PROJETO 2 – Amostragem, Interpolação e Quantização

João Vitor Rodrigues Baptista
15/0013329
Universidade de Brasilia
Faculdade Gama
Brasilia, Brasil
jvrbaptista@live.com

***Abstract*— O presente relário esta exposto as atividades teoricas e computacionais da materia de processamento digital de imagens da faculdade Gama.**

Keywords—Imagens, Computational, Interpolação, niveis.

I. INTRODUÇÃO

Interpolação é o processo de apartir de informaçoes conhecidas inferir o comportamento de dados não conhecidos e relevantes. No caso da imagem seria criar novos pixels através dos dados dos pixels existentes.

Tais tecinas de interpolação pode ser vastamente utilizada para aimentar a imagem baseada em imagens que já existem. No presente projeto é apresentado duas tecnicas, uma simples e uma baseada na proximidade de pixel e outra baseada nos quatro vizinhos. Será apresentado as imagens interpolada nas mesmas condições e ambiente.

A quantização de pixel é utlizada para visualizar mudanças de tons que são muito suaves. No presente projeto é mostrado a tecnica baseada no K-mean mudando o centroide para verificar os diferentes comportamentos.

II. EXPERIMENTO

No presente relatório esta exposto a teoria e a simulações computacionais referentes a resolução espacial de imagens, metodologias de interpolação para redimensionamento de imagens e a quantização em niveis de cinza.

Todas as simulações computacionais foram desenvolvidas em python 3 utilizando diversas bibliotecas de manipulação e plote de imagens e o editor Spyder. Foi desenvolvidas funções de interpolação e quantização de niveis de cinza.

III. RESULTADOS

Os resultados obtidos foram conforme o esperado apresentado. Sera apresentado nas seções apropriadas com uma breve discurção da teoria utilizada para o desenlvimento da metodogia e as particularidades de implementação computacional.

A. Resolução espacial de imagens

Primeiramente, foi selecionado uma imagem .jpeg qualquer de dimensões 512x512 pixel. Como as imagens são interpretadas como tensores de, nesse caso, 512x512x3, ou seja, três matrizes de 512x512. Utilizando a conversão de RGB para Grayscale do opencv podemos ter apenas uma matriz 512x512 com niveis de cinza de 0 a 255, no opencv2 deve ser normalizado de 0 até 1.

Como mostrado na fig. 1. A imagem tem apenas um canal de cor com dimensões 512x512 representada em niveis de cinza de 0 até 255.

Figura 1 - Imagem escolhida processada para as dimenções e canais de interesse.de 512 x 512

Em seguida, foi feito uma função que faz a sub-amostragem da imagem, que retorna uma lista de matrizes com as imagens reduzidas pela metade, um quarto, um 8 e um desseseis avos da imagem original, respectivamente 256x256, 128x128, 64x64 e 32x32. A sub-amostragem foi feita de forma bem simples, apenas descartado os valores pares da imagem de referência usando funções de manipulação da biblioteca numpy.

A função recebe a imagem de referencia e o numero de interações que o usuario deseja, assim retorna uma lista das imagens sub-amostrados. As imagens a seguir são as saidas da lista restutantes da função.

Interpolação são processos que são aplicadas para zoom, encolhimento, rotação e correção geometrica de imagens digitais[1]. Interpolação são processos que usa dados conhecidos para estimar dados em locações não conhecidas.

Figura 2 - Imagens escolhidas processada para as dimenções e canais de interesse.de 256 x 256 até 32 x 32

Interpolação são processos que são aplicadas para zoom, encolhimento, rotação e correção geometrica de imagens digitais[1]. Interpolação são processos que usa dados conhecidos para estimar dados em locações não conhecidas.

B. Nearest neighbor interpolation

Uma das abordagens mais simples é atribuir ao lugar do pixel que esta faltando a intensidade do pixel mais mais proximo, ou seja, do pixel vizinho. Essa tecnica é simples mas tem tendencia de produzir artefatos indesejados, como distorções e bordas retas.

Os resultados do experimento esta mostrado a seguir. Foi feito uma função que retora uma a imagem 512 x 512 independente do tamanho da imagem de entrada. O experimento foi feito a interpolação das imagens de dimensão 256 x 256, 128 x 128, 64x 64 e 32 x 32.

