MC920— Processamento de Imagens

Professor: Hélio Predini

Leonardo Rodrigues Marques `RA: 178610`

Trabalho 1

1 Processamento de Imagens

Para usar o programa e gerar as imagens, instale as bibliotecas opencv2 e numpy e digite no terminal:

```
make run
```

As imagens geradas estarão na pasta output/xyz/, de acordo com o a secção x.y.z. Por exemplo, as imagens geradas da secção 1.1.c estarão na pasta ouput/11c/ e consecutivamente.

Para apagar as imagens, basta digitar no terminal:

```
make clean
```

O repositório do projeto está disponível em Github do Trabalho 1

As imagens foram carregadas usando a função **imread** do openCV e armazenadas em uma array **images**. O código está localizado no arquivo *load_images.py*

1.1 Transformação de Intensidade

b) Negativo da Imagem

Para se obter o negativo da imagem, foi usado a função $f(value_{pixel}) = 255 - value_{pixel}$. O resultado obtido foi o esperado, sem muita complexidade ou limitações técnicas. A função implementada está no arquivo *negate_images.py*.

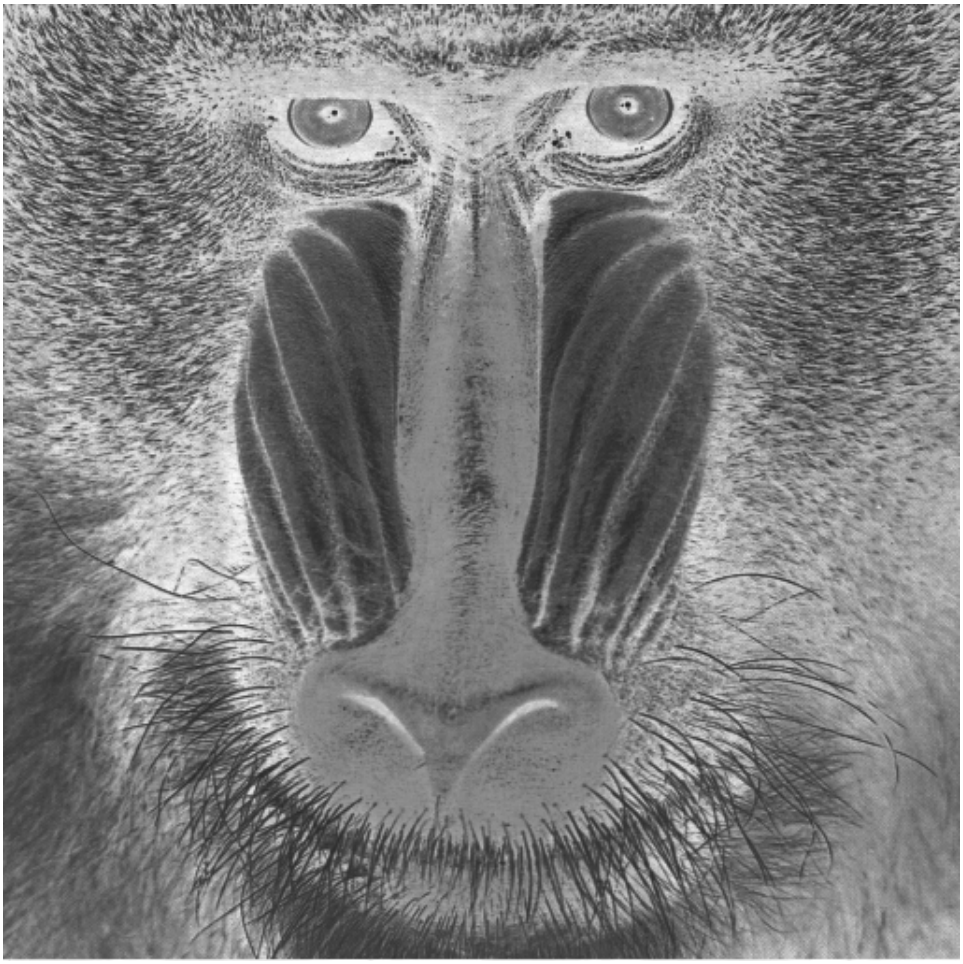

Figura 1: Negativo da Imagem 'baboon.png'

c) Imagem Transformada

Para converter a imagem para o intervalo requisitado [100, 200], foi usada a função $f(value_{pixel}) = \frac{1}{2.55} value_{pixel} + 100$. O resultado obtido foi o esperado, sem muitas limitações técnicas, apenas com cuidado no tratamento das conversões de float para int. A função implementada está no arquivo *transform_images.py*.

