
Relatório - Atividade 4

UNIVERSIDADE ESTADUAL DE CAMPINAS

INSTITUTO DE COMPUTAÇÃO

INTRODUÇÃO AO PROCESSAMENTO DIGITAL DE IMAGENS

JUNHO DE 2018

FELIPPE TRIGUEIRO ANGELO
210479

1 Descrição geral da atividade 4

A atividade 4 da disciplina Introdução ao Processamento Digital de Imagens teve como objetivo a implementação de quatro métodos de interpolação de pixels para a correta representação de imagens rotacionadas e escalonadas. Os métodos implementados são: (i) Interpolação pelo Vizinho mais próximo, (ii) Interpolação Bilinear, (iii) Interpolação Bicúbica, (iv) Interpolação por polinômios de Lagrange.

Na atividade em questão, o usuário fornece, além da imagem de entrada, a transformação, juntamente com o método de interpolação dentre as quatro opções disponíveis que ele deseja utilizar.

2 Executando o Programa

Para executar o programa em questão, entre no terminal do linux, navegue até a pasta onde os códigos estão localizados e digite o seguinte comando:

```
sudo python3 transform.py [transformação] [parametro(s)] [método-de-interpolação] [imagem-de-entrada] [imagem-de-saída]
```

```
Ex: sudo python3 transform.py -a -30 ne baboon.png baboon_rotated.png
```

2.1 Dados de Entrada

Para a correta execução do programa é necessário que alguns cuidados sejam tomados com os dados de entrada:

- As imagens de entrada, devem estar na mesma pasta em que está contido o código-fonte, devem possuir o formato .png e devem ser monocromáticas;
- O formato da imagem de saída também deve ser .png.
- Para a seleção da transformação, deve-se utilizar apenas uma das seguintes entradas:
 - -a ângulo: Operação de Rotação juntamente com o ângulo medido em graus;
 - -d largura altura: Operação de escala com entrada da Largura e Altura da nova imagem;
 - -e fator de escala: Operação de escala com entrada do fator de escala. É utilizado apenas um fator de escala para ambos os eixos.
- Para a seleção do método de interpolação, deve-se utilizar apenas uma das seguintes entradas:
 - ne: Método de interpolação pelo vizinho mais próximo;
 - bl: Método de interpolação Bilinear;
 - bc: Método de interpolação Bicúbica;
 - la: Método de interpolação pelo polinômio de Lagrange.

2.2 Dados de Saída

Após o código ser executado, é exibido na tela a imagem resultante das operações inseridas na entrada, juntamente com a imagem original para se estabelecer um comparativo entre as duas. A imagem resultante também é armazanada na pasta Output_Images localizada na pasta do código fonte.

3 Descrevendo o Código

3.1 Algoritmos e Estruturas de Dados utilizadas

3.1.1 Interpolação pelo Vizinho mais Próximo

O primeiro método de interpolação que foi implementado, foi o método de interpolação pelo vizinho mais próximo. O mesmo, consiste em atribuir ao valor de um pixel da imagem reamostrada, o

valor do pixel mais próximo na imagem original. Para realizar tal operação foi escrita a função `nearest_neighbor()` presente no arquivo `interpolation_methods.py`. A função recebe como entrada a imagem original que se deseja realizar a operação, os valores de escala nos eixos `sx` e `sy`, assim como o ângulo de rotação.

Quando o usuário passa como entrada as novas dimensões da imagem, os valores de `sx` e `sy` são calculados e passados para a função. Neste caso o parâmetro de ângulo é colocado como `None`. Já quando o usuário realiza uma operação de escala, passando como parâmetro o valor que a imagem deve ser escalonada, tanto `sx` quanto `sy` recebem o valor de escala que foi passado como entrada. Neste caso, o parâmetro de ângulo também é colocado como `None`. Por último, quando o usuário deseja realizar uma rotação, os parâmetros `sx` e `sy` são colocados como `None` e o ângulo é passado para a função. Uma descrição do cabeçalho da função é feita abaixo. É importante citar que a função não modifica a imagem de entrada, mas sim retorna uma nova imagem com as propriedades de transformação e interpolação desejadas.

```
1 outImage = nearest_neighbor (original_Image, sx, sy, angle)
```

