

# Exercício-Programa II de MAC0210

# 1 Parte 0 - Laboratório

Nesta parte do problema tivemos que fazer a implementação das funções que servirão para o estudo da interpolação de polinômios aplicados a imagens. As decisões de projeto e detalhes da implementação estão descritos abaixo.

Vale lembrar que as funções trabalham com imagens com três canais de cores. Portanto, quando nos referirmos a um ponto de uma matriz, na verdade estaremos nos referindo a um vetor de três coordenadas, onde cada coordenada representa uma cor (no caso, vermelho, verde e azul, respectivamente). As matrizes que usamos têm três dimensões.

## 1.1 compress.m

Esse arquivo contém apenas uma função, a `compress`, com o seguinte protótipo:

```
function compress (originalImg, k)
```

Ela recebe um arquivo de imagem no formato *png* e devolve, em um outro arquivo *png*, a imagem comprimida com a taxa  $k$ .

A leitura da imagem recebida é armazenada em uma matriz grande. Comprimimos retirando todas as linhas e colunas  $i$  tal que  $i\%(k + 1) = 1$ , onde  $\%$  representa a operação de resto.

A compressão é feita sequencialmente, analisando a matriz grande por pontos. Quando encontramos um par (linha, coluna) em que as duas satisfazem o requisito, o atribuímos à matriz pequena. O ponto  $(x, y)$  da matriz grande é colocado no seguinte ponto da matriz pequena:  $(\left\lfloor \frac{x}{k+1} + 1 \right\rfloor, \left\lfloor \frac{y}{k+1} + 1 \right\rfloor)$ .

Feito isso, a matriz pequena é escrita numa imagem *png*.

## 1.2 calculateError.m

O arquivo possui apenas uma função, a `calculateError`, que tem o protótipo:

```
function calculateError(originalImg, decompressedImg)
```

Essa função calcula o erro relativo entre duas imagens (aqui, usamos para a imagem comprimida e uma imagem descomprimida), usando as fórmulas fornecidas no enunciado.

Primeiramente, lemos as imagens e armazenamos em matrizes, então, fazemos a conta. Não há muito segredo aqui. O único detalhe da implementação é que usamos a função `norm` do Octave para calcular a norma euclidiana.

### 1.3 decompress.m

Esse é o function file mais importante dos três enviados. Ele faz a descompressão de uma imagem usando algum método de interpolação e nos devolve o arquivo com a imagem descomprimida. Ele possui as seguintes funções, com os protótipos:

```
function decompress (compressedImg, method, k, h)
```

A função `decompress` recebe uma imagem em *png* e a descomprime em uma razão *k*, utilizando o método bilinear ou o método bicúbico.

Ela lê a imagem e a armazena em uma matriz. A descompressão será feita inserindo *k* linhas e colunas entre as linhas e colunas da matriz. Calculamos o tamanho da *p* da imagem descomprimida usando a fórmula  $p = n + (n - 1) \cdot k$ , onde *n* é o tamanho da matriz da imagem pequena.

Dependendo do método escolhido pelo usuário, ela chama a função que desenvolverá o método da interpolação, que devolverá uma matriz com os pontos interpolados.

Feito isso, essa matriz é escrita em um arquivo *png*.

```
function B = bilinear (A, k, h, p)
```

Esse representa um dos métodos de interpolação descrito no enunciado, o método Bilinear por partes.

Para começar, chamamos a função `expand`, que nos devolve uma matriz com o tamanho que precisamos (mais detalhes abaixo).

Com essa matriz definimos quadrados de lado  $k + 2$  e, então, armazenamos os vértices do quadrado. Usamos  $X = \text{inv}(A) * B$  para resolver o sistema linear do método da interpolação (no enunciado está na forma  $B = AX$ ). Com a matriz  $X$  dos valores da solução, fazemos a interpolação para cada cor de todos os pontos dentro de cada quadrado definido, usando a fórmula dada:

$$f(x, y) \approx p_{ij}(x, y) = a_0 + a_1(x - x_i) + a_2(y - y_j) + a_3(x - x_i)(y - y_j)$$

No entanto, fizemos pequenas adaptações para a implementação funcionar:

$f = X(1) + X(2) \cdot x + X(3) \cdot y + X(4) \cdot x \cdot y$ ; , onde:

- $f$  é o resultado do polinômio interpolador.