Figura 2: Transformação da Imagem 'city.png'

d) Linhas Pares Invertidades

Para se obter uma imagem com linhas pares invertidas, usamos a função **flip** do numpy em um loop com step 2 que percorria as linhas. O resultado obtido foi esperado, sem limitações ou complexidade. A função implementada está no arquivo *reverse_images.py*.

Figura 3: Linhas Pares Invertidas da Imagem 'city.png'

e) Reflexão das Linhas Pares

Para se obter a reflexão, setamos um loop decrescente com uma variação da metade da matriz até 0. O index i da operação era usado na imagem original para se obter as linhas e o complemento ($image.height - i$) era usado na nova imagem para atribuir as linhas extraídas da imagem original. Esse processo é muito delicado pois envolve limites de matriz, oque pode gerar erros fatais. A função implementada está no arquivo *reflection_images.py*.

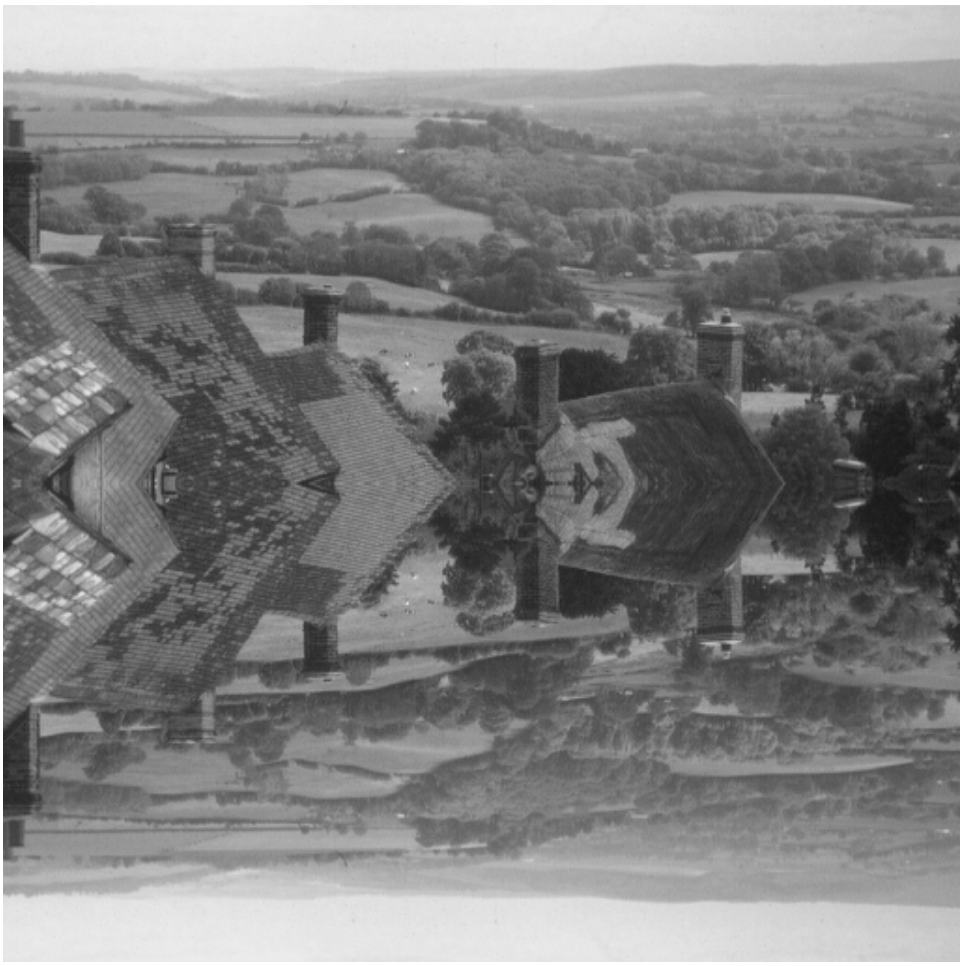

Figura 4: Reflexão da Linhas da Imagem 'city.png'

f) Espelhamento Vertical

Para se obter o Espelhamento Vertical, foi usado a função **flip** de numpy com **axis=0**. Não houve muita complexidade. O função implementada está no arquivo *mirroring_images.py*

Figura 5: Espelhamento Vertical da Linhas da Imagem 'city.png'

1.2 Ajuste de Brilho

Para aplicar a correção gama, a imagem foi normalizada usando a função $f(value_{pixel}) = \frac{value_{pixel}}{2.55}$, multiplicando-se pelo exponencial requerida com diferentes valores de gamma e posteriormente, reconvertida para os valores originais. A função está implementada no arquvivo *lighten_imgaes.py*.