Para o cálculo da interpolação pelo método do vizinho mais próximo, inicialmente foi calculado a transformação inversa para cada uma das operações. Estas operações inversas podem ser vistas nas equações abaixo:

$$x = \text{coluna}/sx \quad (1)$$

$$y = \text{linha}/sy \quad (2)$$

$$x = c * \cos\theta - l * \sin\theta \quad (3)$$

$$y = c * \sin\theta + l * \cos\theta \quad (4)$$

As equações 1 e 2 são utilizadas quando o usuário deseja realizar a transformação de escala e as restantes são utilizadas quando deseja-se realizar a transformação de rotação. Estas operações são realizadas em todos os métodos de interpolação, com o intuito de obter a posição correspondente do pixel de saída, na imagem original.

Para a realização do método de interpolação em questão é utilizada a função `round()` que aproxima o número passado como argumento, para o inteiro mais próximo. Assim os pixels da imagem são calculados como segue:

$$f(x', y') = f(\text{round}(x), \text{round}(y)) \quad (5)$$

As figuras 1, 2 e 3 exemplificam o uso desse método de interpolação para as três transformações em questão.

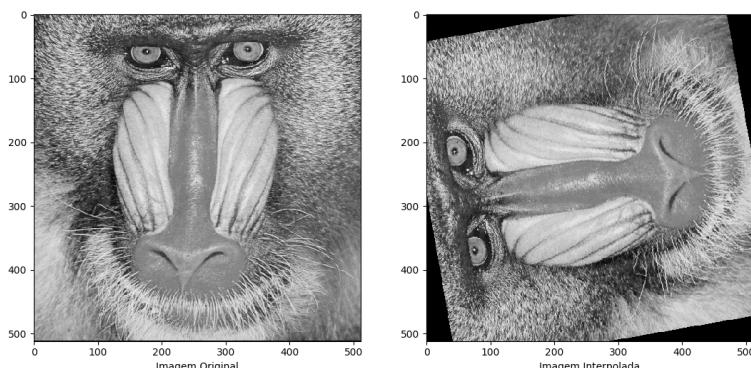


Figure 1: Interpolação de uma transformação de rotação usando o método de interpolação pelo vizinho mais próximo.

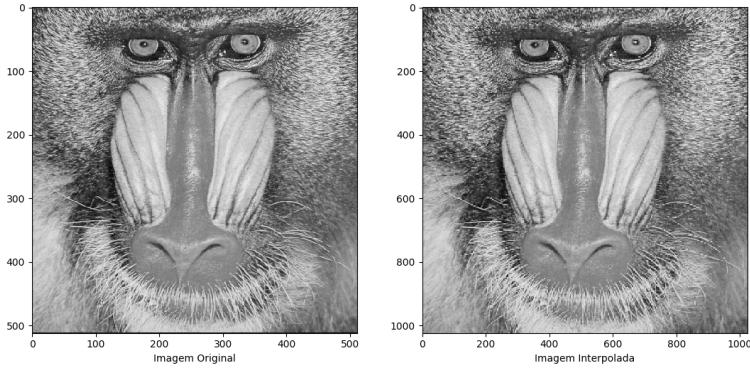


Figure 2: Interpolação de uma transformação de escala usando o método de interpolação pelo vizinho mais próximo.

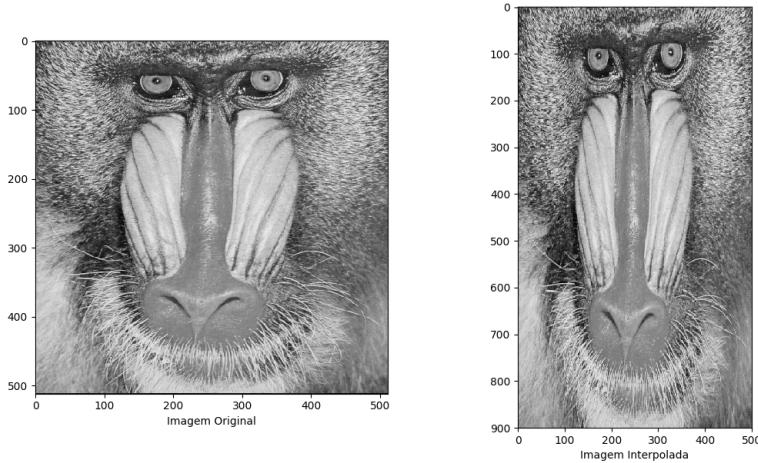


Figure 3: Interpolação de uma transformação de dimensão usando o método de interpolação pelo vizinho mais próximo.

