

Igor Engelmann Batista

RA : 260511

Introdução ao Processamento de Imagem Digital (MC920)

Trabalho 1

Considerações Gerais:	2
Questão 1:	2
Questão 2:	4
Questão 3	6
Questão 4	6
Questão 5	7
Questão 6	8
Questão 7	9
Questão 8	10
Questão 9	12
Questão 10	13

Considerações Gerais:

Para diminuir o peso dos arquivos enviados, decidi limpar as saídas do notebook que possui o código, elas serão geradas novamente ao executar-se todas as células. Da mesma forma, esse relatório contém apenas prints das saídas do notebook, diminuindo o peso desse documento, mas mantendo a possibilidade de uma inspeção mais aprofundada, olhando-se diretamente para a saída produzida pelo código. Em contrapartida, decidi por enviar as imagens de saída em .png diretamente no arquivo .zip (mesmo sabendo que elas serão sobreescritas ao se executar todas as células do notebook) pois, além de ser o comando do trabalho, cabe como via de segurança caso haja algum problema em algum aspecto da execução.

Alguns processos foram realizados em todas as questões. Esses, serão abordados nesta sessão para evitarmos repetição desnecessária ao longo do texto. Caso algum deles tenha sido modificado na resolução de alguma questão específica, tal mudança será discutida na sessão correspondente.

Usei o OpenCV para a maioria das operações. Para carregar e trabalhar corretamente as imagens coloridas, foi necessário converter de BGR (ordem eletromagnética, e padrão do opencv) para RGB. No caso de imagens em escala de cinza, isso não foi necessário. Além disso, no início do processo converti o tipo de dado das imagens para arrays Numpy, com tipo *uint16* ou *float32* (dependendo das operações subsequentes), para que houvesse maior liberdade de valores durante as transformações, resultando em maior precisão, e retornando para *uint8* apenas no final, após ter feito a normalização para [0~255] utilizando a função *cv2.normalize()* com a normalização *cv2.NORM_MINMAX*, que faz uma transformação linear, a qual acredito que mantenha a relação entre os valores dos pixels, tentando manter a fidelidade dos valores da imagem.

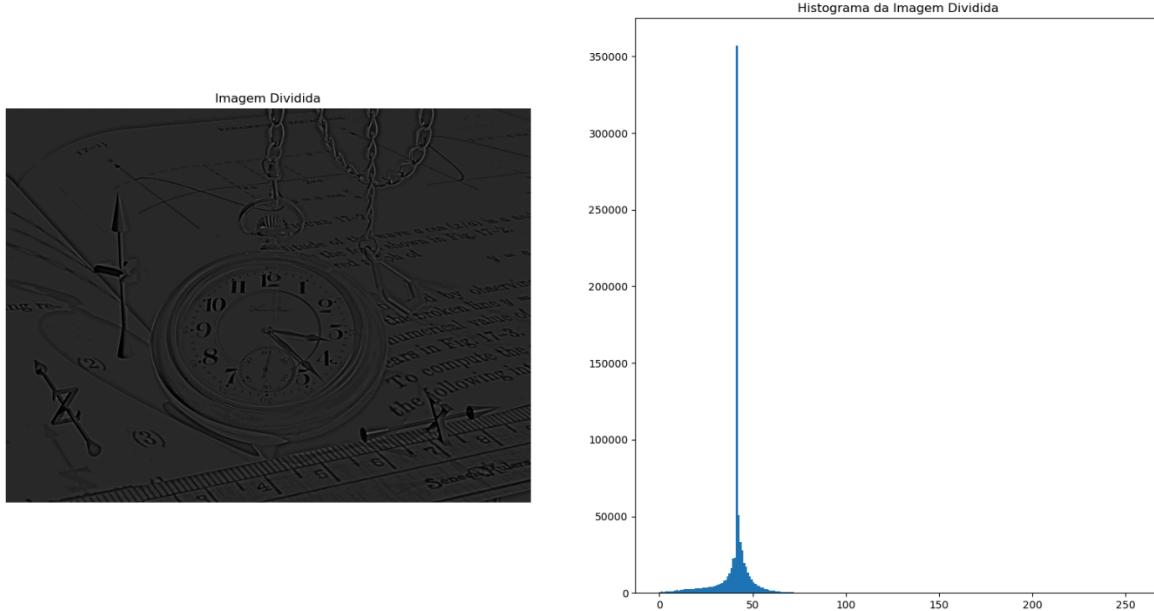
Nos momentos em que se mostrou necessário realizar operações que produzem efeitos de borda (a aplicação de um filtro, por exemplo), decidi pela utilização da opção *cv2.BORDER_REFLECT*, pois dentre as possibilidades, acredito que essa mantenha com mais precisão o escopo de valores nas bordas e, por inspeção visual, gera uma saída mais natural.

Outro ponto recorrente, é a utilização do *matplotlib.pyplot.imshow()* para apresentar as imagens, entretanto, por padrão essa função realiza uma transformação linear para aumentar o contraste e facilitar a visualização. Dado o escopo desse trabalho, não é recomendável esse processo, pois em uma imagem com níveis de cinza entre [100~200] seria mostrado alguns pixels totalmente pretos e/ou brancos, entretanto, dada a faixa de valores, isso é impossível pois o menor valor é 100, e não 0, por exemplo. Para lidar com essa situação, a própria função tem como parâmetros “*vmin* e *vmax*” que substituem os valores mínimos e máximos da imagem, respectivamente, no cálculo dessa normalização, assim, preenchendo-os com os valores 0 e 255 (que já são os valores de destino da transformação linear) esse procedimento é anulado, e podemos visualizar os valores reais da imagem. Esse processo foi feito em todas as chamadas de exibição da imagem.