- $X(i)$  é o correspondente a  $a_{i-1}$ , proveniente da solução do sistema linear.
- $x$  e  $y$  são as coordenadas do ponto no quadrado, definidos assim:
  - $x = ((m-i)/(k+1)) * h$ ; , onde  $m$  é a real coordenada do ponto na matriz,  $i$  é o início do quadrado na matriz grande,  $k$  é a taxa de descompressão e  $h$  é referente ao lado do quadrado, definido no enunciado.
  - $y = ((n-j)/(k+1)) * h$ ; , onde  $n$  é a real coordenada do ponto na matriz,  $j$  é o início do quadrado na matriz grande, e o resto é análogo.
- Isso faz com que os índices dentro do quadrado estejam entre 0 e  $h$ , e, por consequência, o quadrado tenha lado de tamanho  $k$ .

O cálculo é feito para todos os pontos que não são os vértices do quadrado.

Em nosso método, consideramos  $(x_0, y_0)$  como  $(1, 1)$  da matriz e  $(x_{p-1}, y_{p-1})$  como  $(p, p)$  da matriz.

```
function B = bicubico (A, k, h, p)
```

Essa função representa o outro método de interpolação descrito no enunciado, o método Bicubico.

Da mesma forma que no outro método, chamamos a função `expande` que devolve uma matriz com o tamanho necessário (mais detalhes abaixo).

Para o cálculo das derivadas parciais existem 3 funções: `derivax`, `derivay` e `derivaxy` que já verificam as condições para as derivadas na borda. Uma observação a se realizar é que consideramos que nos casos em que aparecem pontos que extrapolam a grade fizemos os cálculos utilizando a diferença unilateral com o ponto da borda e seu adjacente.

Para cada quadrado  $Q_{ij}$  descrito no enunciado calculamos a matriz com os 16 coeficientes de  $p_{ij}$ . Calculamos os valores das cores de cada pixel da imagem com a fórmula do polinômio interpolador para cada quadrado  $Q_{ij}$  do enunciado.

```
function B = expande (A, k, p)
```

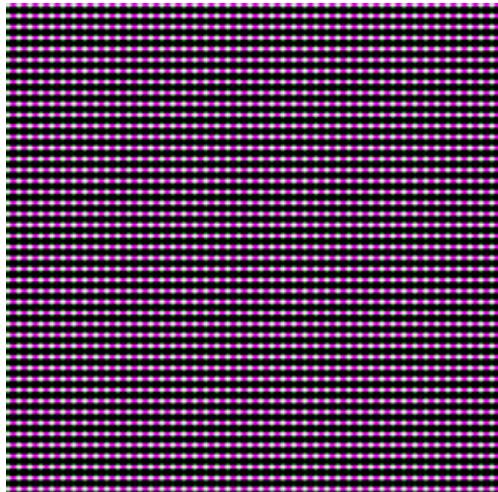
Essa função faz algo parecido com o oposto do descrito em **compress.m**

Recebe uma matriz quadrada  $A$  de tamanho  $p$  e coloca  $k$  linhas e colunas de zeros entre suas linhas e colunas, atribuindo tudo isso em uma matriz  $B$ .