3.1.2 Interpolação Bilinear

O segundo método de interpolação que foi implementado foi a interpolação bilinear. Esse método consiste em uma média ponderada de distância dos quatro pixels vizinhos mais próximos para determinar a intensidade de cada pixel (x', y') na imagem transformada. Matematicamente essa operação é definida por meio da equação 6.

$$f(x', y') = (1-dx)(1-dy)f(x, y) + dx(1-dy)f(x+1, y) + (1-dx)dyf(x, y+1) + dxdyf(x+1, y+1) \quad (6)$$

Para a realização desse método são utilizadas as mesmas transformações inversas descritas no método anterior. Assim, obtidas essas transformações, calcula-se a média bilinear e o respectivo pixel. É importante citar que o valor de $f(x', y')$ também é passado para a função `(round())`. Abaixo é descrito o cabeçalho da função `bilinear()`, presente no arquivo `interpolation_methods.py`, que realiza o método de interpolação bilinear.

```
1 outImage = bilinear(original_Image, sx, sy, angle)
```

As figuras 4, 5 e 6 exemplificam o uso desse método de interpolação para as três transformações em questão.

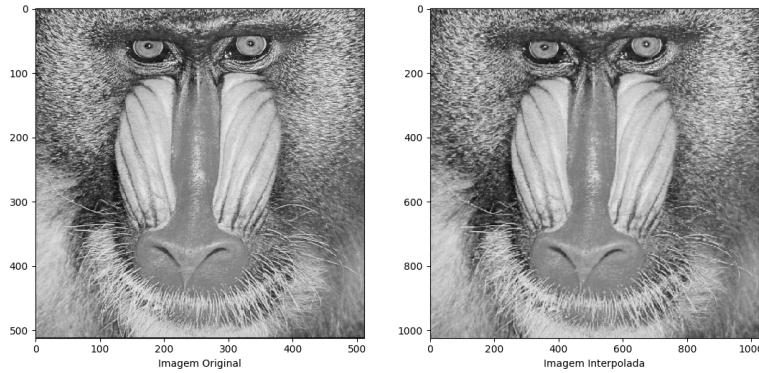


Figure 4: Interpolação de uma transformação de escala usando o método de interpolação bilinear.

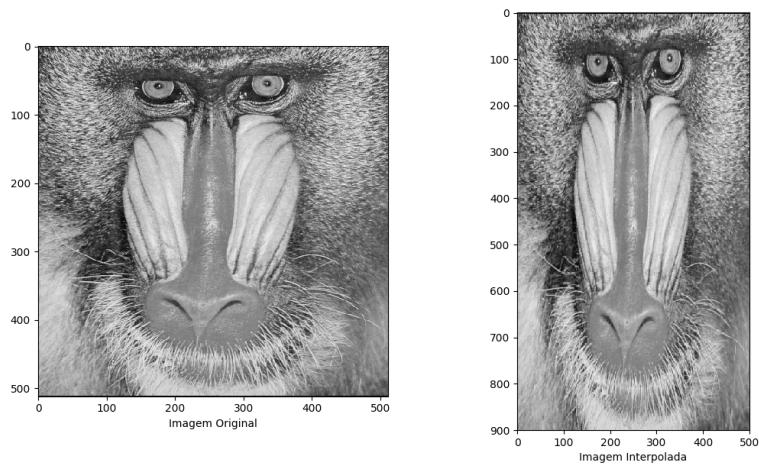


Figure 5: Interpolação de uma transformação de dimensão usando o método de interpolação bilinear.