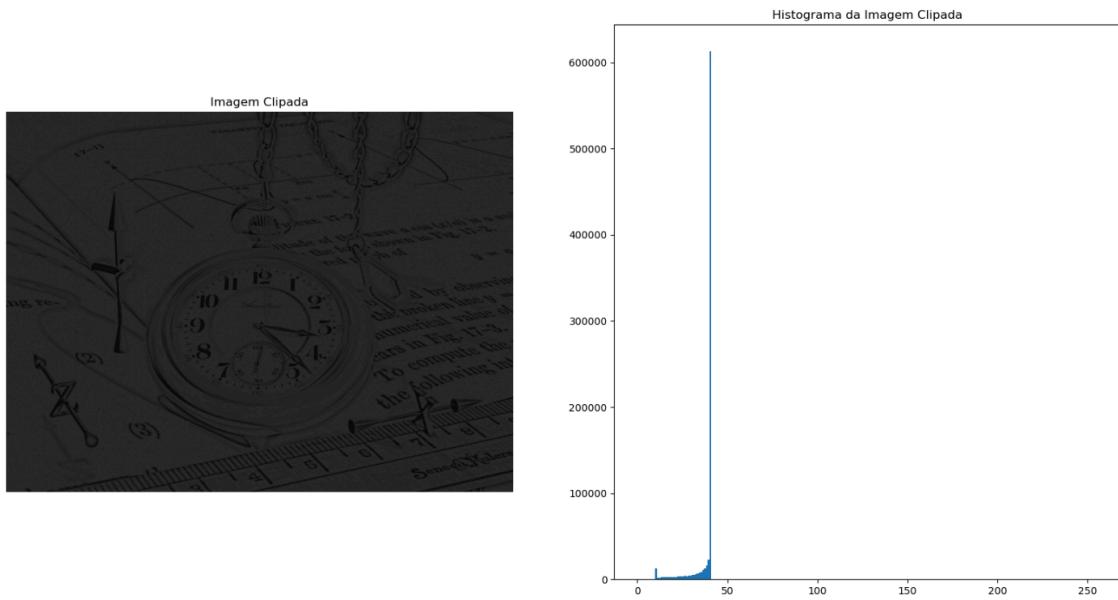
Questão 1:

A questão 1 solicita que seja gerado um efeito de “Esboço a lápis”:

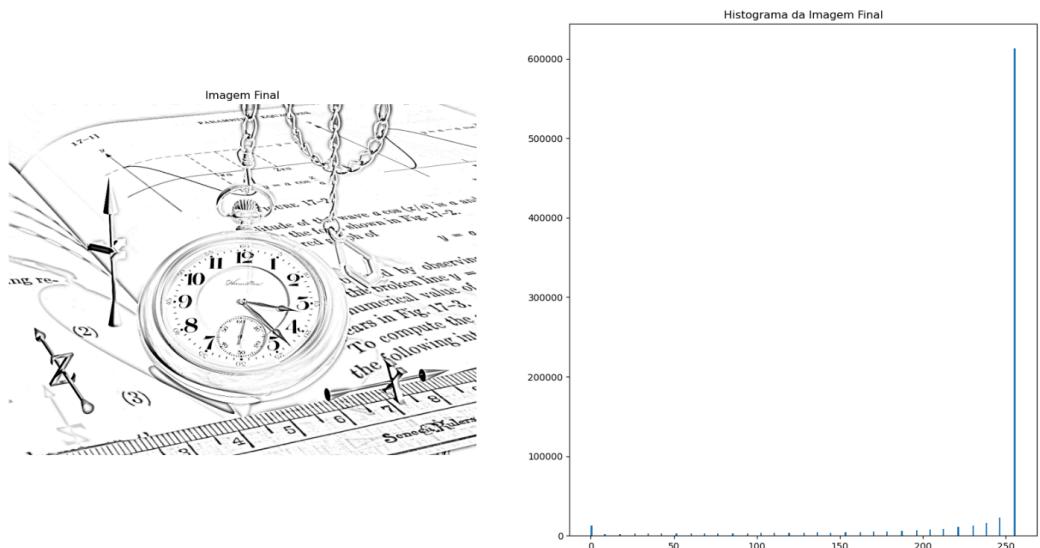
Após realizar os procedimentos indicados – a saber: (i) converter a imagem colorida para níveis de cinza, (ii) aplicar um filtro de desfoque gaussiano, (por exemplo, com uma máscara de 21×21 pixels) para suavizar os detalhes da imagem, (iii) dividir a imagem em tons de cinza pela versão desfocada para realçar os contornos (aqui se fez necessário somar 1 aos valores da imagem que serviria como divisor, para evitar uma divisão por 0, decidi também somar 1 em todos os valores da imagem a ser dividida, para tentar manter minimamente a relação entre os valores de pixels) – obtive a seguinte resposta:



Visivelmente, a imagem não parece uma representação feita a lápis. Suponho que isso se dê por dois fatores, o primeiro, que a média geral dos valores ficou muito baixa, como é possível ver pelo histograma acima. Além disso, é característico do desenho à lápis, que o fundo branco seja maioria da imagem, pois isso facilita o trabalho do desenhista. Dessa forma, imaginei que realizar o processo de “clip” (utilizando a função `np.clip()`) da imagem para os valores [10~40] fosse uma boa saída. Como é possível ver no histograma, 40 é logo abaixo do pico na contagem dos valores, diminuindo assim o espectro de tons de branco diferente e maximizando as regiões com apenas o tom original do papel. Por outro lado, definindo 10 como intensidade mínima, ainda temos uma boa faixa de valores para diferentes níveis de preto, o que é mais razoável para um desenho a lápis. Dessa forma, após realizar a “clipagem” obtive a imagem a seguir:



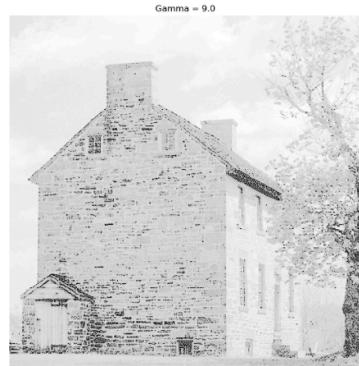
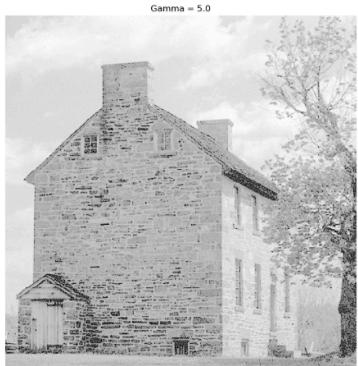
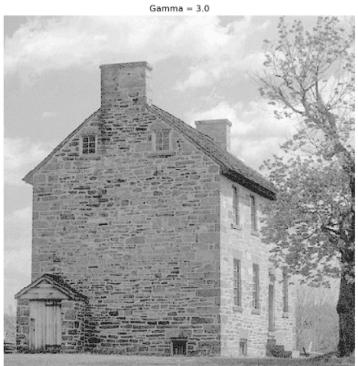
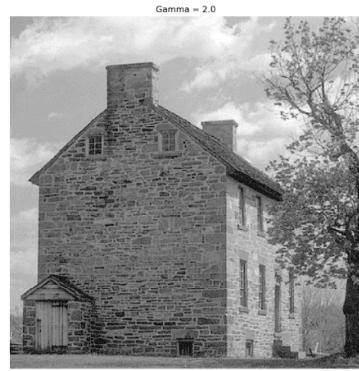
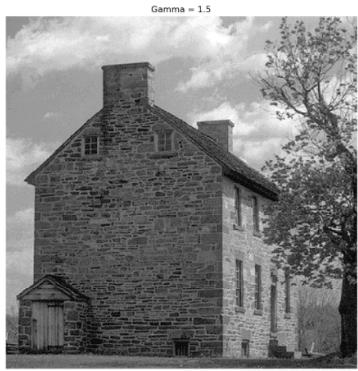
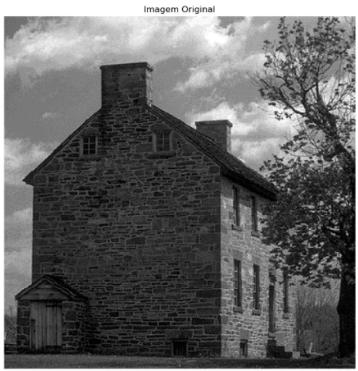
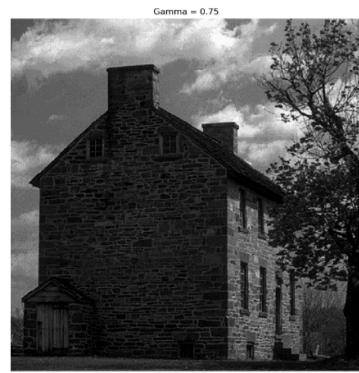
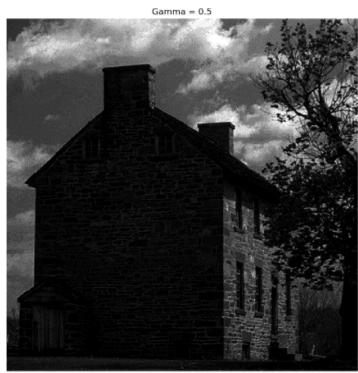
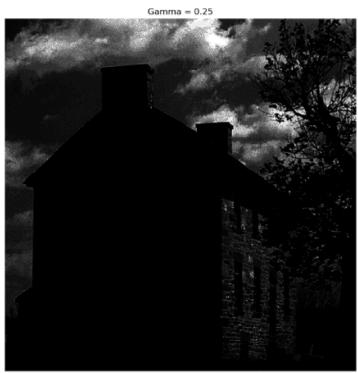
Embora seja visível que a imagem ficou mais regular, ainda está muito escura – como pode ser visto no histograma. Então, bastou fazer a normalização para o intervalo [0~255], e obter o resultado esperado:



Questão 2:

A questão 2 solicita que seja utilizada uma equação exponencial para alterar o brilho da imagem.

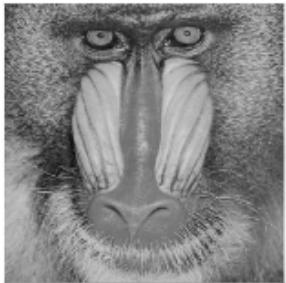
Nessa questão, decidi usar diversos valores diferentes para y , pois além de explorar o efeito aplicado pela fórmula exponencial tanto para escurecer quanto para clarear a imagem, acredito que essa abordagem gera uma visualização mais agradável. Percebi também, que os valores menores que 1, escurecem a imagem bem rapidamente, já para valores maiores que 1, as diferenças são mais suaves, precisando de números maiores para grandes efeitos, todos os resultados podem ser vistos nas imagens abaixo.