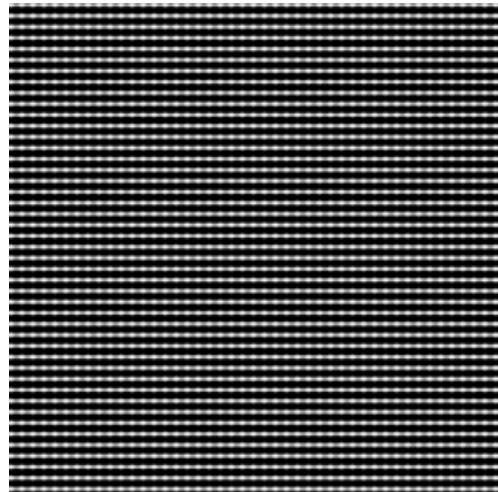
Vamos andando ponto a ponto, e caso esse ponto tenha o par (linha, coluna) =  $(i, j)$  tal que ambos os números cumpram o requisito  $x\%(k + 1) = 1$ , onde  $\%$  representa a operação de resto, o correspondente  $(\frac{i-1}{k+1} + 1, \frac{j-1}{k+1} + 1)$  da matriz  $A$  é atribuído na matriz  $B$ .

## 2 Parte 1 - Zoológico

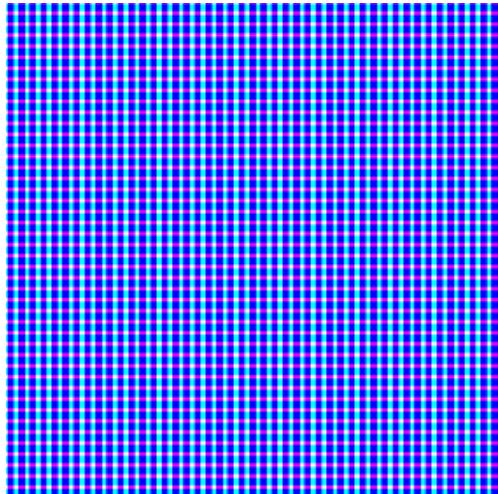
Para os experimentos do zoológico, utilizamos as seguintes imagens:



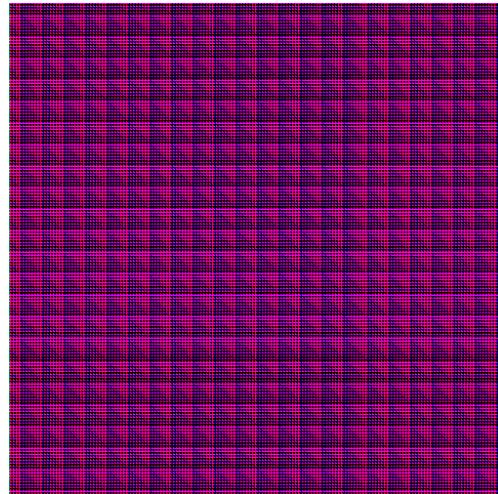
(a) Figura 1



(b) Figura 1 em preto e branco.



(a) Figura 2



(b) Figura 3

- Figura 1: gerada através da função  $f(x, y) = (\text{sen}(x), \frac{\text{sen}(y)+\text{sen}(x)}{2}, \text{sen}(x))$ , descrita no enunciado (281 por 281 pixels);
- Figura 2: gerada através da função  $f(x, y) = (\cos(x), 2 \cdot \cos(y), x^2)$  (281 por 281 pixels);
- Figura 3: gerada por  $f(x, y) = (\tan(x) - \tan(y), \tan(y) - y, \tan(x))$  (505 por 505 pixels).

## 2.1 Usando o método Bilinear

Usando o método Bilinear, descomprimimos as imagens, e obtivemos as seguintes respostas para as perguntas do enunciado:

**1.** Funciona bem para imagens preto e branco?

Sim, pois comparando com uma imagem colorida, o erro não foi muito diferente.

**2.** Funciona bem para imagens coloridas?

Funciona. As imagens ficam visualmente parecidas e o erro fica pequeno (a maioria na ordem de  $10^{-1}$ )

**3.** Funciona bem para todas as funções de classe  $C^2$ ?

Sim, como exemplificado pelas figuras 1 e 2.

**4.** E para funções que não são de classe  $C^2$ ?

Também funciona, como na figura 3.

**5.** Como o valor de  $h$  muda a interpolação?

O valor de  $h$  não parece interferir tanto na interpolação, pois, como exibido abaixo pelas figuras (A) e (B), que foram descomprimidas com  $h$  diferente ( $h = 5$  e  $h = 7$ ), o erro ficou muito próximo.