As figuras a seguir mostram a imagens interpoladas para metodologia.

.

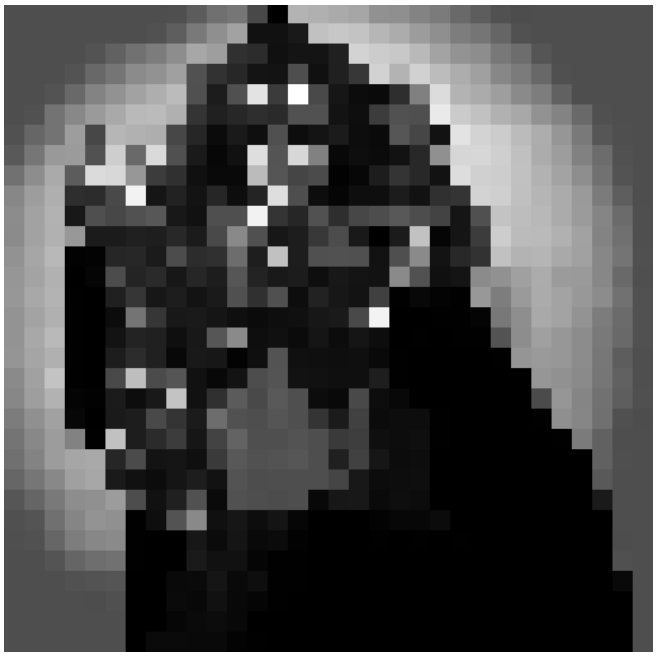

Figura 2 - Imagem interpolada apartir da imagem de 32 x 32 com a tecnica de Nearest Neighbor

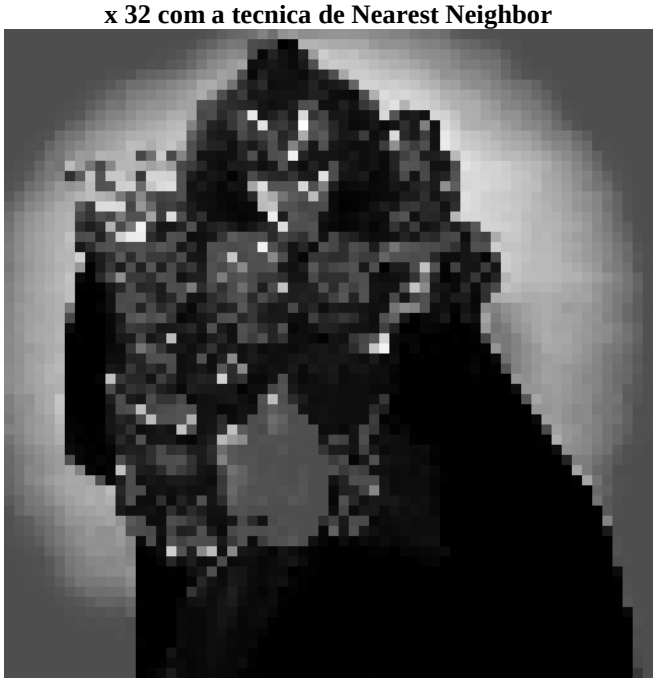

Figura 3 - Imagem interpolada apartir da imagem de 64 x 64 com a teom a tecnica de Nearest Neighbor

Figura 4 - Imagem interpolada apartir da imagem de 128 x 128 com a tecnica de Nearest Neighbor

Figura 5 Imagem interpolada apartir da imagem de 256 x 256 com a tecnica de Nearest Neighbor

C. Interpolação bilinear

Uma das abordagens mais simples é atribuir ao lugar do pixel que esta faltando a intensidade do pixel mais mais proximo, ou seja, do pixel vizinho. Essa tecnica é simples mas tem tendencia de produzir artefatos indesejados, como distorções e bordas retas.

Os resultados do experimento esta mostrado a seguir. Foi feito uma função que retora uma a imagem 512 x 512 independente do tamanho da imagem de entrada. O experimento foi feito a interpolação das imagens de dimensão 256 x 256, 128 x 128, 64x 64 e 32 x 32.

As figuras a seguir mostram a imagens interpoladas para metodologia.