É válido notar também, como característica dessa forma de mudança de brilho, que a variação dos valores é relativa, por exemplo, pixels muito escuros, tendem a permanecer muito escuros, mesmo para valores elevados de gamma, e vice versa para pixels muito claros. Esse é um efeito interessante que mostra a diferença entre fazer o ajuste de brilho exponencialmente, ou “somando e subtraindo constantes”. Ressaltando assim que a forma exponencial se mostra mais apropriada para realizar essa alteração na imagem.

Questão 3

A questão solicita que seja montado um mosaico, da seguinte forma:



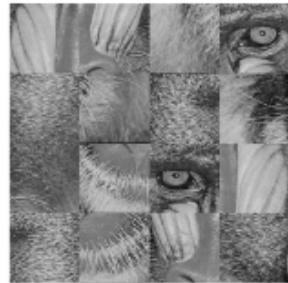
(a) imagem

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

(b) ordem dos blocos

6	11	13	3
8	16	1	9
12	14	2	7
4	15	10	5

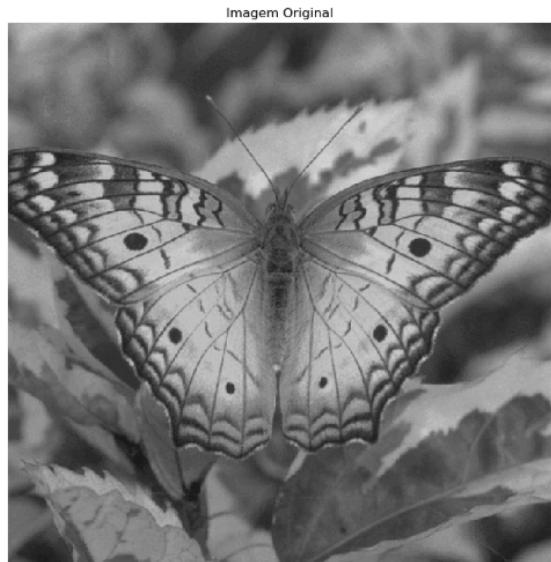
(c) nova ordem dos blocos



(d) mosaico

Nessa questão, decidi por criar uma lista com o destino dos blocos, ou seja:

"destino[x-1] = Nova posição + 1". Ex: *"destino[4] = 15"* significa que o bloco da posição 5 na imagem acima, irá para a posição 16. Em seguida, fiz mais duas listas, contendo os índices limites de cada bloco, tanto de destino, quanto de origem. Decidi fazer dessa forma a indexação, pois permitiria que o passo de efetivamente remontar a imagem, fosse feito de maneira semi-vetorizada. Ou seja, um loop de 16 passos, mas que a mudança das linhas é feita de forma vetorializada para cada bloco. Para esse caso, escolhi a imagem da borboleta, pois acho que possui um padrão facilmente identificável.



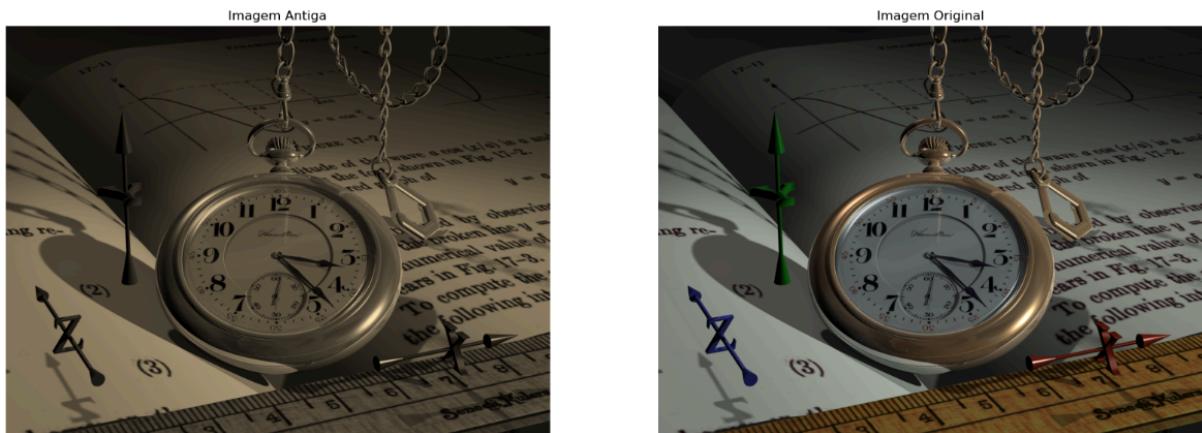
Questão 4

A questão 4 solicita que seja realizado uma transformação linear em todos os pixels de uma imagem RGB. A matriz a ser utilizada está mostrada abaixo:

$$\begin{bmatrix} 0.393 & 0.769 & 0.189 \\ 0.349 & 0.686 & 0.168 \\ 0.272 & 0.534 & 0.131 \end{bmatrix}$$

Nessa questão, decidi utilizar a função *"cv2.transform()"* do OpenCV, pois ela faz exatamente o que foi proposto (multiplicar cada pixel pela matriz dada, resultando numa

transformação linear), e como é implementada pela biblioteca, faz isso de forma otimizada. Dessa forma, obtive o seguinte resultado:



O efeito dessa transformação se mostrou coerente com o esperado, dando uma aparência de imagem envelhecida, algo parecido com o “Efeito Sépia”.

