

MC920/MO443 - Trabalho 1

Arthur Rezende 166003

1 Introdução

Este documento é um relatório contendo descrições e resultados do programa feito para o trabalho 1.

2 Descrição Geral

O código está organizado em um jupyter notebook chamado *trabalho1_166003.ipynb*. Com as imagens de entrada na mesma pasta. Para executar basta abrir o arquivo e rodar na pasta na qual foi enviada.

Para a última questão, afim de deixar o fluxo do notebook mais legível os filtros estão sendo importados do arquivo *kernels.py*. Para os exemplos nas instruções onde haviam imagens as mesmas foram utilizadas, para as questões sem especificação foram utilizadas imagens de preferência do autor.

Os parâmetros de tamanho e posições das imagens apresentadas como resultado foram selecionadas manualmente baseado no que apresentasse a melhor visualização das mesmas.

As bibliotecas utilizadas foram:

1. NumPy, para processamento vetorial das imagens
2. OpenCV, carregamento e manipulação de imagens
3. Matplotlib, visualização das imagens originais e alteradas

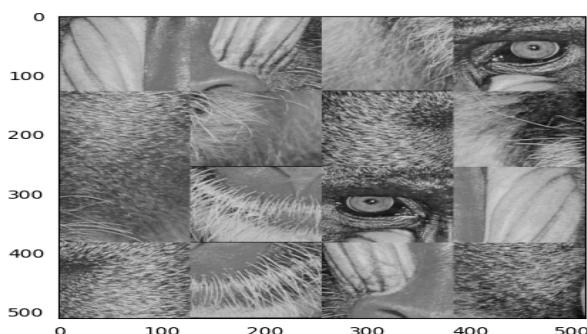
3 Questões

Nessa sessão cada questão será comentada e os resultados apresentados.

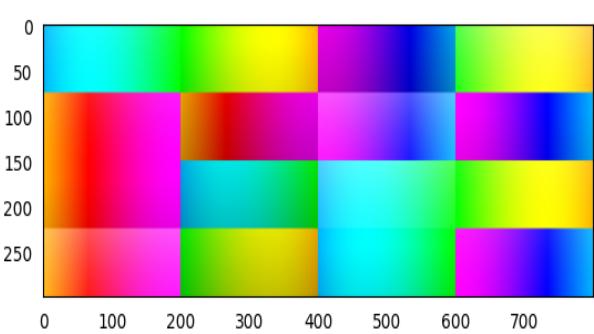
3.1 Mosaico

As peças do mosaico foram montadas dividindo a matriz original em quatro e cada divisão novamente em quatro. Foi utilizado a função `split` do numpy.

Assim obtendo 16 imagens, em seguida foram reordenadas usando uma máscara de índices com a ordem pre estabelecida no enunciado, para essa reordenação fosse possível foi preciso fazer um reshape para que o vetor das peças fosse linear pois caso contrário só seria possível alternar entre um bloco de 4 pedaços e não com todas as 16 partes. A solução também funciona para uma imagem retangular, conforme ilustrado na 1b



(a) Mosaico do Baboon



(b) Mosaico em Faixa RGB

Figura 1: Resultado do algoritmo do mosaico em duas imagens

3.2 Combinação de Imagens

Para combinar as imagens corretamente é necessário após a média ponderada converter as imagens para o tipo inteiro utilizando 8 bits. Para assegurar que o intervalo esteja entre 0 e 255, caso contrário as imagens podem não ser fielmente ou até mesmo totalmente representáveis. A saída do código está representada na figura 2.