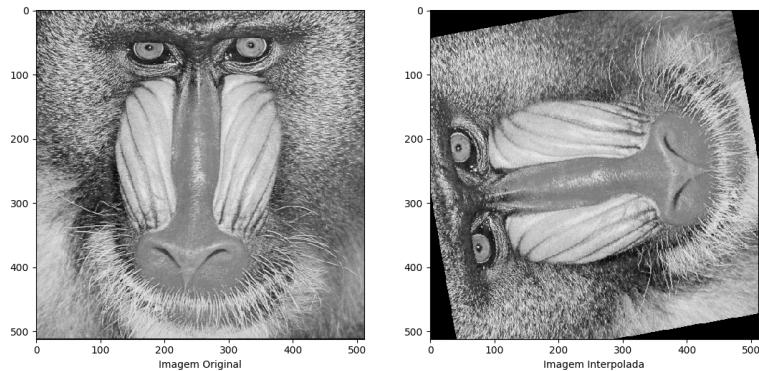


Figure 6: Interpolação de uma transformação de rotação usando o método de interpolação bilinear.

3.1.3 Interpolação Bicúbica

O terceiro método de interpolação utilizado foi o método de interpolação bicúbica. Este método utiliza uma vizinhança de 4×4 pontos ao redor do ponto em questão para calcular seu valor de intensidade. Uma função comum para calcular as intensidades do pixel na imagem interpolada é a função B-spline cúbica, definida por meio das equações 7, 8 e 9.

$$f(x', y') = \sum_{m=-1}^2 \sum_{n=-1}^2 f(x + m, y + n) R(m - dx) R(dy - n) \quad (7)$$

$$R(s) = \frac{1}{6}[P(s+2)^3 - 4P(s+1)^3 + 6P(s)^3 - 4P(s-1)^3] \quad (8)$$

$$P(t) = t; t > 0 \mid P(t) = 0; \text{caso contrário} \quad (9)$$

Para a realização desse método são utilizadas as mesmas transformações inversas descritas no método do vizinho mais próximo. Assim, obtidas essas transformações, calcula-se o valor de $f(x', y')$ por meio da equação 7. É importante citar que o valor de $f(x', y')$ também é passado para a função `(round())`. Abaixo é descrito o cabeçalho da função `bicubic()`, presente no arquivo `interpolation_methods.py`, que realiza o método de interpolação bilinear.

```
1 outImage = bicubic(original_Image, sx, sy, angle)
```

As figuras 7, 8 e 9 exemplificam o uso desse método de interpolação para as três transformações em questão.

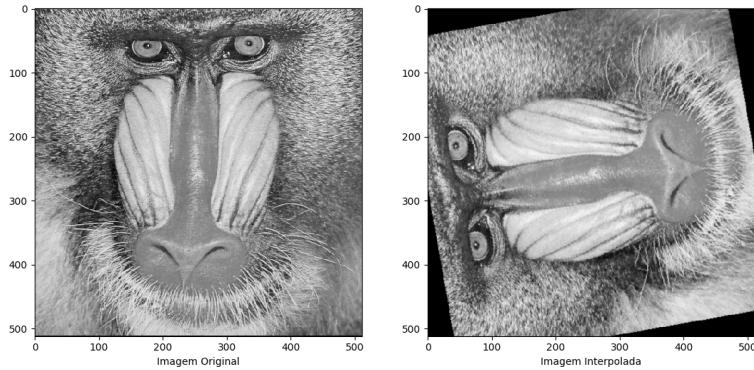


Figure 7: Rotação utilizando a interpolação bicúbica.

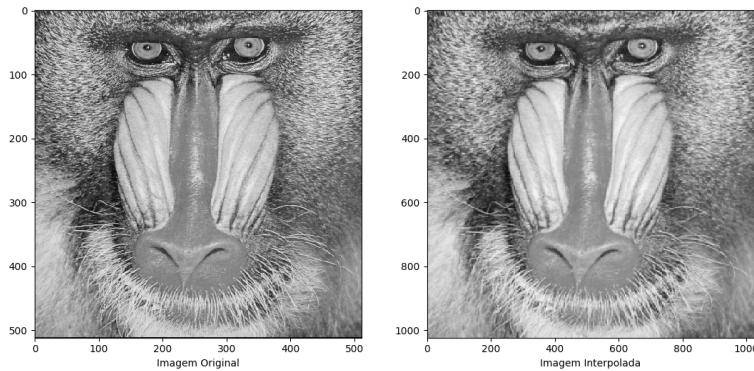


Figure 8: Escala utilizando a interpolação bicúbica.