Questão 5

A questão 5 solicita dois processos. O primeiro é alterar os valores dos pixels segundo a seguinte expressão:

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

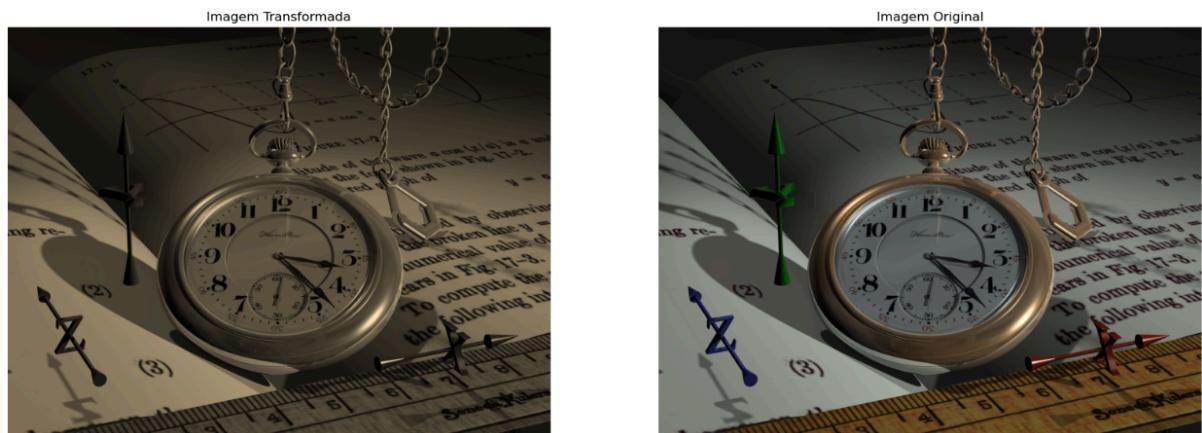
$$G' = 0.349R + 0.686G + 0.168B$$

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

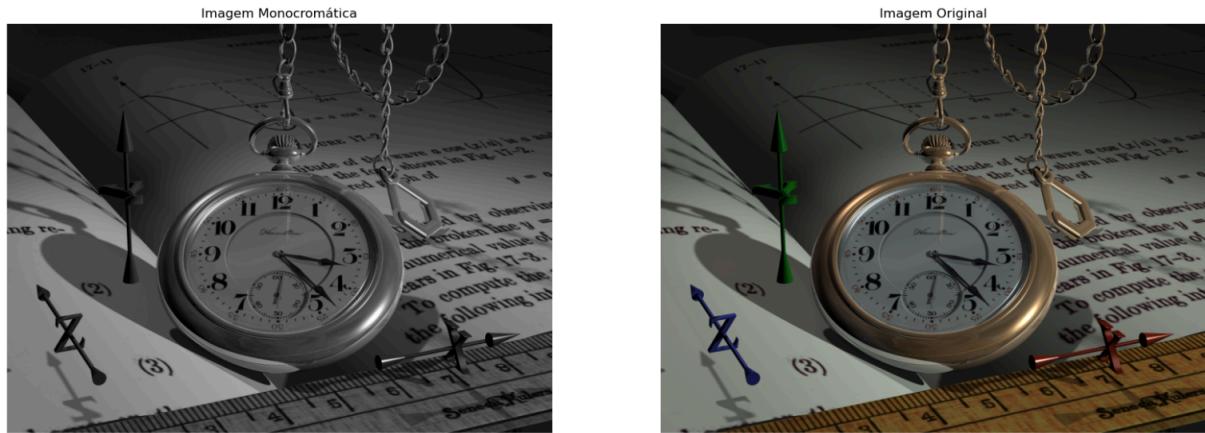
E o segundo, tornar a imagem monocromática, da seguinte forma:

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

Para a primeira parte dessa questão, procurei vetorizar a transformação em cada canal de cor, e, o resultado foi semelhante à transformação feita na questão 4, - o que é esperado, visto que são os mesmos valores- ou seja, a aparência de foto antiga, como é possível ver abaixo.



Na segunda parte da questão, foi possível vetorizar a operação inteira, resultando numa imagem monocromática, como é possível observar abaixo:

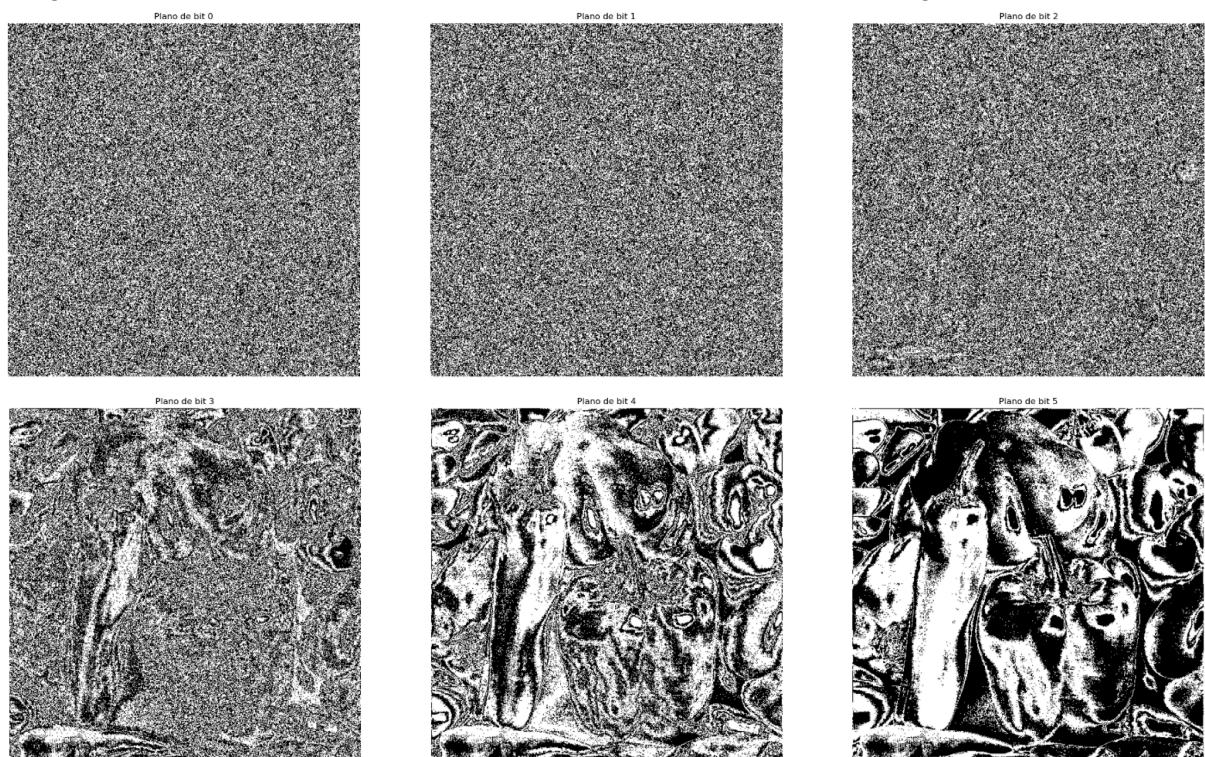


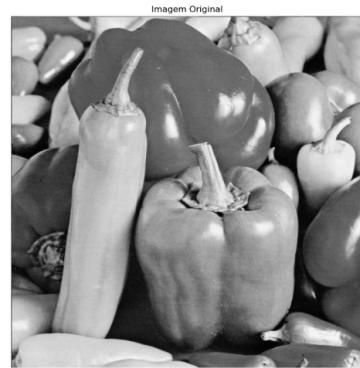
Cabe ressaltar que a operação de transformar em uma imagem Monocromática gerou intensidades de cinza diferentes para cores diferentes, mesmo que tenham a valores globais parecidos, como exemplificado nos vetores que aparecem na imagem.

Questão 6

A questão 6 solicita que a imagem seja separada nos seus respectivos planos de bits, uma espécie de fatiamento na “profundidade da cor”.

Para essa questão, decidi criar um vetor com as máscaras que seriam utilizadas para extrair os bits de cada plano (`00000001`, `00000010`, ..., `10000000`), e após fazer o “and bitwise” da imagem original com cada máscara, fiz um deslocamento para a direita, transformando o bit extraído no valor menos significativo (ou seja, a imagem seria composta apenas por 0's e 1's) e posteriormente multiplicando esse valor por 255, para criar a imagem em preto e branco correspondente. Dessa forma, obtive os seguintes resultados:

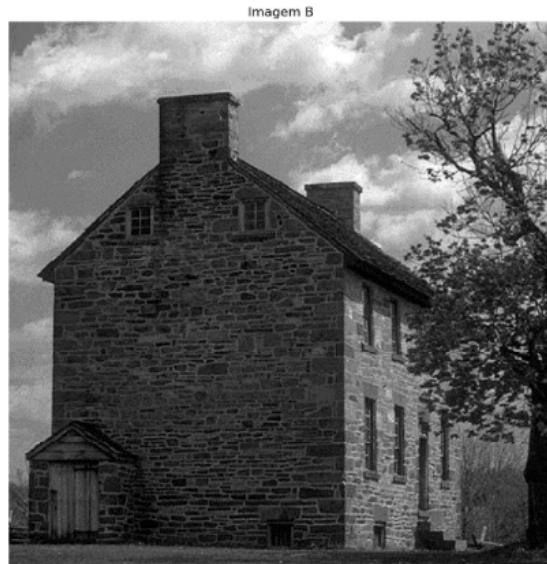
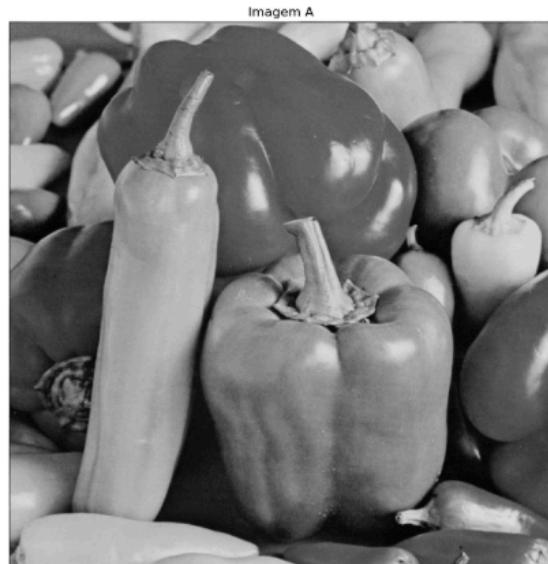