Figura 6 - Imagem interpolada apartir da imagem de 32x 32com a tecnica de Bilinear

Figura 7 - Imagem interpolada apartir da imagem de 64 x 64 com a tecnica de Bilinear

Figura 8 - Imagem interpolada apartir da imagem de 128 x 128 com a tecnica de Bilinear

Figura 9 - Imagem interpolada apartir da imagem de 256 x 256 com a tecnica de Bilinear

Na implentação foi utilizado a equação 1

$$v(x, y) = ax + by + cxy + d \qquad 1$$

Onde a, b,c e d são os vizinhos do pixel de referencia e x e y são as coordenadas do pixel.

Fica evidente que a tecnica biliear alcança resultados melhores comparada com a Nearest neighbor com uma pequena adicição de com de complexidade computacional

D. Quantização em níveis de cinza

Foi feita um agrupamento usando o algoritimo de K-means implementada usando a biblioteca scipy.

Primeira imagem representa a imagem agrupada usando 8 centroides, a segunda imagem usando 4 centroides, terceira imagem usando 2 centroides e a ultima 1.

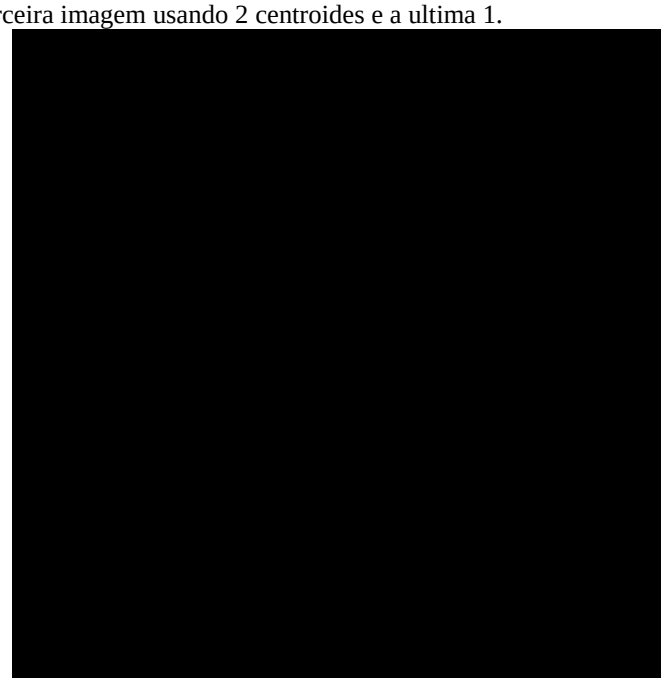

Figura 10 - Figura da imagem agrupada com apenas 1 cecentroide

Figura 11 - Imagem agrupada usando apenas 2 centroides.

Figura 12 - Imagem agrupada usando 4 centroides no algoritmo

Figura 13 - Imagem agrupada usando 8 centroides

DISCURSÃO

É possivel notar as limitações de ambas as metodologias quanto as imagens menores, contudo é possivel observar que que a interpolação bilinear é superior em alguns aspectos pois suaviza a imagem nas bordas, ou seja, supera as mudanças drasticas de cor feitas pela do vizinho mais proximo.

Contudo, ambas as metodologias tem uma boa peformace para as imagens de 256 x 256. Fazendo uma comparação com todas as imagens é possivel notar que a bilinar é superior, pois é mais suave, mesmo com uma pequena adição de complexidade

Como o esperado fazendo o agrupamento dos niveis de cinza em 8, 4, 2 e 1 é possivel observar na partes mais suaves de fundo da imagem mudanças dastricas que quanto menos niveis de pixel de representação mais nitida é a mudaça.

REFERENCIAS

[1] Gonzalez, Rafael C. Processamento digital de imagens / Rafael C. Gonzalez e Richard C. Woods ; revisão técnica : Marcelo Vieira e Mauricio Escarpinati ; [tradução Cristina Yamagami e Leonardo Piamonte]. -- 3. ed. -- São Paulo : Pearson Prentice Hall, 2010..

