.uint8)
        for i in range(len(imagem)):
                for j in range(len(imagem[0])):
                        res = multiplicar_ponto_matriz(i,j,mtz_rot)
                        imagem_nova[res[0]+sobra_y,res[1]+sobra_x] = imagem[i,j]

        alt_img_rot = len(imagem_nova)
        lar_img_rot = len(imagem_nova[0])
        dy                      = int(len(imagem_nova)/2)
        dx                      = int(len(imagem_nova[0])/2)

        mul     = matriz_rotacao(radians(-ang))
        ida     = matriz_translacao(-dx, -dy)
        volta   = matriz_translacao(dx, dy)
        mtz_rot = np.matmul(mul,ida)
        mtz_rot = np.matmul(volta,mtz_rot)

        for i in range(len(imagem_nova)):
                for j in range(len(imagem_nova[0])):
                        res = multiplicar_ponto_matriz(i,j,mtz_rot)
                        x = res[0]-sobra_x
                        y = res[1]-sobra_y
                        if(x >= 0 and x < altura and y >= 0 and y < largura):
                                imagem_nova[i,j] = imagem[x,y]

        cv2.imwrite('angulo_{}_{}'.format(ang,imagem_nome),imagem_nova)

rotacao('lena.png',45)
rotacao('lena.png',-45)
```

Para realizar a rotação de imagens precisamos identificar o centro da imagem (dx e dy), pois assim ao invés de realizar a rotação no ponto $(0,0)$ da imagem, transladamos o centro da imagem para o ponto $(0,0)$ realizamos a rotação e depois voltamos ela para o centro da nova imagem. Com essa ideia definimos as funções *matriz_rotacao* e *matriz_translacao* que retorna as matrizes para realizar a rotação, a ida do centro para o $(0,0)$ contida na variável *ida*, a matriz de rotação (mul) e a matriz para retornar a imagem

(*volta*). Com as três matrizes realizamos a multiplicação delas para reunir todo o processo de rotação em uma única matriz (*mtz_rot*).

Para não gerar uma imagem rotacionada cortando as pontas, é criado uma nova imagem vazia com as dimensões da imagem rotacionada. Obtemos essas dimensões pela função *parametros_rotacao*, na qual a partir dos pontos de extremo da imagem ela identifica para a nova imagem os pontos de mínimo e máximo dos eixos x e y. Assim, identifica a largura e altura da nova imagem, além de informar o quanto os *pixels* tem que se moverem para se encaixarem na nova imagem.

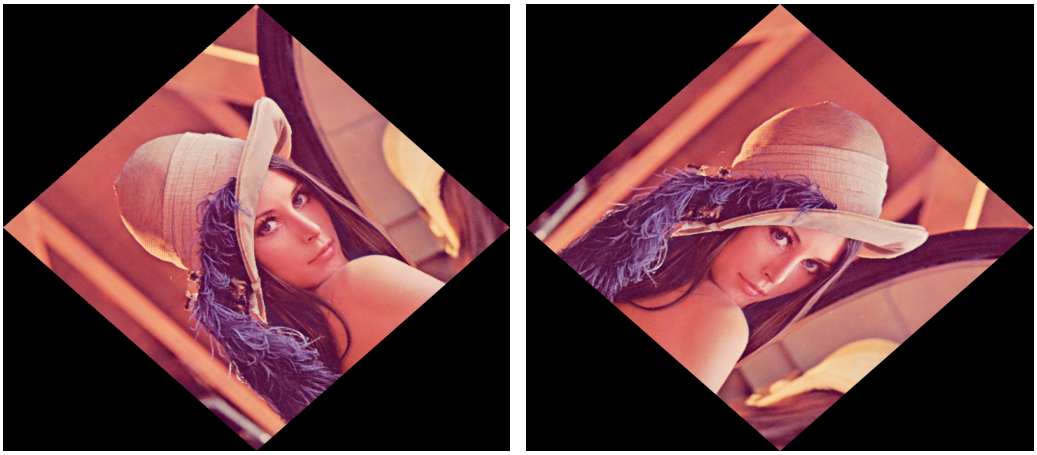
No primeiro laço é realizado a rotação da imagem. Nessa operação também está contida as translações de ida e volta da imagem. Nesse laço também é aplicado a correção para a rotação estar centralizada na nova imagem. Assim a operação de rotação é realizada para cada *pixel* da imagem.

Para corrigir o problema de esparsidade dos pixels realizamos o mapeamento inverso dos pixels de dentro da imagem. Para realizar esse processo precisamos calcular a rotação inversa, assim é possível identificar o ponto referente na imagem original. Para realizar essa operação realizamos os mesmos passos para realizar a rotação, apenas passamos o angulo inverso e o centro da nova imagem. Assim no laço de correção realizamos a atribuição dos valores da imagem original para a imagem rotacionada, o *if* garante que só é mapeado os *pixels* que existem na imagem original. A Figura 3 ilustra a rotação em 45° e -45°.

2.3 *Coloração Artificial de Imagens*

A coloração Artificial de imagens converte uma imagem de tons de cinza em uma imagem colorida, mapeando os tons de cinza da imagem em uma cor diferente. Essa técnica é útil para realçar imagens com tons de cinza muito parecidos que dificultando a visualização de detalhes das imagens Pedrini [2018]. A seguir são apresentados o código implementado da técnica e um exemplo de sua aplicação.

Figura 3: Exemplo da aplicação da técnica de rotação



(a) Imagem rotacionada 45° (b) Imagem rotacionada -45°