**6.** Como se comporta o erro?

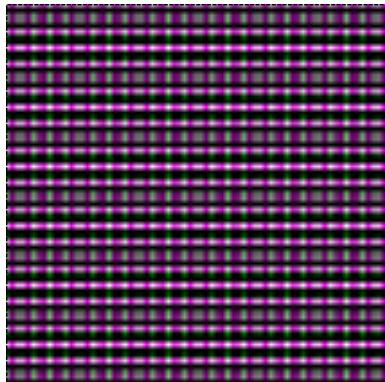
Os erros ficaram bem pequenos, como exibido abaixo. Foi interessante ver que mudando a cor de uma imagem seu erro muda, e que quanto menor o  $h$  usado para compressão e decompressão, menor o erro.

**7.** Considerando uma imagem de tamanho  $p^2$ , comprimida com  $k = 7$  e descomprimida com  $k = 7$  e com  $k = 1$  três vezes, quais são as conclusões?

As imagens ficam parecidas visualmente, porém o erro é menor (como mostram as figuras (B) e (C)).

Abaixo, seguem as imagens acima descomprimidas.

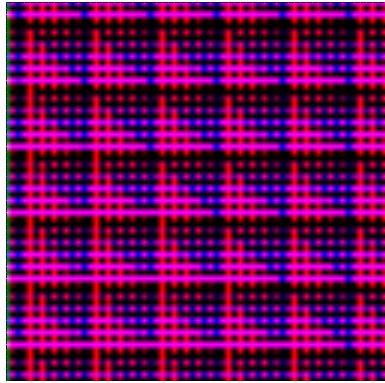
Obs: As imagens foram comprimidas e descomprimidas com o mesmo  $k$ , para manter o tamanho e o cálculo do erro ser facilitado.



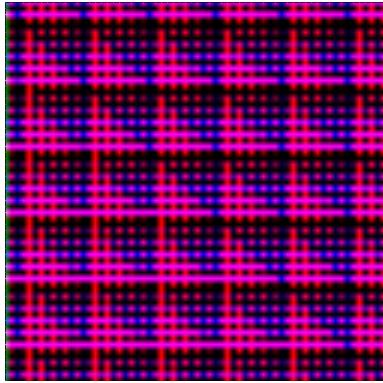
(a) 1 descomprimida com  $h = 4$  e  
 $k = 3$ , err = 0.76841



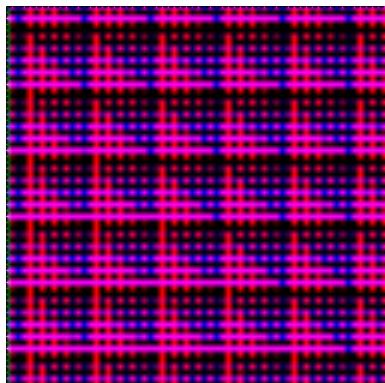
(b) 1 descomprimida com  $h = 4$  e  
 $k = 3$ , err = 0.77409



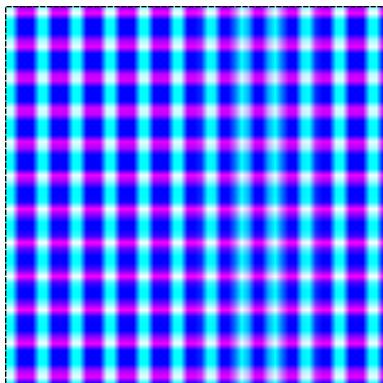
(a) 2 descomprimida com  $h = 5$  e  
 $k = 7$ , err = 1.0884 (FIGURA A)



(b) 2 descomprimida com  $h = 7$  e  
 $k = 7$ , err = 1.0884 (FIGURA B)



(a) 2 descomprimida com  $h = 2$  e  
 $k = 1$  três vezes, err = 0.09700  
(FIGURA C)



(b) 3 descomprimida com  $h = 2$  e  
 $k = 4$ , err = 0.66150

## 2.2 Usando o método Bicúbico

Usando o método Bicúbico, descomprimimos as imagens, e obtivemos as seguintes respostas para as perguntas do enunciado:

**1.** Funciona bem para imagens preto e branco?

As imagens ficaram um pouco diferente das originais, perdeu um pouco do padrão da imagem original, mas ainda mantiveram padrões dos níveis de cor da imagem original. O erro não é exatamente muito grande pois ainda é próximo de 1. Se comparado com a decompressão para as imagens coloridas, o erro e o resultado estão bem parecidos. Um fator curioso é que se comparado com o experimento das imagens reais o erro é maior.