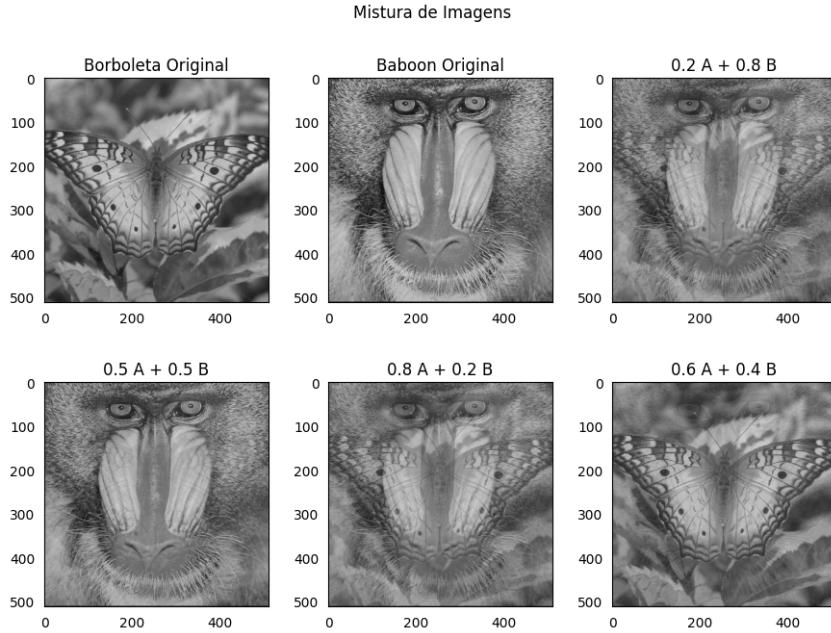


Figura 2: Imagens misturadas pela média ponderada

3.3 Transformação de Imagens

Neste item foram solicitados 5 operações. Sendo elas:

- Inverter níveis de cinza
- Converter o intervalo de cinza entre $[A, B]$
- Inverter os pixels das linhas pares
- Espelhar as linhas da metade superior na parte inferior
- Espelhar a imagem verticalmente

Todas foram feitas vetorialmente com a biblioteca NumPy. Sendo a primeira somente necessário realizar a subtração de 255 de todos valores da imagem. Na segunda, para converter o intervalo é necessário aplicar uma normalização na imagem sendo essa representada pela equação:

$$coef = \frac{A - B}{I_{max} - I_{min}}$$

$$I_{AB} = coef * (I - I_{min}) + A$$

Sendo A e B os limites, inferior e superior do novo intervalo e I a imagem. Por fim também é necessário converter a imagem para inteiro, afim de preservar os valores da mesma.

Para as operações restantes basta selecionar as linhas corretas e inverter as mesmas. Com a vetorização do numpy podemos selecionar as linhas pares com a notação de slicing, começando da linha 1 pois é o segundo elemento e com um step de dois, assim sempre serão selecionadas as linhas pares bastando somente inverter a outra dimensão com o slicing negativo. Para espelhar somente a parte inferior da imagem é necessário encontrar a metade da imagem antes, após isso basta acessar com a notação de *slicing*. Por fim o mais simples é espelhar a imagem verticalmente pois basta inverter a sua posição com a notação negativa com o slicing. Todas essas operações estão ilustradas na figura 3 abaixo.