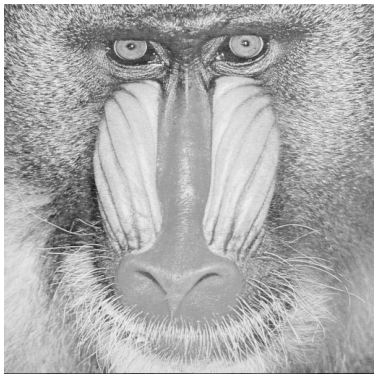

Figura 6: Correção do Brilho em 'baboon.png' com $\gamma = 1.5$

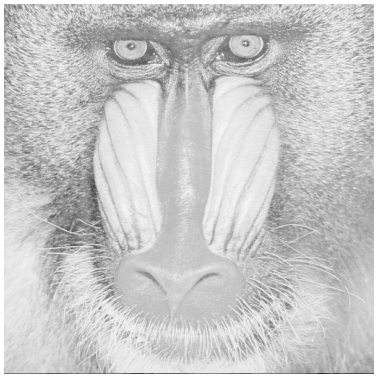

Figura 7: Correção do Brilho em 'baboon.png' com $\gamma = 2.5$

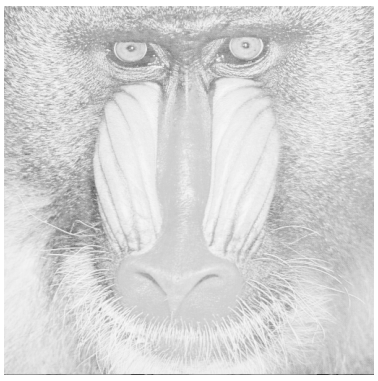

Figura 8: Correção do Brilho em 'baboon.png' com $\gamma = 3.5$

1.3 Plano de Bits

Para se obter o plano, foi usado operações de bitwise. Em primeiro lugar, o bit do plano era deslocado para direita para limpar os bits da direita e posteriormente era operado com a instrução e valor 1 para limpar os bits da esquerda. Logo, multiplicado por 255 para restaurar os valores padrões da imagem e obter uma imagem do plano em questão. A função implementada está no arquivo *bitplane_images.png*.