**2.** Funciona bem para imagens coloridas?

Em relação à cor, apenas uma teve uma grande diferença no padrão de cor em relação à original, e como nas imagens preto e branco o erro não seja tão distante de 1. Em comparação com o experimento das imagens reais o erro é maior.

**3.** Funciona bem para todas as funções de classe  $C^2$ ?

Mantiveram os padrões de organização dos elementos da imagem, embora em um dos testes o resultado foi uma imagem com cores muito diferentes.

**4.** E para funções que não são de classe  $C^2$ ?

A imagem ficou bem diferente, a imagem perdeu parte do aspecto quadriculado e em todos os casos o erro ultrapassou 1.

**5.** Como o valor de  $h$  muda a interpolação?

Não tiveram diferenças significativas, conforme o  $h$  aumenta o erro aumenta.

**6.** Como se comporta o erro?

Ele diminui da imagem colorida para a preto e branco ainda que pouco, quase não muda quando se altera o  $h$  e diminui quando ocorre a compressão 3 vezes seguidas.

**7.** Considerando uma imagem de tamanho  $p^2$ , comprimida com  $k = 7$  e descomprimida com  $k = 7$  e com  $k = 1$  três vezes, quais são as conclusões?

Na compressão em três vezes seguidas o erro diminuiu se comparado com a compressão com uma vez.

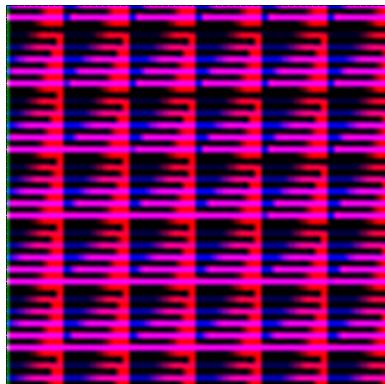
Abaixo, seguem as imagens acima descomprimidas.



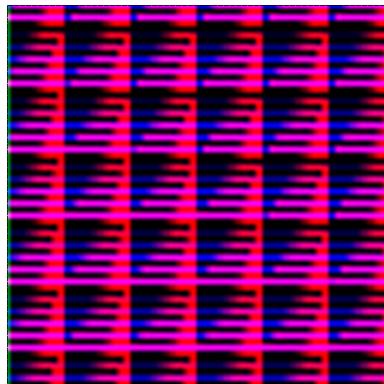
(a) 1 descomprimida com  $h = 4$  e  
 $k = 3$ , err = 0.85446



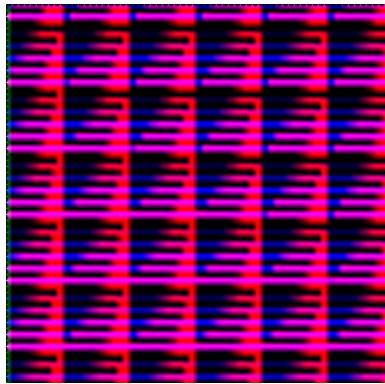
(b) 1 descomprimida com  $h = 4$  e  
 $k = 3$ , err = 0.82142



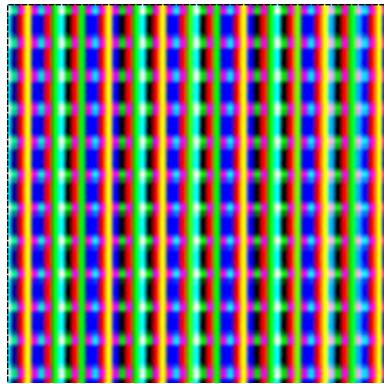
(a) 2 descomprimida com  $h = 5$  e  
 $k = 7$ , err = 1.0321



