# Exp. 01 - Lab. Introdução ao Processamento Digital de Imagens e Visão Computacional

June 30, 2021


### 0.1 Centro de Engenharia Elétrica e Informática

### 0.2 Departamento de Engenharia Elétrica

### 0.3 Disciplina: Int. ao Processamento de Imagem Digital e Visão Computacional

### 0.4 Professora: Luciana Veloso

### 0.5 Aluno(a): George Camboim - 119210121


## 1 Experimento 01: Processamento Digital de Imagens

[1]:
```
import cv2                              # OpenCV para manipulação de imagens.
import numpy as np                      # Numpy para manipulação de matrizes e␣
 ↪arrays.
import matplotlib.pyplot as plt         # Pyplot para plotagem de gráficos e␣
 ↪imagens.
```

### 1.1 1. Numpy Arrays

Podemos utilizar a biblioteca Numpy para escrever arrays e matrizes de forma semelhante ao Matlab:
* a) A = np.array( [1, 2, 3, 4, 5] )
* b) B = np.array( [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ] )
* c) C = np.array( [ [ [1, 2], [3, 4] ], [ [1, 2], [3, 4] ] ] )

Descrição:
* O código do item **a)** produz um array unidimensional a partir de uma lista * Já o do item **b)** produz um array bidimensional, ou uma matriz, a partir de uma lista de listas * Por fim, o código do item **c)** produz um array tridimensional * Os arrays podem ser exibidos a partir do comando **print**, como por exemplo em **print(A)** * É possível checar as dimensões de cada array printando o atributo **shape**, digitando por exemplo **print(A.shape)**

[2]:
```
A = np.array( [1, 2, 3, 4, 5] )
print("Array A:")
print(A)
print("\nShape de A:", A.shape)
```

```
Array A:
[1 2 3 4 5]

Shape de A: (5,)
```

[3]:
```
B = np.array( [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ] )
print("Array B:")
print(B)
print("\nShape de B:", B.shape)
```

```
Array B:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Shape de B: (3, 3)
```

[4]:
```
C = np.array( [ [ [1, 2], [3, 4] ], [ [5, 6], [7, 8] ] ] )
print("Array C:")
print(C)
print("\nShape de C:", C.shape)
```

```
Array C:
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]

Shape de C: (2, 2, 2)
```

Elementos internos dos arrays podem ser acessados utilizando colchetes, por exemplo:

- A[0] acessa o primeiro elemento de A, nesse caso 1;
- B[0, 2] acessa o terceiro elemento da primeira linha de B, ou seja 3.
- C[1, 0, 1] acessa o segundo elemento da primeira linha da segunda matriz de C, ou seja 6;

[5]:
```
print("A[0] ==", A[0])
```

```
A[0] == 1
```

[6]:
```
print("B[0,2] ==", B[0,2])
```

```
B[0,2] == 3
```

[7]:
```
print("C[1, 0, 1] ==", C[1, 0, 1])
```

```
C[1, 0, 1] == 6
```

Também é possível acessar fatias do array utilizando o operador **:** como pode ser visto a seguir:

- A[0:4] retorna um array formado pelos quatro primeiros elementos de A;
- B[2, 0:2] retorna um array unidimensional formado pelos dois primeiros elementos da terceira linha de B;
- C[0, :, :] retorna um array bidimensional formado por todas as linhas e colunas da primeira matriz de C;
- Observe que a contagem dos índices é iniciada em 0;

[8]:
```
print("A[0:4]:\n\n", A[0:4])
```

```
A[0:4]:

 [1 2 3 4]
```

[9]:
```
print("B[2, 0:2]:\n\n", B[2, 0:2])
```

```
B[2, 0:2]:

 [7 8]
```

[10]:
```
print("C[0, :, :]:\n\n", C[0, :, :])
```

```
C[0, :, :]:

 [[1 2]
 [3 4]]
```

## 1.2 Questão 01

**a. Produza as seguintes matrizes:**

```
A1 = A[1:4]

B1 = B[1:2, 0:1]

B2 = B[-1, -2]

C1 = C[-2, -2, -2]

C2 = C[0, 0, 0]

C3 = C[:, 0, 0]
```

[24]:
```
A1 = A[1:4]
print("A1:", A1)
B1 = B[1:2, 0:1]
B2 = B[-1, -2]
print("B1:", B1)
print("B2:", B2)
C1 = C[-2, -2, -2]
C2 = C[0, 0, 0]
C3 = C[:, 0, 0]
print("C1:", C1)
print("C2:", C2)
print("C3:", C3)
```

```
A1: [2 3 4]
B1: [[4]]
B2: 8
C1: 1
C2: 1
C3: [1 5]
```

**b. Qual o significado de passar um índice negativo?**

Um índice negativo representa uma ordem reversa de indexação, iniciando a partir do último elemento com índice -1.