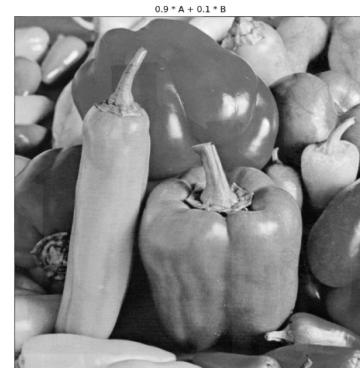
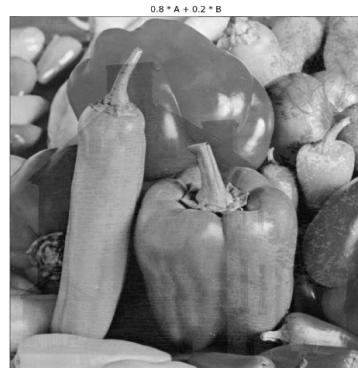
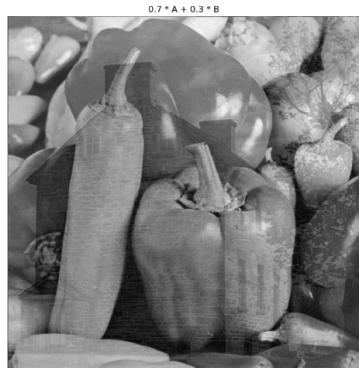
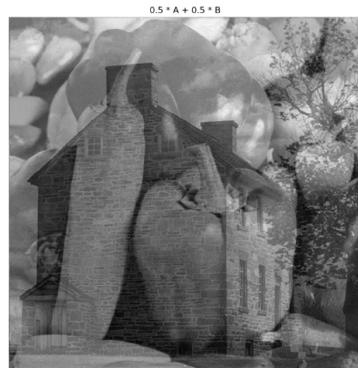
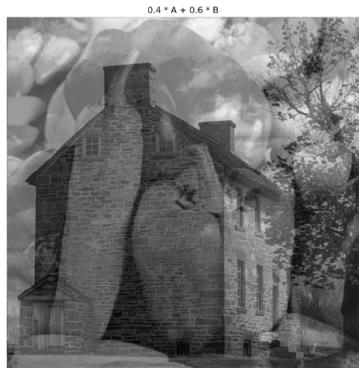
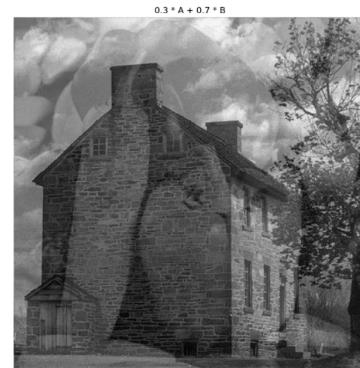
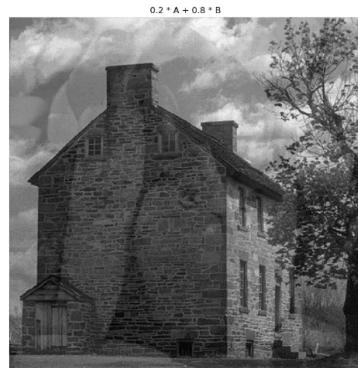
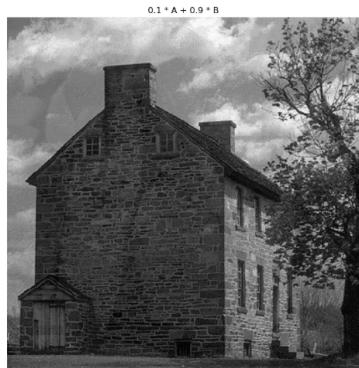
Embora seja possível notar alguns padrões incipientes a partir do plano de bit 2, (o terceiro menos significativo) ainda é coerente dizer que para os 3 planos de bits menos significativos, temos apenas ruído na imagem. Da mesma forma, é interessante visualizar a relação entre os planos 6 e 7 (os dois bits mais significativos) pois já um aparente efeito de negativo entre eles, quebrado apenas pelos pontos extremos da imagem, como reflexos (extremo claro) ou sombras densas (extremos escuros). Acredito que explorar uma transformação utilizando a relação entre esses dois planos, permita encontrar rapidamente pontos da imagem com essas características.

Questão 7

A questão 7 solicita que seja feito uma combinação de imagens, por meio de uma média ponderada de seus níveis de cinza. Escolhi as seguintes imagens como base:



Nessa questão, escolhi variar a porcentagem de cada imagem de 10% em 10%, acredito que essa variação mais contínua permita perceber em detalhes qual o efeito gerado por essa operação, além de tornar a visualização mais agradável. Resultados:



Percebi que, dependendo da porcentagem utilizada, a sobreposição de imagens gera um efeito de textura, ou até mesmo parecendo que uma das imagens faz parte da outra, como uma pintura na parede, por exemplo.

Questão 8

A questão 8 solicita que sejam feitas diversas operações na imagem, alterando valores, ou a disposição das linhas, a saber:

- (a) partindo de uma imagem monocromática:
- (b) obter o negativo da imagem, ou seja, o nível de cinza 0 será convertido para 255, o nível 1 para 254 e assim por diante
- (c) converter o intervalo de intensidades para [100, 200]
- (d) inverter os valores dos pixels das linhas pares da imagem. Ou seja, os valores dos pixels da linha 0 serão posicionados da direita para a esquerda, os valores dos pixels da linha 2 serão posicionados da direita para a esquerda e assim por diante
- (e) espelhar as linhas da metade superior da imagem na parte inferior da imagem e

(f) aplicar um espelhamento vertical na imagem levando-se em conta todas as linhas da imagem.

Nessa questão, fiz uso da função `flip()` do OpenCV, – a qual, dependendo do argumento passado, pode fazer a inversão no eixo das linhas (horizontal), ou das colunas (vertical) –, e principalmente, de slices nos vetores. Dessa forma, obtive o seguinte:



Acredito que cabe destaque às transformações dos itens (c), (d), e (e).