CODIGO FONTE

```
# -*- coding: utf-8 -*-
"""
Created on Sat Sep  7 09:02:56 2019

@author: helpthx
"""

import numpy as np
import cv2
import matplotlib.pyplot as plt
from scipy import ndimage
from pylab import
imread,imshow,figure,show,subplot
from numpy import reshape,uint8,flipud
from sklearn.cluster import MiniBatchKMeans
print('Versão da OpenCV: ', cv2.__version__,
end='\n\n')

# Importando a imagem usada
img =
cv2.imread('/home/helpthx/Documents/PROJETO_2
/kiss512x512.jpg', cv2.IMREAD_GRAYSCALE)
plt.gray()
plt.imshow(img)
plt.show()
plt.imsave('/home/helpthx/Documents/PROJETO_2
/base512x512.png', img)

img.shape

# Função para fazer o sub sapling da imagem
def simple_subsampling(img, n):
  lista_imagens = []
  for i in range(n+1):
    img = img[1::2, 1::2]
    lista_imagens.append(img)
```

```
  return lista_imagens

# Chamando a função
lista_imagens = simple_subsampling(img, 4)

lista_imagens[0].shape
# (256, 256)

lista_imagens[1].shape
# (128, 128)

lista_imagens[2].shape
# (64, 64)

lista_imagens[3].shape
# (32, 32)

lista_imagens[4].shape
# (16, 16)


# Salvando imagens
plt.imshow(lista_imagens[0])
plt.show()
plt.imsave('/home/helpthx/Documents/PROJETO_2
/sub_samplu256x.png',
           lista_imagens[0])


plt.imshow(lista_imagens[1])
plt.show()
plt.imsave('/home/helpthx/Documents/PROJETO_2
/sub_samplu128x.png',
           lista_imagens[1])


plt.imshow(lista_imagens[2])
plt.show()
plt.imsave('/home/helpthx/Documents/PROJETO_2
/sub_samplu64x.png',
           lista_imagens[2])


plt.imshow(lista_imagens[3])
plt.show()
plt.imsave('/home/helpthx/Documents/PROJETO_2
/sub_samplu32x.png',
           lista_imagens[3])

# Interpolação nn
def nn_interpolate(A, new_size):
    """Vectorized Nearest Neighbor
Interpolation"""

    old_size = A.shape
    row_ratio, col_ratio =
np.array(new_size)/np.array(old_size)

    # row wise interpolation
    row_idx = (np.ceil(range(1, 1 +
int(old_size[0]*row_ratio))/row_ratio) -
1).astype(int)

    # column wise interpolation
    col_idx = (np.ceil(range(1, 1 +
int(old_size[1]*col_ratio))/col_ratio) -
1).astype(int)

    final_matrix = A[:, row_idx][col_idx, :]

    return final_matrix

# Chamando a função
nn_iterpolation256 =
nn_interpolate(lista_imagens[0], 512)
nn_iterpolation128 =
nn_interpolate(lista_imagens[1], 512)
nn_iterpolation64 =
nn_interpolate(lista_imagens[2], 512)
nn_iterpolation32 =
nn_interpolate(lista_imagens[3], 512)

# Salvando arquivos
plt.imshow(nn_iterpolation256)
plt.show()
plt.imsave('/home/helpthx/Documents/PROJETO_2
/nn_inteporlation256x.png',
           nn_iterpolation256)


plt.imshow(nn_iterpolation128)
plt.show()
plt.imsave('/home/helpthx/Documents/PROJETO_2
```

```
/nn_inteporlation128x.png',
            nn_iterpolation128)


plt.imshow(nn_iterpolation64)
plt.show()
plt.imsave('/home/helpthx/Documents/PROJETO_2
/nn_inteporlation64x.png',
            nn_iterpolation64)


plt.imshow(nn_iterpolation32)
plt.show()
plt.imsave('/home/helpthx/Documents/PROJETO_2
/nn_inteporlation32x.png',
            nn_iterpolation32)

# Interpolação Biliniar
# Bilinear interpolation
def bilinear_interpolate(image):
  image =
cv2.cvtColor(image,cv2.COLOR_GRAY2RGB)
  (h, w, channels) = image.shape

  h2 = 512
  w2 = 512
  temp = np.zeros((h2, w2, 3), np.uint8)
  x_ratio = float((w - 1)) / w2;