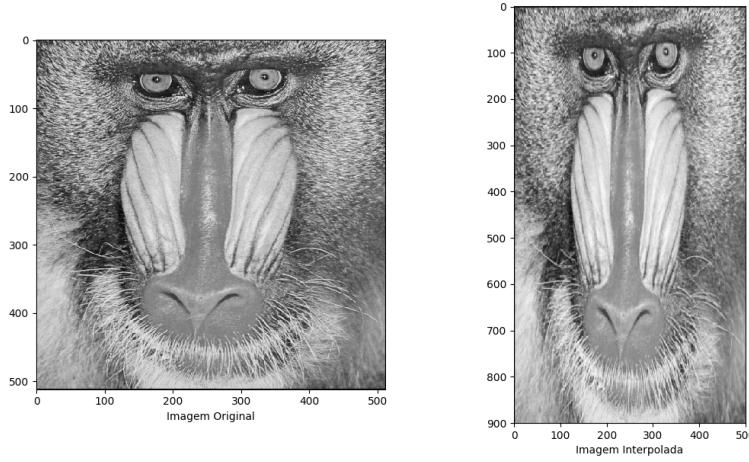


Figure 9: Mudança de dimensões utilizando a interpolação bicúbica.

3.2 Interpolação utilizando Polinômios de Lagrange

O último método de interpolação que foi implementado foi o método de interpolação por Polinômios de Lagrange. Este método também se baseia em vizinhanças 4×4 para o cálculo da interpolação. O cálculo da interpolação é feito por meio das equações 10 e 11.

$$f(x', y') = \frac{-dy(dy - 1)(dy - 2)L(1)}{6} + \frac{(dy + 1)(dy - 1)(dy - 2)L(2)}{2} + \frac{-dy(dy + 1)(dy - 2)L(3)}{2} + \frac{dy(dy + 1)(dy - 1)L(4)}{6} \quad (10)$$

$$L(n) = \frac{-dx(dx - 1)(dx - 2)f(x - 1, y + n - 2)}{6} + \frac{(dx + 1)(dx - 1)(dx - 2)f(x, y + n + 2)}{2} + \frac{-dx(dx + 1)(dx - 2)f(x + 1, y + n - 2)}{2} + \frac{dx(dx + 1)(dx - 1)f(x + 2, y + n - 2)}{6} \quad (11)$$

Para a realização desse método são utilizadas as mesmas transformações inversas descritas no método do vizinho mais próximo. Assim, obtidas essas transformações, calcula-se o valor de $f(x', y')$ por meio da equação 10. É importante citar que o valor de $f(x', y')$ também é passado para a função `(round())`. Abaixo é descrito o cabeçalho da função `lagrange()`, presente no arquivo `interpolation_methods.py`, que realiza o método de interpolação bilinear.

```
1 outImage = lagrange(original_Image, sx, sy, angle)
```

As figuras 10, 11 e 12 exemplificam o uso desse método de interpolação para as três transformações em questão.

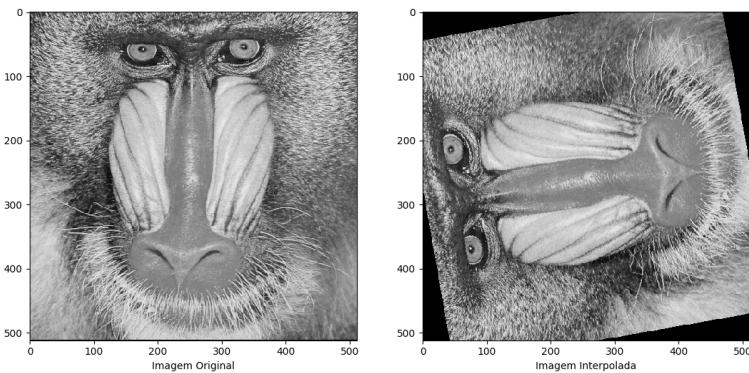


Figure 10: Rotação utilizando a interpolação por polinômio de Lagrange.