**c. C1 é igual a C2? Se sim,Por que?**

Sim. Porquê C[-2, -2, -2] e C[0, 0, 0] estão indexando o mesmo elemento de formas diferentes.

## 1.3 2. Tipos de Imagem

1. Vamos trabalhar com quatro tipos de imagens:
    a. **Imagens de Intensidades:** São matrizes cujos valores representam intensidades em cada ponto. Elementos de intensidade da classe uint8 terão valores no intervalo [0, 255]. Já elementos da classe uint16 terão valores entre [0, 65535].
    b. **Imagens Binárias:** São um arranjo lógico em forma de matriz cujos valores são booleanos, podendo ser 0s ou 1s;
    c. **Imagens Indexadas:** São imagens cujo valor de cada pixel está associado a uma cor descrita por um mapa de cores (colormap);
    d. **Imagens Coloridas:** São Imagens com múltiplos canais onde os múltiplos valores associados a um determinado pixel descrevem a sua cor. Um exemplo seriam imagens RGB, onde os diferentes canais descrevem a intensidade luminosa das cores vermelho, verde e azul, respectivamente, de uma imagem;

- Note que os tipos de imagem não são excludentes, uma mesma imagem pode estar associada a mais de um dos tipos descritos.

[16]:
```
%matplotlib notebook
# Essa linha torna a figura do matplotlib interativa, permitindo visualizar os␣
 ↪valores de cada pixel com o mouse.
# A figura deixará de ser interativa quando outra célula for executada. É␣
 ↪possível reativá-la ao rodar essa célula novamente.

# Um array do numpy pode ser interpretado como uma imagem de intensidade:
I = np.array( [[0, 50, 100, 150, 200, 250],
               [0, 50, 100, 150, 200, 250],
               [0, 50, 100, 150, 200, 250],
               [0, 50, 100, 150, 200, 250],
               [0, 50, 100, 150, 200, 250] ] ).astype(np.uint8)

# Podemos utilizar o matplotlib para visualizar a imagem através da função␣
 ↪imshow:
fig = plt.figure(figsize=(4, 4))
plt.imshow(I, cmap="gray")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[16]:
```
<matplotlib.image.AxesImage at 0x7f1077c88290>
```

- Observe que a associação das intensidades a cores é arbitrária, podendo ser obtida uma imagem diferente mudando o colormap:

[17]:
```
%matplotlib notebook
fig, axs = plt.subplots(nrows = 2, ncols = 3, figsize=(8, 6))

## Primeira linha de subfiguras
# Primeira Subfigura
axs[0][0].imshow(I)
axs[0][0].set_title("Colormap Padrão", fontsize = 10)

# Segunda Subfigura
axs[0][1].imshow(I, cmap="gray")
axs[0][1].set_title("Colormap em Escala de Cinza", fontsize = 10)

# Terceira Subfigura
axs[0][2].imshow(I, cmap="Purples")
axs[0][2].set_title("Colormap em tons de roxo", fontsize = 10)

## Segunda linha de subfiguras
# Primeira Subfigura
axs[1][0].imshow(I, cmap="Reds")
```

```
axs[1][0].set_title("Colormap em tons de vermelho", fontsize = 10)

# Segunda Subfigura
axs[1][1].imshow(I, cmap="Greens")
axs[1][1].set_title("Colormap em tons de verde", fontsize = 10)

# Terceira Subfigura
axs[1][2].imshow(I, cmap="Blues")
axs[1][2].set_title("Colormap em tons de azul", fontsize = 10)

# Referências de colormaps: https://matplotlib.org/stable/tutorials/colors/
 ↪colormaps.html
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[17]:
```
Text(0.5, 1.0, 'Colormap em tons de azul')
```

Observe que a associação dos valores mínimos e máximos também é arbitrária e pode ser controlada pelos argumentos **vmin** e **vmax**:

[19]:
```
%matplotlib notebook
fig, axs = plt.subplots(nrows = 2, ncols = 3, figsize=(8, 6))

## Primeira linha de subfiguras
# Primeira Subfigura
axs[0][0].imshow(I, cmap="gray")
axs[0][0].set_title("Valores Padrão", fontsize = 10)

# Segunda Subfigura
axs[0][1].imshow(I, vmin = 100, vmax = 200, cmap="gray")
axs[0][1].set_title("vmin = 100, vmax = 200", fontsize = 10)

# Terceira Subfigura
axs[0][2].imshow(I, vmin = 25, vmax = 75, cmap="gray")
axs[0][2].set_title("vmin = 25, vmax = 75", fontsize = 10)

## Segunda linha de subfiguras
# Primeira Subfigura
axs[1][0].imshow(I, vmin = -1, vmax = 0, cmap="gray")
axs[1][0].set_title("vmin = -1, vmax = 0", fontsize = 10)

# Segunda Subfigura
axs[1][1].imshow(I, vmin = 115, vmax = 125, cmap="gray")
axs[1][1].set_title("vmin = 115, vmax = 125", fontsize = 10)
```

```
# Terceira Subfigura
axs[1][2].imshow(I, vmin = 255, vmax = 256, cmap="gray")
axs[1][2].set_title("vmin = 255, vmax = 256", fontsize = 10)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[19]:
```
Text(0.5, 1.0, 'vmin = 255, vmax = 256')
```

### 1.4 Questão 02

**O que acontece com os valores fora do intervalo [vmin, vmax]?**

Na array, os valores não mudam e continuam armazenados da mesma forma. Os valores vmin e vmax só interferem na exibição visual da plotagem, onde são usados para definir limites de cores dentro dos colormaps.

### 1.5 3. Convertendo Tipos de Dados:

[21]:
```
%matplotlib notebook

fig, axs = plt.subplots(nrows = 1, ncols = 3, figsize=(8, 6))

# É preciso se atentar aos valores da matriz ao converter entre os tipos de␣
 ↪array
I16 = np.array( [[  0,   0,   0,   0,   0,   0],
                 [ 50,  50,  50,  50,  50,  50],
                 [100, 100, 100, 100, 100, 100],
                 [150, 150, 150, 150, 150, 150],
                 [200, 200, 200, 200, 200, 200],
                 [250, 250, 250, 250, 250, 250],
                 [300, 300, 300, 300, 300, 300] ] ).astype(np.uint16)

axs[0].imshow(I16, vmin=0, vmax=255, cmap="gray")
axs[0].set_title("Imagem Original uint16", fontsize=10)
axs[0].set_xlabel("vmin = 0\n vmax = 255", fontsize=10)

# A conversão direta sem o reescalonamento dos valores pode produzir resultados␣
 ↪inesperados
I8a = I16.astype(np.uint8)
axs[1].imshow(I8a, vmin=0, vmax=255, cmap="gray")
axs[1].set_title("Imagem uint8 s/ \nreescalonamento", fontsize=10)
```

```
axs[1].set_xlabel("vmin = 0\n vmax = 255", fontsize=10)

# O reescalonamento melhora os resultados, mas como uint8 não aceita valores␣
 ↪decimais ocorrem erros de quantização
rescale_factor = 255 / np.max(I16)
I8b = ( rescale_factor * I16 ).astype(np.uint8)
axs[2].imshow(I8b, vmin=0, vmax=255, cmap="gray")
axs[2].set_title("Imagem uint8 c/ \nreescalonamento", fontsize=10)
axs[2].set_xlabel("vmin = 0\n vmax = 255", fontsize=10)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[21]:
```
Text(0.5, 0, 'vmin = 0\n vmax = 255')
```