```
# -*- coding: utf-8 -*-
import numpy as np
import cv2
import colorsys
import imageio

def apply_palette(img, palette):
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    new = np.zeros((len(img_gray),len(img_gray[0]),3), np.uint8)
    for i in range(len(img_gray)):
        for j in range(len(img_gray[0])):
            new[i,j] = palette[img_gray[i,j],0]
    return new

def create_palette(hue):
    black_saturation      = np.linspace(255, 255, 128)
    black_intensity       = np.linspace(0, 255, 128)

    saturation_white      = np.linspace(255, 0, 128)
    intensity_white       = np.linspace(255, 255, 128)

    palette_hue           = np.linspace(hue, hue, 256)
    palette_saturation    = np.concatenate((black_saturation, saturation_white))
    palette_intensity     = np.concatenate((black_intensity, intensity_white))

    palette_hue           = np.tile(palette_hue.reshape((256,1)), 256)
    palette_saturation    = np.tile(palette_saturation.reshape((256,1)), 256)
    palette_intensity     = np.tile(palette_intensity.reshape((256,1)), 256)

    img = cv2.merge([palette_hue,palette_saturation,palette_intensity])
    img = np.uint8(img)

    return cv2.cvtColor(img, cv2.COLOR_HSV2BGR)
```

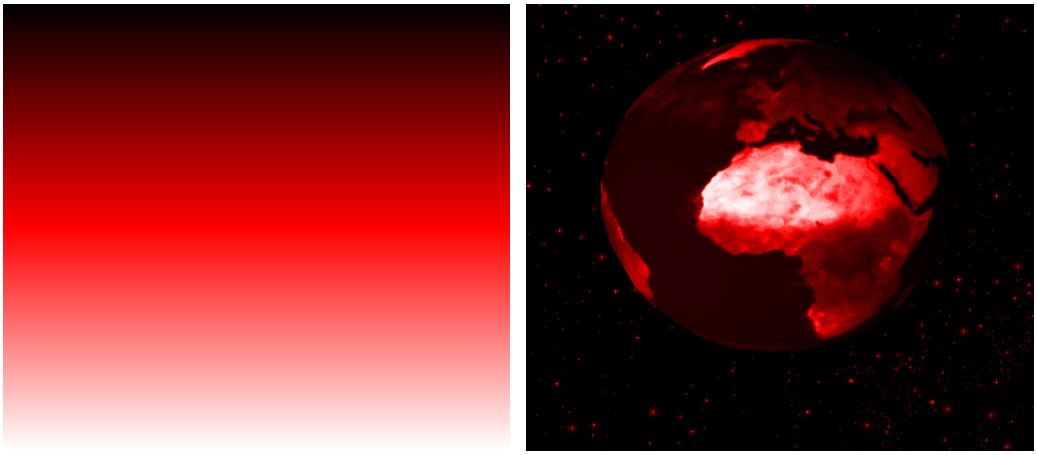
```
def colorization(gif_name, start_hue, end_hue):
    gif = imageio.mimread(gif_name)
    new_gif = []
    colors = np.linspace(start_hue, end_hue, len(gif))
    for i in range(len(colors)):
        color = colors[i]
        frame = gif[i]
        new_gif.append(apply_palette(frame,create_palette(color)))
    gif = imageio.mimwrite('colorization_{}'.format(img_name),new_gif,'gif')