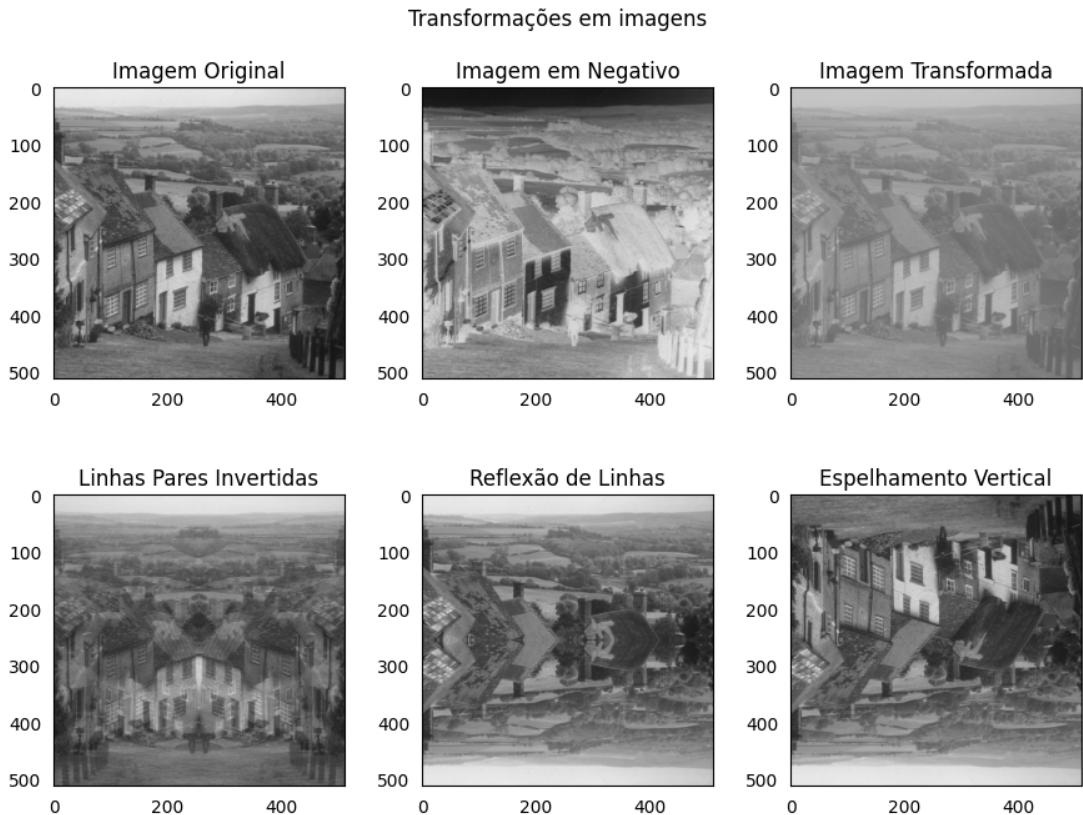


Figura 3: Diferentes transformações na mesma imagem

3.4 Imagens coloridas

Nesta atividade foi feito uma operação para alterar os canais RGB da imagem. Seguindo a equação abaixo:

$$R' = 0.393R + 0.769G + 0.189B$$

$$G' = 0.349R + 0.686G + 0.168B$$

$$B' = 0.272R + 0.534G + 0.131B$$

Onde os canais resultantes da nova imagem são obtidos pela média dos demais canais e ele mesmo. Computacionalmente foi utilizado o comando `numpy.dot` com uma matriz dos coeficientes.

Na figura 4, podemos ver a diferença na cor da imagem original, sendo que a primeira está mais "sem vida" isto é com as cores mais sutis se aproximando de tons de cinza.

Também é possível perceber que os canais de cor das imagens tem uma aparência mais brilhante na imagem transformada do que na original. No canal vermelho em específico é possível notar mais detalhes do que no canal não transformado. Porém no resultado final a imagem está menos saturada.

Para o segundo processamento uma imagem colorida foi convertida em um único canal de cor. Seguindo a média ponderada abaixo:

$$I = 0.2989R + 0.5870G + 0.1140B$$

Esse processamento converteu os níveis em uma escala próxima ao verde. Com essa tonalidade prevalecendo sobre os demais. O efeito está ilustrado com uma faixa de tonalidade rgb na figura 5, abaixo. E na imagem do fisiculturista figura 12, no final.