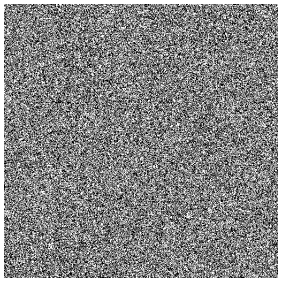

Figura 9: Imagem 'baboon.png' em plano 0

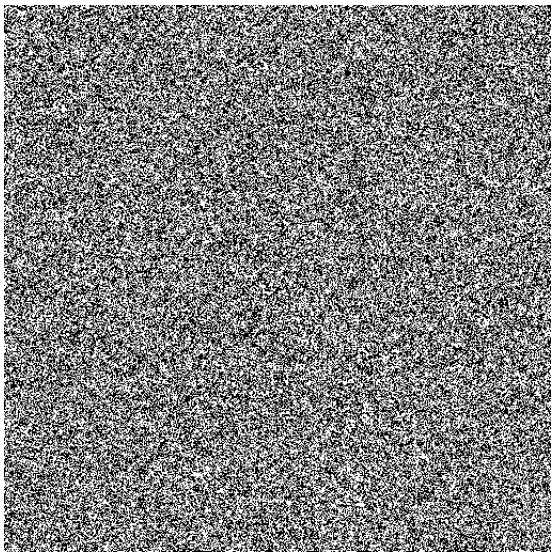

Figura 10: Imagem 'baboon.png' em plano 1

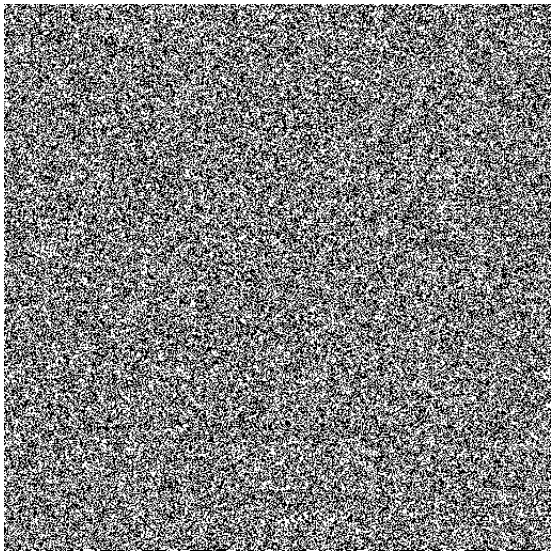

Figura 11: Imagem 'baboon.png' em plano 2

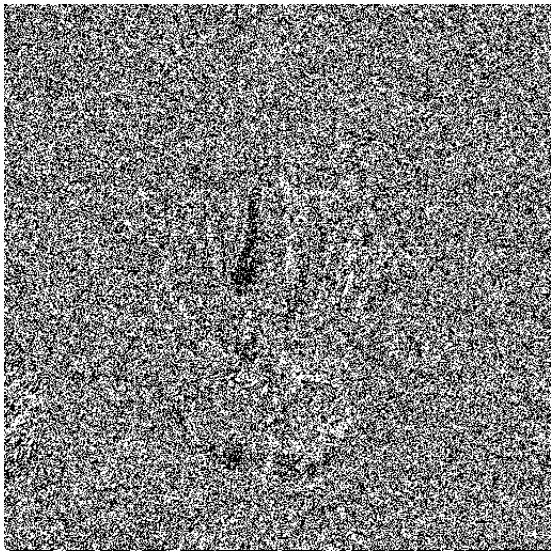

Figura 12: Imagem 'baboon.png' em plano 3

Figura 13: Imagem 'baboon.png' em plano 4

Figura 14: Imagem 'baboon.png' em plano 5

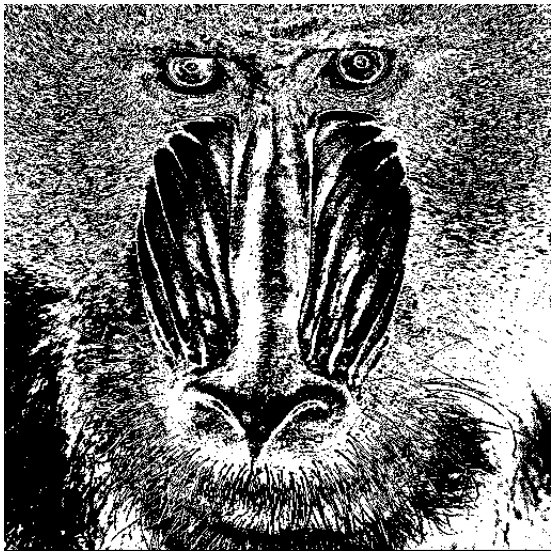

Figura 15: Imagem 'baboon.png' em plano 6

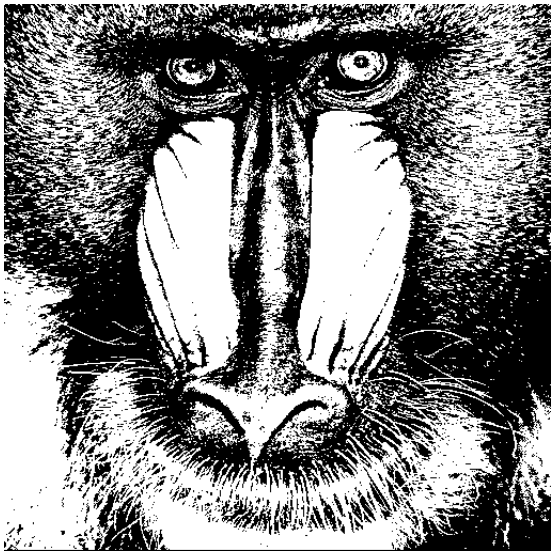

Figura 16: Imagem 'baboon.png' em plano 7

1.4 Mosaico

Para se obter o mosaico, foi criado um mapa com os blocos antigos relacionados aos novos. Utilizando desse mapeamento e da altura dos blocos, conseguimos obter os índices das linhas e das colunas do inicio e fim dos blocos a serem trocadas. Com essas índices, basta apenas copiar os valores dos pixels entres os blocos selecionados. A função implementada está no arquivo *mosaic_images.png*.