(b) 2 descomprimida com  $h = 7$  e  
 $k = 7$ , err = 1.0322



(a) 2 descomprimida com  $h = 2$  e  
 $k = 1$  três vezes, err = 1.0208



(b) 3 descomprimida com  $h = 2$  e  
 $k = 4$ , err = 0.88062

Obs: As imagens foram comprimidas e descomprimidas com o mesmo  $k$ , para manter o tamanho e o cálculo do erro ser facilitado.

### 3 Parte 2 - Selva

Para a selva, utilizamos as seguintes imagens:



(a) Figura 4



(b) Figura 5.



(c) Figura 6



(d) Figura 7

- Figura 4: Arara (colorida) (281 por 281 pixels);
- Figura 5: Cachorro (preto e branco) (505 por 505 pixels);
- Figura 6: Maçãs (colorida) (505 por 505 pixels);
- Figura 7: Bicicleta (preto e branco) (931 por 931 pixels).

#### 3.1 Usando o método Bilinear

Usando o método Bilinear, descomprimimos as imagens, e obtivemos as seguintes respostas para as perguntas do enunciado:

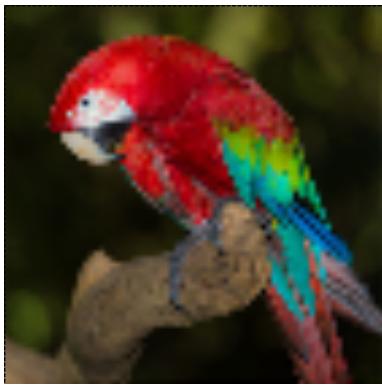
1. Funciona bem para imagens preto e branco? Sim, pois comparando com uma imagem colorida de mesmo tamanho, o erro fica diferente, mas pequeno.
2. Funciona bem para imagens coloridas? Também funciona. As imagens ficam visualmente parecidas e o erro fica pequeno (a maioria na ordem de  $10^{-2}$ )
3. Como o valor de  $h$  muda a interpolação? Em imagens reais, o valor de  $h$  não parece interferir tanto na interpolação, pois, como exibido abaixo pelas figuras (F) e (G), que foram descomprimidas com  $h$  diferente ( $h = 3$  e  $h = 5$ ), o erro ficou muito próximo, mas parece diminuir quando  $h$  aumenta.
4. Como se comporta o erro? Os erros ficaram bem pequenos, como exibido abaixo. Foi interessante ver que mudando a cor de uma imagem seu erro muda, e que quanto

menor o  $k$  usado para compressão e descompressão, menor o erro, como mostrado pelas imagens (D) e (E).

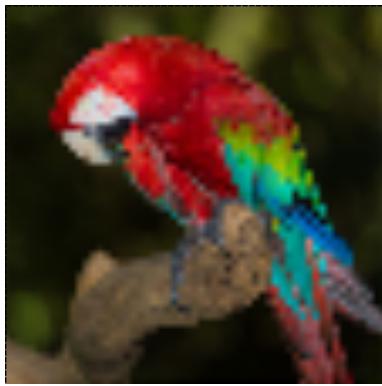
5. Considerando uma imagem de tamanho  $p^2$ , comprimida com  $k = 7$  e descomprimida com  $k = 7$  e com  $k = 1$  três vezes, quais são as conclusões?

As imagens ficam parecidas visualmente, porém o erro é maior (como mostram as imagens (H) e (I)).