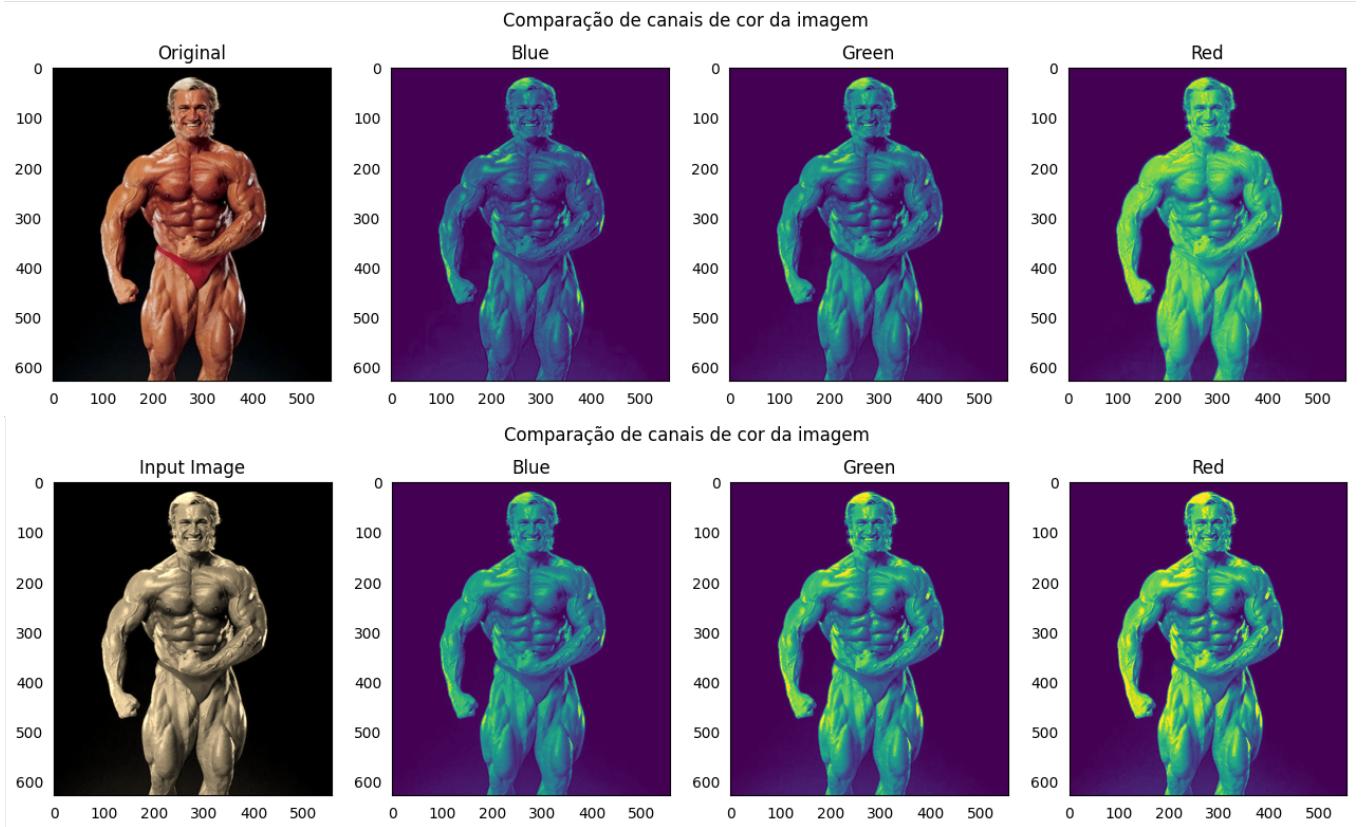


Figura 4: Imagem e seus canais de cores antes e depois da transformação

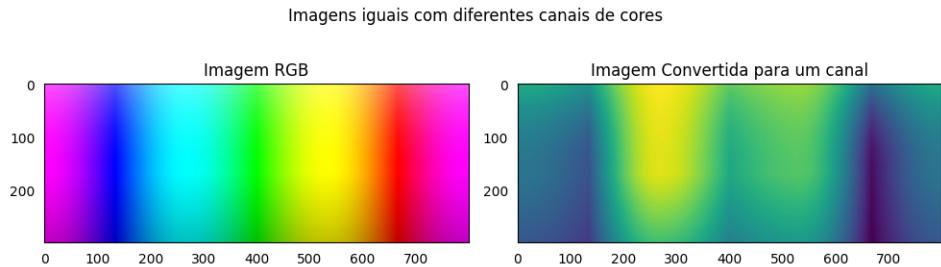


Figura 5: Transformação de três canais para um canal

3.5 Operações de Brilho

Ao elevar os pixels de uma imagem com uma potência crescente, aqueles com valor mais alto se destacam em relação aos outros, dado o crescimento exponencial. Entretanto esse valor escolhido deve ser pequeno pois caso contrário a imagem extrapolaria o intervalo de representação muito rapidamente. Neste caso utiliza-se o fator da exponenciação é o γ definido por:

$$\gamma' = \frac{1}{\gamma}$$

Assim a imagem acaba por possuir tons de branco mais fortes. Conforme ilustrado na figura 6 abaixo. Também é possível perceber que o brilho escala rapidamente a intensidade de branco da imagem. Com o fator em 1.5 a imagem está mais clara que a original porém com o fator de 2.5 a diferença entre a tonalidade das duas aumenta muito. Já com o fator em 10, a imagem original está totalmente esbranquiçada e quase que invisível perante a original.