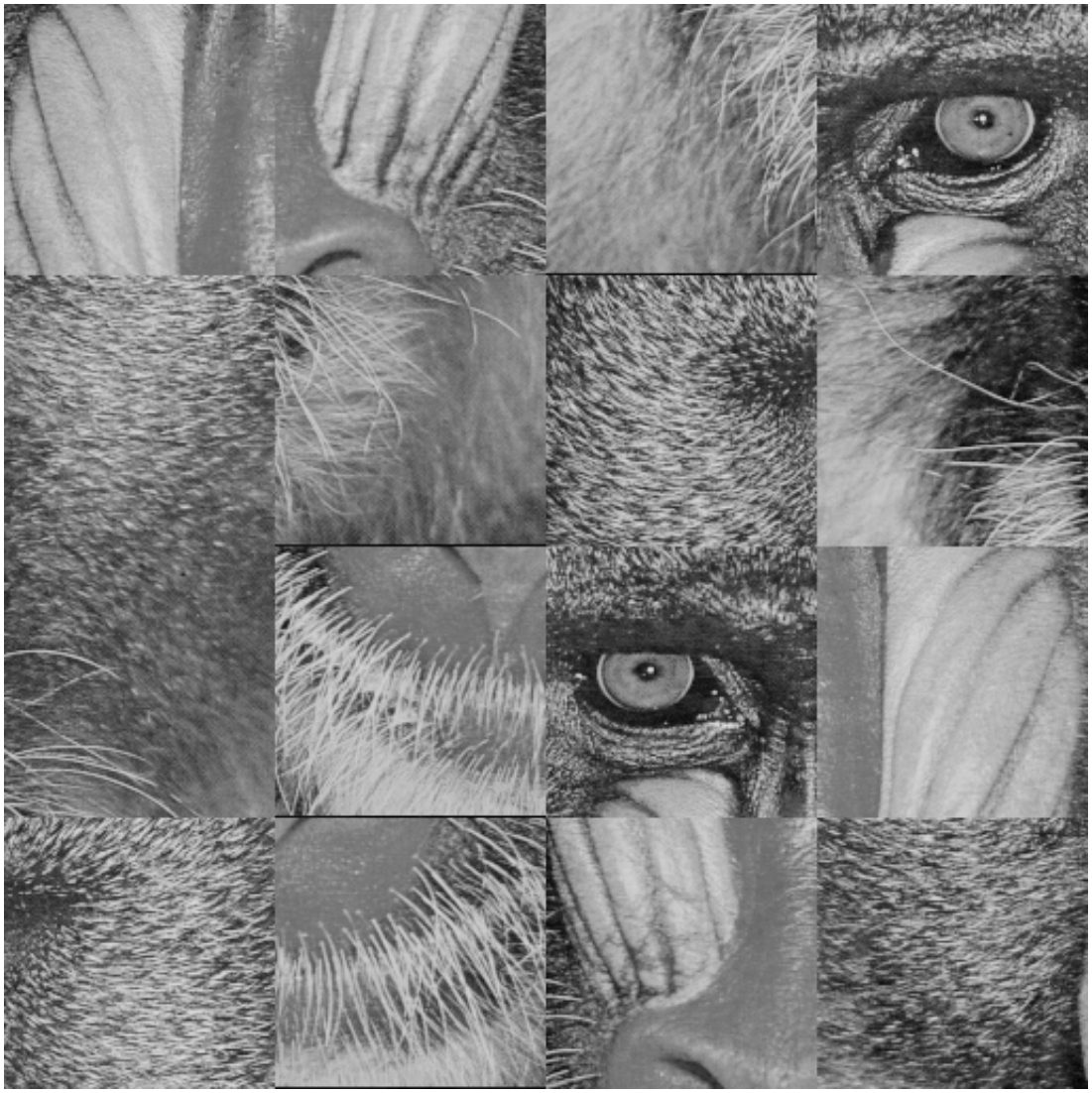

Figura 17: Mosaico montado sobre a Imagem 'baboon.pnh'

1.5 Combinação de Imagens

Para se obter a composição, apenas utilizou-se a função de ponderação apresentada: $f(valueA_{pixel}, valueb_{pixel}) = average * (valueA_{pixel}) + (1 - average) * (valueB_{pixel})$. Dessa forma, obtermos o resultado esperado. A função implementada está no arquvio *combine_images.py*.