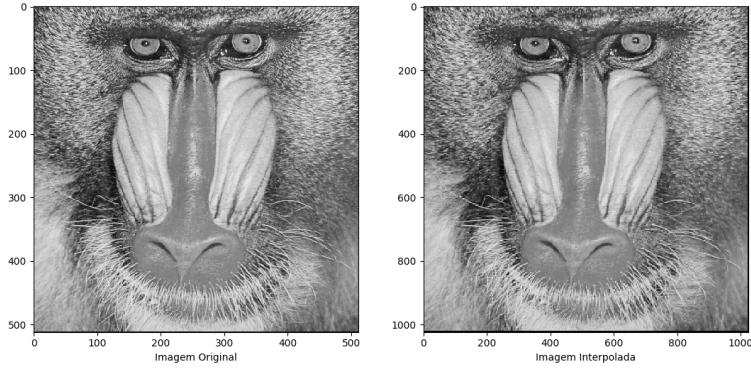


Figure 11: Escala utilizando a interpolação por polinômio de Lagrange.

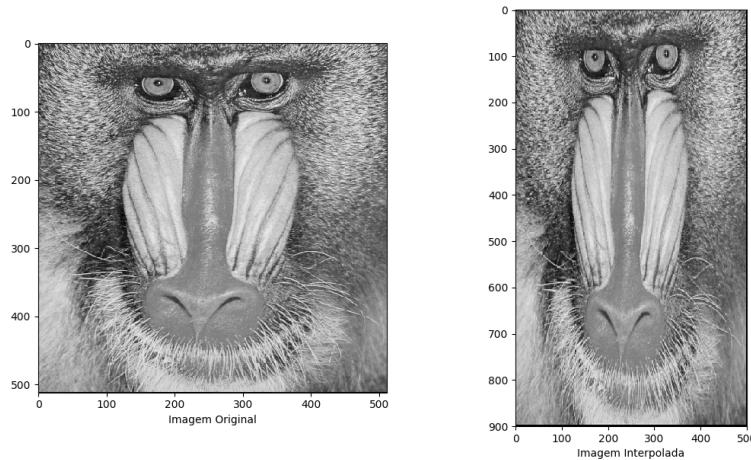


Figure 12: Mudança de dimensões utilizando a interpolação por polinômio de Lagrange.

3.3 Estruturas de dados utilizadas

No código, foram utilizadas as estruturas de dados array e tupla no armazenamento das informações desejadas. No caso das imagens foram utilizados os arrays providos pela biblioteca *numpy*, gerando arrays da classe *ndarray*. Como estamos utilizando imagens de entrada no formato monocromáticas, elas são representadas como arrays do tipo MxN.

As tuplas foram utilizadas no armazenamento das dimensões das imagens.

4 Testes adotados e Limitações do programa

Foram feitos testes com as imagens contidas no endereço da página da disciplina. Portanto, foram testadas imagens quadradas e monocromáticas de dimensão 512x512. Também foram feitos testes com imagens coloridas e retangulares, entretanto nem todos os testes obtiveram bons resultados devido a conversão RGB para monocromático, assim convencionou-se utilizar apenas imagens monocromáticas.

A partir dos resultados dos testes é possível afirmar que os quatro métodos resultam em imagens com características fiéis as da imagem original. Entretanto não foi utilizado nenhum método para mensurar esse grau de similaridade.

O método de interpolação pelo vizinho mais próximo demonstrou ser o de mais simples implementação e que obtém os resultados mais rapidamente. Já o método bicúbico demonstrou ser um método um pouco mais demorado, em decorrência do cálculo do somatório duplo.

5 Referências Bibliográficas

1. **The Python Standard Library.** Disponível em: <https://docs.python.org/3/library/index.html>
2. **The Matplotlib API Pyplot.** Disponível em: https://matplotlib.org/api/pyplot_summary.html