if __name__ == "__main__":
    img_name = 'mundo.gif'
    colorization(img_name,0,180)
```

A partir dessa técnica é possível criar várias imagens com tonalidades de uma mesma cor, sempre variando do preto para a cor e da cor para o branco. Denominamos como *frame* cada imagem de escalas de cores gerada. O número de *frames* é definido de acordo com a operação *np.linspace(start_hue, end_hue, len(gif))*, em que *start_hue* é a primeira cor, *end_hue* é a última cor e *len(gif)* é a quantidade de *frames* que será gerado (*len(gif)* retorna a quantidade de *frames* do *gif*). Essa operação da biblioteca *numpy* gera uma um vetor com valores entre o primeiro valor e o segundo de maneira que todos os valores do vetor sejam equidistantes entre si. Portanto cada valor nesse vetor representa um *hue*. Cada paleta gerada foi feita em duas partes, a saber: a primeira variando do preto para a cor (*hue*) e segunda variando da cor para o branco. Como foi usado o modelo de cores HSI, para gerar a variação entre preto e a cor escolhida foi necessário apenas variar a intensidade, e para gerar a variação entre a cor e o branco foi necessário variar a saturação. Após isso, as duas partes foram concatenadas formando a escala completa que varia do preto ao branco passando pela cor escolhida.

Após a geração das paletas, a imagem escolhida é transformada em tons de cinzas. Dessa maneira cada tom de cinza da imagem é correspondido por uma cor da paleta (já que ambas, imagem e paleta variam de 0 a 255 cores). Essa correspondência é realizada na função *apply_palette(img, palette)*.

É importante ressaltar que a quantidade de paletas geradas corresponde a quantidade de *frames* que o gif possui. Na Figura 4 é apresentado um exemplo de uma paleta cor *hue = 0* (modelo HSI) e ao lado um *frame* do *gif* utilizado com a paleta aplicada.

Figura 4: Exemplo da aplicação da técnica de mudança de escala



(a) Imagem com escala 0.5x1.

(b) Imagem com escala 1x0.5.

3 Conclusão

Neste trabalho foram aplicadas três técnicas muito conhecidas na área de processamento digital de imagens, a *chroma subsampling*, transformações afins (rotação e mudança de escala) e coloração artificial de imagens.

Na técnica *chroma subsampling*, por meio dos experimentos, foi possível notar uma real compressão do tamanho da imagem em questão de bytes sem uma perda de qualidade perceptível a olho nu. Com as transformações afins foi possível modificar imagens rotacionando-as e realizando mudanças de escalas de maneira trivial apenas com alterações de valores nas matrizes de operação. Já na técnica de coloração artificial, gostaríamos de ressaltar a importância dessa técnica em imagens com tons de cinza que não permitem a visualização de detalhes da imagem. Por meio dessa técnicas foi possível notar também a importância das aplicação das técnicas de processamento digital de imagens.

Referências bibliográficas

Eduardo Azevedo, Aura Conci, and Cristina Vasconcelos. *Computação gráfica: Teoria e prática: geração de imagens*, volume 1. Elsevier Brasil, 2018.

Rafael C. Gonzalez and Richard E. Woods. *Digital image processing*. Prentice Hall, Upper Saddle River, N.J., 2008.

Hélio Pedrini. Introdução ao Processamento Digital de Imagem. `http://www.ic.unicamp.br/~helio/disciplinas/MC920/aula_realce.pdf`, 2018. [Online; accessed 16-november-2018].