[22]:
```
%matplotlib notebook

fig, axs = plt.subplots(nrows = 1, ncols = 3, figsize=(8, 6))

# É preciso se atentar aos valores da matriz ao converter entre os tipos de␣
 ↪array
I8 = np.array( [[ 36,  47,  75,  32,  18],
                [  0,  90,   0,  67,  29],
                [150,  22,  50,  20,  54],
                [ 50,  34,  65,  40,  50],
                [  6,  50,   5,  50,  13],
                [ 39,   1,  50,  25,   0]] ).astype(np.uint8)

axs[0].imshow(I8, vmin=0, vmax=255, cmap="gray")
axs[0].set_title("Imagem Original uint8", fontsize=10)
axs[0].set_xlabel("vmin = 0\n vmax = 255", fontsize=10)

# Imagens formadas a partir de arrays float32 são tipicamente utilizadas em␣
 ↪aplicações de IA como Redes Convolucionais.
# Nesse caso, a conversão pode ser realizada dividindo os elementos da matriz␣
 ↪por 255, maior valor para imagens uint8.
# Como o tipo float32 aceita valores decimais e dispõe de mais bits para␣
 ↪representar os valores, não ocorrem perdas.
Ifloat = (I8 / 255).astype(np.float32)
axs[1].imshow(Ifloat, vmin=0, vmax=1, cmap="gray")
axs[1].set_title("Imagem Float", fontsize=10)
axs[1].set_xlabel("vmin = 0\n vmax = 1.0", fontsize=10)

# Imagens (ou máscaras) binárias são geralmente utilizadas para a extração de␣
 ↪características em imagens.
```

```
# Nesse caso, a conversão pode ser realizada a partir de um processo de␣
 ↪limiarização (thresholding).
# Os valores booleanos podem ser convertidos em números (0s e 1s) mudando o␣
 ↪tipo de array para float ou uint.
Ibin = (I8 >= 50)
axs[2].imshow(Ibin, cmap="gray")
axs[2].set_title("Imagem Binária", fontsize=10)
axs[2].set_xlabel("vmin = False\n vmax = True", fontsize=10)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[22]:
```
Text(0.5, 0, 'vmin = False\n vmax = True')
```

## 1.6 Questão 03

**a. Produza as seguintes matrizes:**

```
A = [ [16, 3, 2, 13], [6, 9, 12, 7], [5, 10, 11, 8], [4, 15, 14, 8] ]

B = [ [16, 8, 2, 4], [20, 30, 40, 50], [5, 7, 8, 11] ]

C = [ [20, 30, 40], [50, 90, 15], [80, 30, 10] ]

D = [ [1, 2, 3, 4], [5, 6, 7, 8], [9, 8, 8, 9], [8, 7, 7, 8], [4, 5, 9, 8] ]
```

[27]:
```
A = np.array([ [16, 3, 2, 13], [6, 9, 12, 7], [5, 10, 11, 8], [4, 15, 14, 8] ])
B = np.array([ [16, 8, 2, 4], [20, 30, 40, 50], [5, 7, 8, 11] ])
C = np.array([ [20, 30, 40], [50, 90, 15], [80, 30, 10] ])
D = np.array([ [1, 2, 3, 4], [5, 6, 7, 8], [9, 8, 8, 9], [8, 7, 7, 8], [4, 5,␣
 ↪9, 8] ])
print("A:", A)
print("B:", B)
print("C:", C)
print("D:", D)
```

```
A: [[16  3  2 13]
 [ 6  9 12  7]
 [ 5 10 11  8]
 [ 4 15 14  8]]
B: [[16  8  2  4]
 [20 30 40 50]
 [ 5  7  8 11]]
C: [[20 30 40]
 [50 90 15]
```

```
 [80 30 10]]
D: [[1 2 3 4]
 [5 6 7 8]
 [9 8 8 9]
 [8 7 7 8]
 [4 5 9 8]]
```

**b. Plote as seguintes Imagens de Intensidade e comente os resultados.**

- IA = A
- IB = B, vmin = 4, vmax = 30
- IC = C, vmin = 15, vmax = 20
- ID = D, vmin = 0, vmax = 20

[36]:
```
%matplotlib notebook
fig, axs = plt.subplots(nrows = 2, ncols = 2, figsize=(8, 6))

axs[0][0].imshow(A, cmap="gray")
axs[0][0].set_title("IA", fontsize = 10)

axs[0][1].imshow(B, vmin = 4, vmax = 30, cmap="gray")
axs[0][1].set_title("IB, vmin = 4, vmax = 30", fontsize = 10)

axs[1][0].imshow(C, vmin = 15, vmax = 20, cmap="gray")
axs[1][0].set_title("IC, vmin = 15, vmax = 20", fontsize = 10)

axs[1][1].imshow(D, vmin = 0, vmax = 20, cmap="gray")
axs[1][1].set_title("ID, vmin = 0, vmax = 20", fontsize = 10)

plt.tight_layout()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

**c. Converta a Matriz G abaixo para:**

- Uma imagem de intensidade com valores entre [0, 1].
- Uma imagem binária com limiar de 0.25.
- Uma imagem com valores no intervalo [0, 255].
- Comente os resultados encontrados.