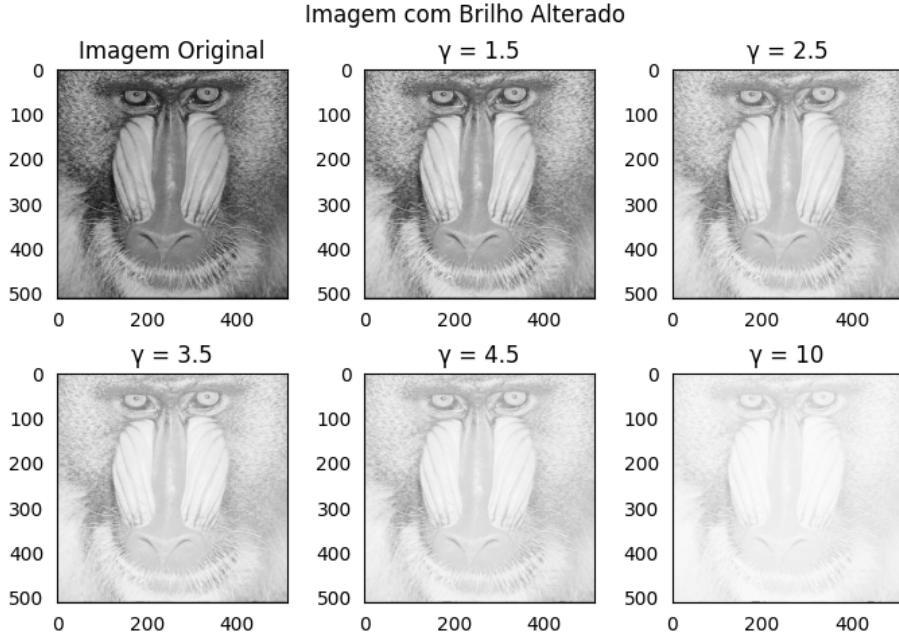


Figura 6: Imagem submetida a diferentes níveis de brilho

3.6 Planos de bit

Para obter os planos de bits das imagens basta realizar um bitwise and nos planos escolhidos. Uma vez que o resultado dessa operação é se o bit pertence ao plano ou não. Nesse caso ele será zero se não pertence e 1 caso pertença ao plano n .

Vale notar que ao extrair algum plano fora do intervalo de 0 a 7, neste caso, retornará uma imagem somente preta com todos os bits zero pois a imagem está sendo representada somente usando 8-bits.

Os planos extraídos podem ser visualizados na figura 7 abaixo, utilizando o Baboon como exemplo.

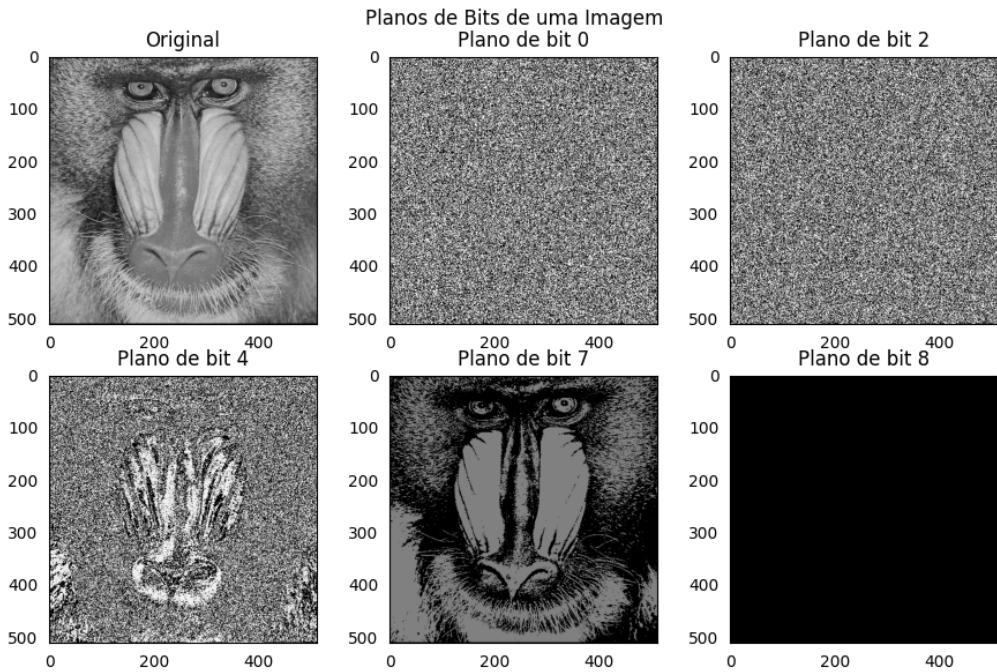


Figura 7: Plano de bits para o Baboon

3.7 Quantização de Imagens

Na quantização de níveis de cinza é necessário trocar a escala de representação da imagem. Isto é se representamos os níveis de cinza com outra escala. Para isso é necessário dividir a escala original pelo

número de níveis. Obtendo então a quantidade de valores a serem representados, seguidamente se aplica a divisão inteira na imagem com a quantidade de valores calculada previamente e então se multiplica por o mesmo valor.

Intuitivamente acredita-se que a imagem não será alterada pois é feito uma divisão e logo depois uma multiplicação. Porém como a divisão é inteira, os valores inferiores ao fator de correção são ignorados, pois é aplicado função piso ao resultado da divisão(divisão inteira). Alterando a representação dos níveis de cinza da imagem conforme é possível visualizar na imagem 8 abaixo:

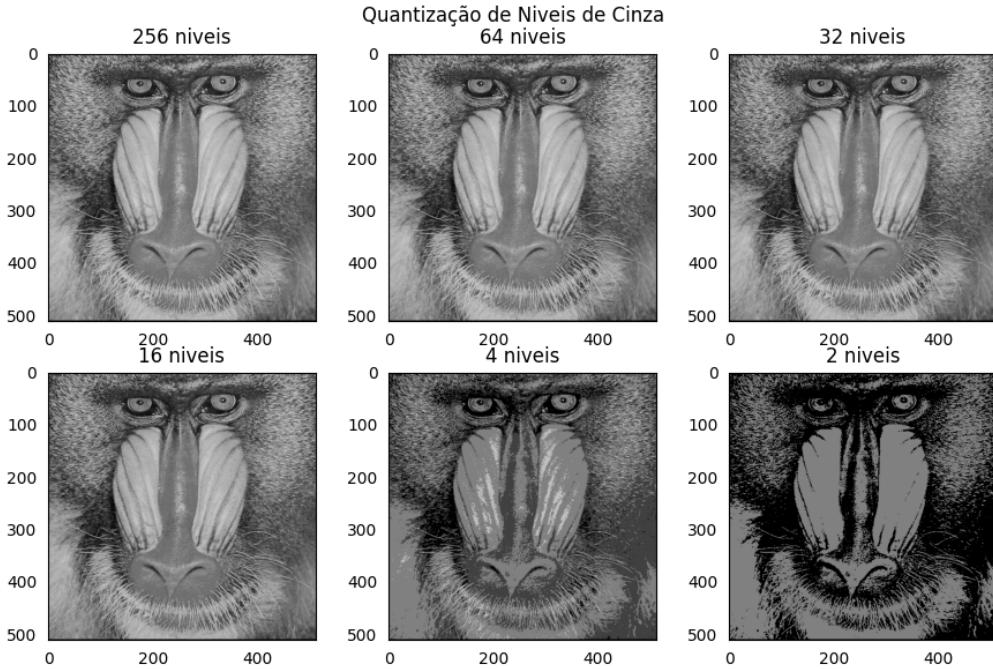


Figura 8: Quantização para o Baboon

3.8 Aplicando filtros

Para realizar o produto de hadamard entre o filtro e a imagem foi utilizado a função *filter2D* da biblioteca openCV. Ela possui uma particularidade que realiza somente uma correlação e não a convolução do filtro caso o kernel não tenha sido transposto. Para as aplicações deste trabalho o kernel foi invertido, isto é, a convolução foi feita.

Além disso a função utilizada, por padrão aplica a configuração de borda onde reflete os pixeis na ordem *gfedcb|abcdefg|h|gfedcba*, isto é, ele copia o elemento que está a uma posição anterior ao elemento da borda. Refletindo a matriz exceto pelo pixel na borda.

A fim de comparar ambas técnicas de borda também foi utilizado a configuração de isolamento, isto é, a função ignora os elementos de fora do range do kernel.

Com a imagem abaixo é possível notar que o filtro h9 é aplicado um blur na imagem e deixa a mesma menos nítida. Nesta imagem foi aplicado a borda de reflexão.

O filtro h1, provoca um efeito na imagem como se ela tivesse sido coberta em cinza pois é possível visualizar o fisiculturista porém menos detalhado.

O filtro h2 é um construído como um filtro gaussiano assim ele suaviza a imagem. Deixando a imagem resultante mais suave porém considerando os pixeis centrais com maior relevância. Diferentemente do filtro h6, que realiza a média dos valores sem nenhum peso a posição do pixel na imagem original. Ainda que a diferença entre eles nos exemplos é pequena, a imagem resultante da aplicação do filtro h2 parece mais "borrada" do que com o filtro de media, h6. Principalmente nas regiões de fronteiras entre cores.

Tanto o h3, h4 e h10. Capturam as bordas da imagem a variar da intensidade e do foco. O h3 parece capturar melhor linhas verticais e o h4 horizontais. Assim para combinar os dois foi feito um filtro novo chamado de *h3h4_{mixed}* sendo a raiz da soma dos quadrados dos filtros. Cada filtro parece que cancelou o outro, retornando uma imagem similar a original.

Ambos filtros h5 e h7 resultam em imagens semelhantes. Os dois parecem colocar uma camada cinza sobre a imagem. Porém o h7 torna a imagem menos nítida pois é possível perceber que alguns detalhes

estão mais difíceis de serem vistos. Além disso as regiões de transição no h5 estão mais destacadas, o que faz sentido visto que é um filtro passa alta.

O filtro h8, tem um resultado parecido o h4 onde é possível perceber o contorno do fisiculturista. Porém com uma nitidez menor.



Figura 9: Filtros aplicados para o Fisiculturista, com borda refletida e convolução

Ao comparar as figuras 10 e 11 é possível identificar algumas diferenças entre refletir a borda e ignorar a mesma. Na figura correspondente ao filtro h1, é possível notar que no caso onde a borda foi ignorada a imagem está com um tom de cinza mais forte. Porém exceto esse tom de cinza mais escuro não foi possível identificar diferenças significativas entre as estratégias de borda aplicadas pela biblioteca.