Seguem as descomprimidas:



(a) 4 descomprimida com  $h = 4$  e  
 $k = 3$ , err = 0.057654 (FIGURA  
D)



(b) 4 descomprimida com  $h = 4$  e  
 $k = 4$ , err = 0.065963 (FIGURA  
E)



(a) 5 descomprimida com  $h = 3$  e  
 $k = 3$ , err = 0.033480 (FIGURA  
F)



(b) 5 descomprimida com  $h = 5$   
 $k = 7$ , err = 0.033479 (FIGURA  
G)



(a) 6 descomprimida com  $h = 2$  e  
k = 7, err = 0.080604 (FIGURA k = 1 três vezes, err = 0.093727  
H) (FIGURA I)



(a) 7 descomprimida com  $h = 5$  e  
k = 5, err = 0.042874

### 3.2 Usando o método Bicúbico

Usando o método Bilinear, descomprimimos as imagens, e obtivemos as seguintes respostas para as perguntas do enunciado:

1. Funciona bem para imagens preto e branco?

Sim, as imagens ficam bem parecidas, apenas com aparência pixelizada e o erro é da ordem  $10^{-3}$ . O erro é ligeiramente menor para imagens em preto e branco quando comparado com as imagens coloridas.

2. Funciona bem para imagens coloridas?

Sim, as imagens ficam parecidas, com aparência pixelizada e o erro é da ordem  $10^{-3}$ . O erro é maior do que quando comparado com imagens preto e branco.

- 3. Como o valor de  $h$  muda a interpolação?**

Apesar da variação não ser significativa, quanto maior o  $h$  menor é o erro (como vista na imagem do pug).

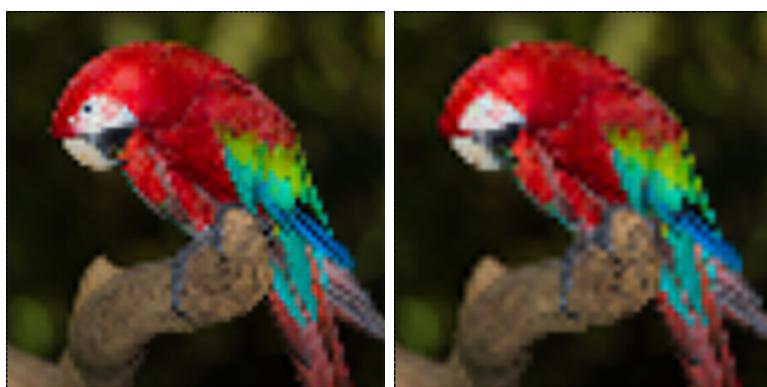
- #### 4. Como se comporta o erro?

Sempre na ordem de  $10^{-3}$ , varia muito pouco conforme variamos o  $h$ , quando aumentamos o  $k$  o erro aumenta e no caso em que ocorre compressão 3 vezes o erro aumenta.

5. Considerando uma imagem de tamanho  $p^2$ , comprimida com  $k = 7$  e descomprimida com  $k = 7$  e com  $k = 1$  três vezes, quais são as conclusões?

A compressão 3 vezes seguidas tem erro maior do que com 1 compressão.

Seguem as descomprimidas:



(a) 4 descomprimida com  $h = 4$  e (b) 4 descomprimida com  $h = 4$  e  
 $k = 3$ , err = 0.057363                            k = 4, err = 0.067982



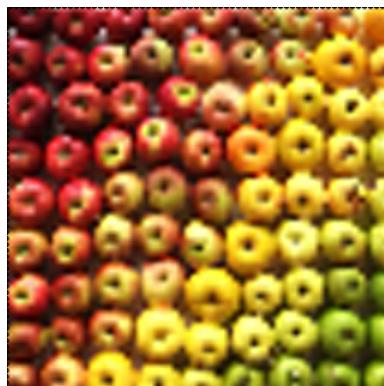
(a) 5 descomprimida com  $h = 3$  e  
 $k = 7$ , err = 0.033906



(b) 5 descomprimida com  $h = 5$  e  
 $k = 7$ , err = 0.033904



(a) 6 descomprimida com  $h = 2$  e  
 $k = 7$ , err = 0.079984



(b) 6 descomprimida com  $h = 2$  e  
 $k = 1$  três vezes, err = 0.091454



(a) 7 descomprimida com  $h = 5$  e  
 $k = 5$ , err = 0.043302