  G = [ [-0.7, 1.2, 0.4, -0.6, -0.4, 1.2], [-1.6, -0.6, 0.4, 0.8, 0.0, 0.8], [1.5, -0.5, 0.1, 0.2, -0.8, 1.1], [2.1, 0.9, 0.5, -0.5, -1.6, -0.7], [0.1, -1.4, 1.1, -1.2, 0.2, -0.9], [-1.0, -2.0, 1.0, 0.6, -0.1, -1.3] ]

[42]:
```
%matplotlib notebook
fig, axs = plt.subplots(nrows = 1, ncols = 3, figsize=(8, 6))

G = np.array([ [-0.7, 1.2, 0.4, -0.6, -0.4, 1.2],
               [-1.6, -0.6, 0.4, 0.8, 0.0, 0.8],
               [1.5, -0.5, 0.1, 0.2, -0.8, 1.1],
               [2.1, 0.9, 0.5, -0.5, -1.6, -0.7],
               [0.1, -1.4, 1.1, -1.2, 0.2, -0.9],
               [-1.0, -2.0, 1.0, 0.6, -0.1, -1.3] ])

axs[0].imshow(G, vmin = 0, vmax = 1, cmap="gray")
axs[0].set_title("Intensidade, vmin=0, vmax=1", fontsize = 10)

G2 = (G >= 0.25)
axs[1].imshow(G2, cmap="gray")
axs[1].set_title("Binário, thresh=0.25", fontsize = 10)

axs[2].imshow(G, vmin = 0, vmax = 255, cmap="gray")
axs[2].set_title("vmin=0, vmax=255", fontsize = 10)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[42]:
```
Text(0.5, 1.0, 'vmin=0, vmax=255')
```

As 3 subfiguras são representações diferentes dos mesmos dados. A primeira oferece uma maior diferenciação entre os valores da matriz. A segunda oferece uma representação binária em preto e branco com valor limitante de 0.25. A terceira exibe uma má representação, onde as variações de valores da imagem são imperceptiveis aos olhos humanos.

**d. Com base na matriz G, comente a diferença entre as imagens G1 e G2 abaixo:**

- Observe os valores de diferentes pixels nas mesmas posições.

[56]:
```
%matplotlib notebook

G = np.array([ [-0.7, 1.2, 0.4, -0.6, -0.4, 1.2],
               [-1.6, -0.6, 0.4, 0.8, 0.0, 0.8],
               [1.5, -0.5, 0.1, 0.2, -0.8, 1.1],
               [2.1, 0.9, 0.5, -0.5, -1.6, -0.7],
               [0.1, -1.4, 1.1, -1.2, 0.2, -0.9],
               [-1.0, -2.0, 1.0, 0.6, -0.1, -1.3] ]).astype(np.float32)

# Produzindo G1 a partir de G
G1 = G - np.min(G)
rescale_factor = 255 / np.max(G1)
```

```
G1 = (rescale_factor * G1).astype(np.uint8)
G1 = (G1 / 255).astype(np.float32)

# Produzindo G2 a partir de G
G2 = G - np.min(G)
G2 = (G2 / np.max(G2)).astype(np.float32)

fig, axs = plt.subplots(nrows = 1, ncols = 2, figsize=(8, 6))

axs[0].imshow(G1, vmin=0, vmax=1, cmap="gray")
axs[0].set_title("Imagem G1", fontsize=10)

axs[1].imshow(G2, vmin=0, vmax=1, cmap="gray")
axs[1].set_title("Imagem G2", fontsize=10)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[56]:
```
Text(0.5, 1.0, 'Imagem G2')
```

Existe uma pequena diferença nas casas decimais em algumas das células das duas imagens. Ambas matrizes G1 e G2 são resultados de um processamento da matriz G que busca converter seus falores para float32. Porém, por realizar operações diferentes, mesmo que com o mesmo objetivo, os resultados tendem a diferir nas casas decimais menos significativas.