Figura 18: Imagens 'baboon' e 'butterfly' combinadas com 0.2

Figura 19: Imagens 'baboon' e 'butterfly' combindas com 0.5

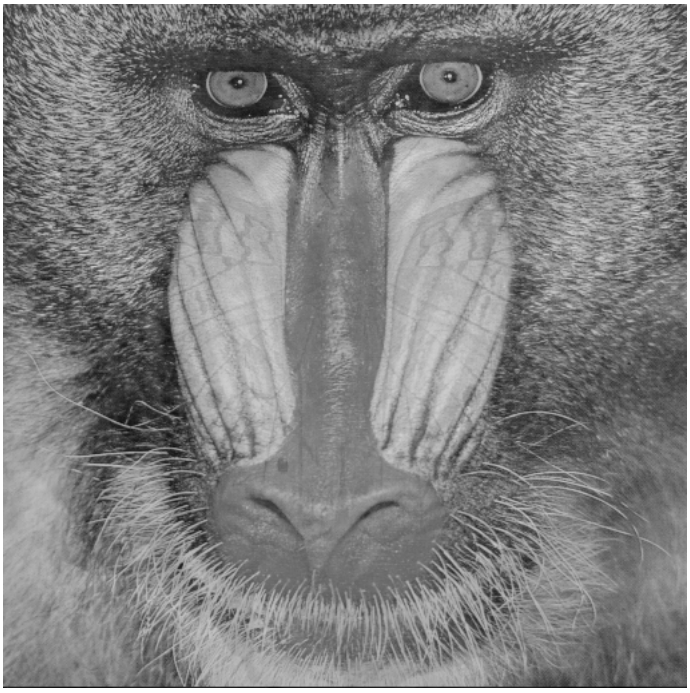

Figura 20: Imagens 'baboon' e 'butterfly' combinadas com 0.8

1.6 Filtragem de Imagens

Na filtragem, foi usado um a função filter2D do OpenCV.

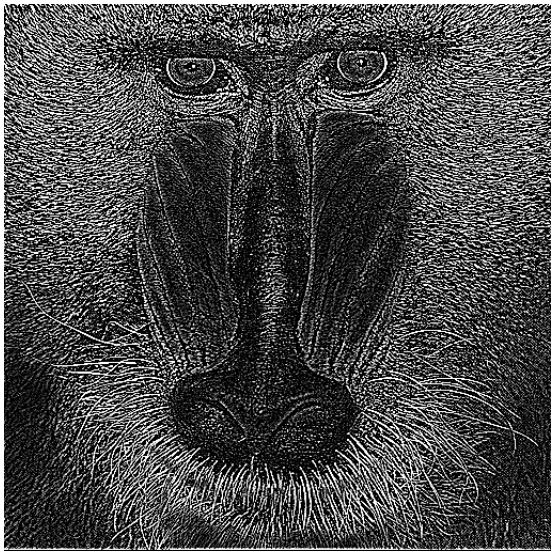

Figura 21: Imagem 'baboon.png'com filtro H1

Figura 22: Imagem 'baboon.png'com filtro H3

Figura 23: Imagem 'baboon.png'com filtro H4

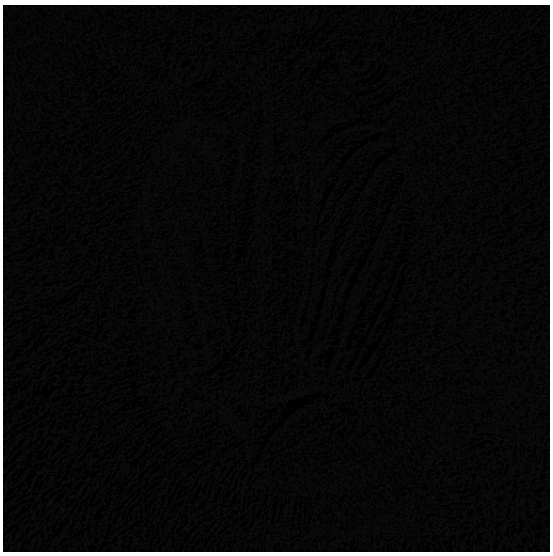

Figura 24: Imagem 'baboon.png'com filtro H3-H4 combinados pela função