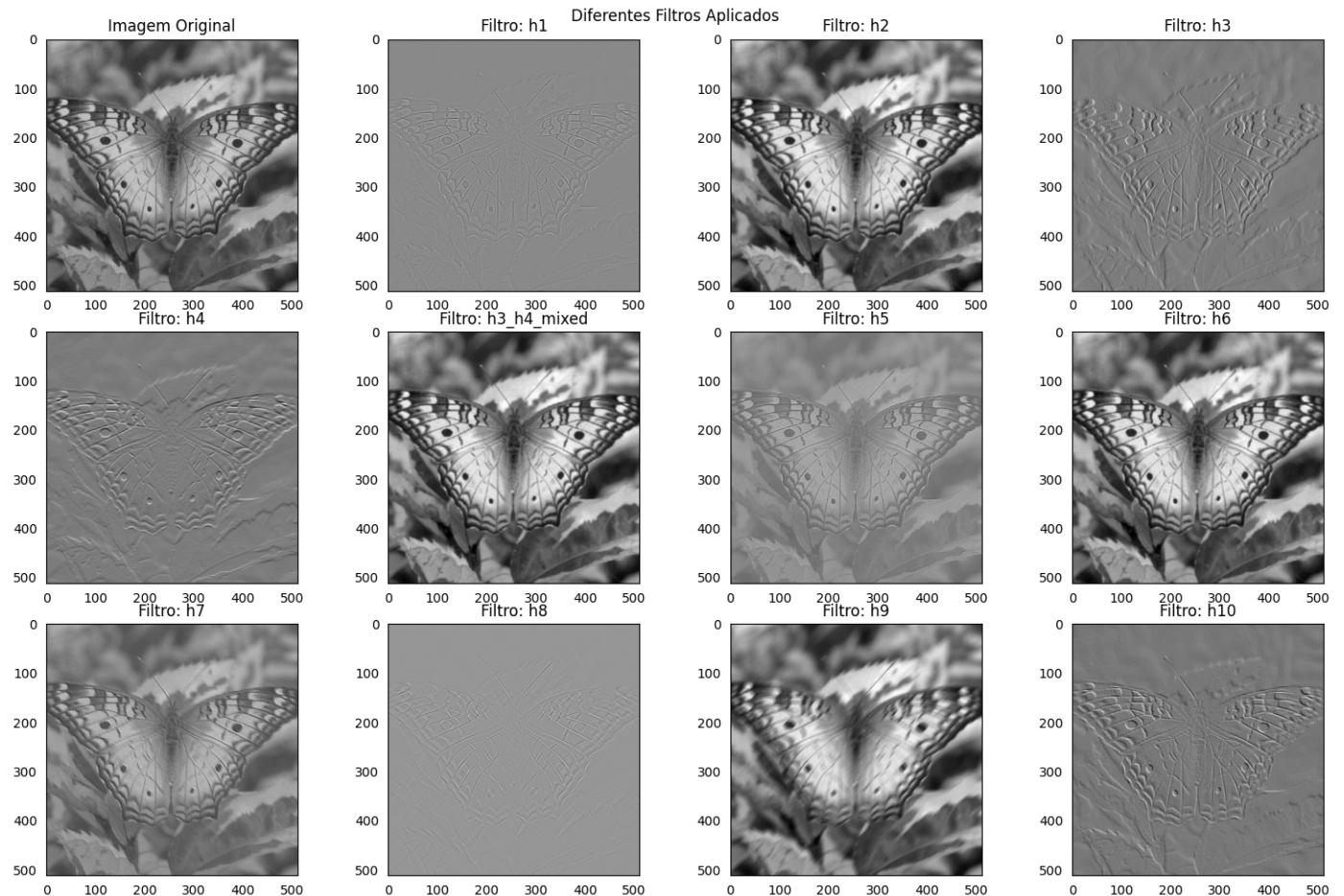


Figura 10: Filtros aplicados para a borboleta, com borda refletida e convolução

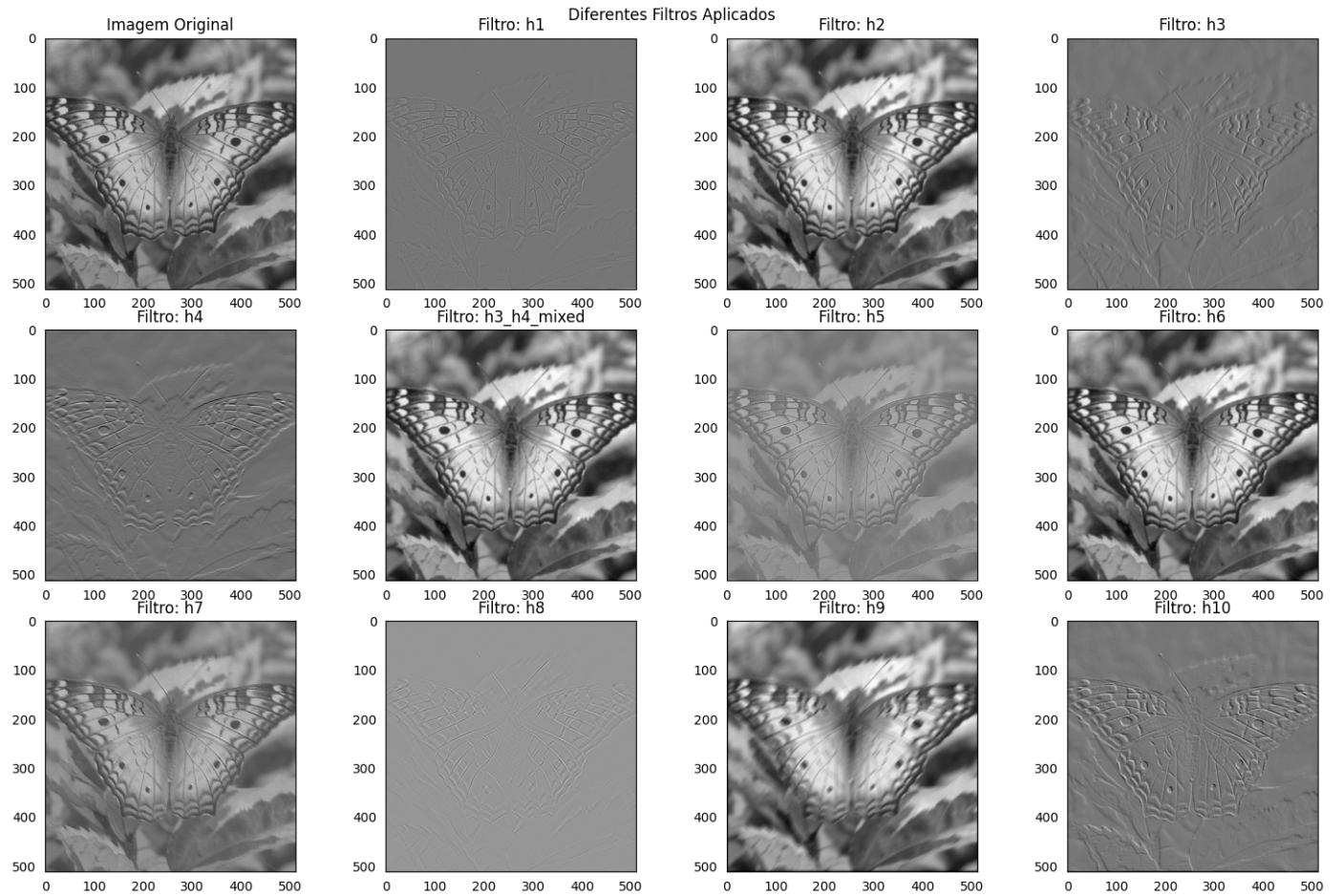


Figura 11: Filtros aplicados para a borboleta, com borda ignorada e convolução

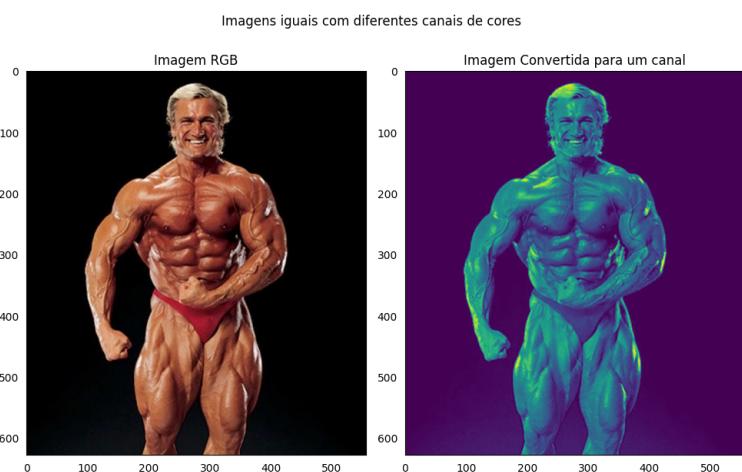


Figura 12: Imagem transformada em um canal