## 1.7 4. Lendo imagens do computador

A biblioteca OpenCV possibilita carregar imagens do computador como arrays do numpy através da função **imread** (Observe que o comando cv2.imread lê uma imagem colorida como BGR, enquanto o comando plt.imread lê como RGB):

- **Sintaxe:** im = cv2.imread( filepath, 0 )
- **Descrição:** A função carrega a o arquivo cujo caminho está contido na string e a armazena na variável im como um array do numpy. O flag 0 indica o carregamento do arquivo como imagem monocromática.

## 1.8 Questão 04

**a.** Realize a leitura da imagem lenna.jpg (ou outra a sua escolha) e utilize o matplotlib para visualizar a imagem. Analise a variável im produzida, verificando suas dimensões e valores máximo e mínimo.

[50]:
```
plt.figure()
im = cv2.cvtColor(cv2.imread('lenna.jpg'), cv2.COLOR_BGR2RGB)
```

```
plt.imshow(im)
print('Shape:', im.shape)
print('Max:', im.max())
print('Min:', im.min())
```

```
<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


Shape: (240, 256, 3)
Max: 255
Min: 0
```

**b.** Faça o mesmo para a imagem morangos.jpg (ou outra a sua escolha), mas utilize também a mágica "%matplotlib notebook". Explore as ferramentas da janela produzida.

[54]:
```
%matplotlib notebook

plt.figure()
im = cv2.cvtColor(cv2.imread('morango.PNG'), cv2.COLOR_BGR2RGB)
plt.imshow(im)
print('Shape:', im.shape)
print('Max:', im.max())
print('Min:', im.min())
```

```
<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


Shape: (192, 189, 3)
Max: 255
Min: 0
```

A biblioteca OpenCV possibilita redimensionar imagens carregadas através da função **resize**:

- **Sintaxe:** dst_img = cv2.resize( src_img, (cols, rows) )
- **Descrição:** A função altera as dimensões da imagem para as dimensões fornecidas (cols, rows), retornando , que é a imagem redimensionada. pode ser uma imagem monocromática, colorida ou binária e as novas dimensões podem ser maiores ou menores que as originais.

**c.** Carregue a imagem digital.jpg e altere suas dimensões utilizando a função resize para as dimensões listadas a seguir. Verifique as dimensões de cada imagem para confirmar o redimensionamento.

- 240 x 240
- 120 x 120
- 60 x 60
- 30 x 30

[71]:
```
im = cv2.cvtColor(cv2.imread('mandril.tiff'), cv2.COLOR_BGR2RGB)
im_240 = cv2.resize( im, (240, 240) )
print('Shape (240):', im_240.shape)
print('Max (240):', im_240.max())
print('Min (240):', im_240.min(), '\n')

im_120 = cv2.resize( im, (120, 120) )
print('Shape (120):', im_120.shape)
print('Max (120):', im_120.max())
print('Min (120):', im_120.min(), '\n')

im_60 = cv2.resize( im, (60, 60) )
print('Shape (60):', im_60.shape)
print('Max (60):', im_60.max())
print('Min (60):', im_60.min(), '\n')

im_30 = cv2.resize( im, (30, 30) )
print('Shape (30):', im_30.shape)
print('Max (30):', im_30.max())
print('Min (30):', im_30.min(), '\n')
```

```
Shape (240): (240, 240, 3)
Max (240): 254
Min (240): 2

Shape (120): (120, 120, 3)
Max (120): 255
Min (120): 2

Shape (60): (60, 60, 3)
Max (60): 255
Min (60): 3

Shape (30): (30, 30, 3)
Max (30): 251
Min (30): 6
```

**d.** Plote as imagens redimensionadas utilizando subplots e comente os resultados observados.

[72]:
```
%matplotlib notebook

fig, axs = plt.subplots(nrows = 2, ncols = 2, figsize=(8, 6))
# 240x240
axs[0][0].imshow(im_240)
axs[0][0].set_title('240x240')
# 120x120
axs[0][1].imshow(im_120)
```

```
axs[0][1].set_title('120x120')
# 60x60
axs[1][0].imshow(im_60)
axs[1][0].set_title('60x60')
# 30x30
axs[1][1].imshow(im_30)
axs[1][1].set_title('30x30')
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[72]:
```
Text(0.5, 1.0, '30x30')
```

Com a diminuição do número de pixeis nas imagens, ocorre um efeito de pixelização e redução de definição das imagens.

[ ]:
```

```