  y_ratio = float((h - 1)) / h2;
  for i in range(1, h2 - 1):
    for j in range(1 ,w2 - 1):
      x = int(x_ratio * j)
      y = int(y_ratio * i)
      x_diff = (x_ratio * j) - x
      y_diff = (y_ratio * i) - y
      a = image[x, y] & 0xFF
      b = image[x + 1, y] & 0xFF
      c = image[x, y + 1] & 0xFF
      d = image[x + 1, y + 1] & 0xFF
      blue = a[0] * (1 - x_diff) * (1 -
y_diff) + b[0] * (x_diff) * (1-y_diff) + c[0]
* y_diff * (1 - x_diff)   + d[0] * (x_diff *
y_diff)
      green = a[1] * (1 - x_diff) * (1 -
y_diff) + b[1] * (x_diff) * (1-y_diff) + c[1]
* y_diff * (1 - x_diff)   + d[1] * (x_diff *
y_diff)
      red = a[2] * (1 - x_diff) * (1 -
y_diff) + b[2] * (x_diff) * (1-y_diff) + c[2]
* y_diff * (1 - x_diff)   + d[2] * (x_diff *
y_diff)
      temp[j, i] = (blue, green, red)

  return cv2.cvtColor(temp,
cv2.COLOR_BGR2GRAY)

# Chamando as funções
bl_iterpolation256 =
bilinear_interpolate(lista_imagens[0])
bl_iterpolation128 =
bilinear_interpolate(lista_imagens[1])
bl_iterpolation64 =
bilinear_interpolate(lista_imagens[2])
bl_iterpolation32 =
bilinear_interpolate(lista_imagens[3])

# Salvando arquivos
plt.imshow(bl_iterpolation256)
plt.show()
plt.imsave('/home/helpthx/Documents/PROJETO_2
/bl_inteporlation256x.png',
            bl_iterpolation256)


plt.imshow(bl_iterpolation128)
plt.show()
plt.imsave('/home/helpthx/Documents/PROJETO_2
/bl_inteporlation128x.png',
            bl_iterpolation128)


plt.imshow(bl_iterpolation64)
plt.show()
plt.imsave('/home/helpthx/Documents/PROJETO_2
/bl_inteporlation64x.png',
            bl_iterpolation64)


plt.imshow(bl_iterpolation32)
plt.show()
plt.imsave('/home/helpthx/Documents/PROJETO_2
/bl_inteporlation32x.png',
            bl_iterpolation32)
```

```
# Qubatização de niveis de cinza
image =
cv2.imread('/home/helpthx/Documents/PROJETO_2
/kiss512x512.jpg',
                    cv2.IMREAD_GRAYSCALE)
print(image.shape)

def quantizador_kmeans(image, n):
    # Extract width & height of image
    (HEIGHT, WIDTH) = image.shape[:2]

    # Convert image to L, A, B color space
    # image = cv2.cvtColor(image,
cv2.COLOR_BGR2LAB)

    # Reshape the image to a feature vector
    image = image.reshape((image.shape[0] *
image.shape[1], 1))

    # Apply MiniBatchKMeans and then create
the quantized image based on the predictions
    clt = MiniBatchKMeans(n_clusters = n)
    labels = clt.fit_predict(image)
    print(labels)
    quant =
clt.cluster_centers_.astype("uint8")[labels]

    # reshape the feature vectors to images
    quant = quant.reshape((HEIGHT, WIDTH))
    image = image.reshape((HEIGHT, WIDTH))

    return quant, image

quant_8, image_8 = quantizador_kmeans(image,
8)
quant_4, image_4 = quantizador_kmeans(image,
4)
quant_2, image_2 = quantizador_kmeans(image,
2)
quant_1, image_1 = quantizador_kmeans(image,
1)

# Salvando arquivos
plt.gray()

plt.imshow(quant_8)
plt.show()
plt.imsave('/home/helpthx/Documents/PROJETO_2
/quant_8.png',
           quant_8)


plt.imshow(quant_4)
plt.show()
plt.imsave('/home/helpthx/Documents/PROJETO_2
/quant_4.png',
           quant_4)


plt.imshow(quant_2)
plt.show()
plt.imsave('/home/helpthx/Documents/PROJETO_2
/quant_2.png',
           quant_2)


plt.imshow(quant_1)
plt.show()
plt.imsave('/home/helpthx/Documents/PROJETO_2
/quant_1.png',
           quant_1)
```