No item (c), nota-se que a imagem ficou de forma geral mais clara, e perdeu um pouco do contraste, tornando-se mais “apagada”. Isso se deve ao fato de que o nível de intensidade mínimo da imagem agora é 100, e não 0, então não temos mais cores tão escuras quanto na imagem original. Além disso, agora há apenas 100 níveis diferentes de cinza, e não os originais 256, resultando claramente numa diminuição do contraste geral da imagem.

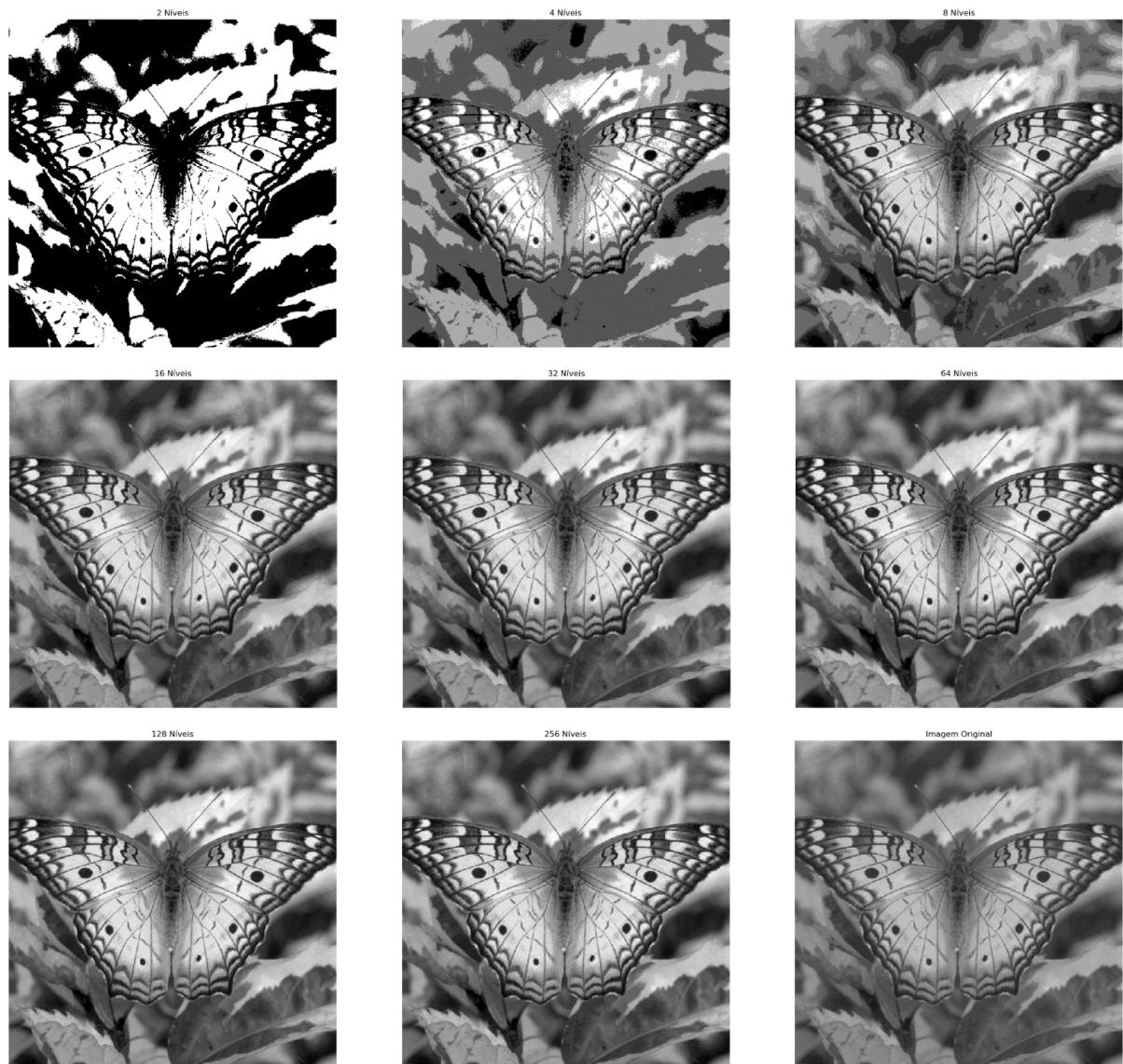
No item (d), inverter apenas metade das linhas, mas de forma alternada, gerou uma espécie de espelhamento, entretanto, toda a informação da imagem continua lá, gerando uma espécie de sobreposição, onde partes mais regulares e centralizadas da imagem têm maior destaque. Importante notar que o efeito de simetria gerado é apenas visual, pois cada linha ainda mantém seus valores originais, não havendo um espelhamento de fato.

No item (e), também foi mexido em apenas metade das linhas, entretanto, elas são contíguas, e foi feito de fato um espelhamento (perdendo toda a informação que havia na parte de baixo da imagem), da forma como foi feito, gerou um efeito conhecido como “espelho d’água” que possui um potencial artístico muito interessante, dependendo da forma como for utilizado.

Questão 9

A questão 9 solicita que uma imagem monocromática seja quantizada, ou seja, diminuir o número de bits de profundidade de cada imagem.

Para cumprir a solicitação da questão, decidi por implementar um vetor com os limites máximos que cada número de bits permite para a escala de cinza (1, 3, 7 ... 127, 255) e depois normalizar as imagens para uma faixa de valores tendo como valor mínimo 0 e máximo o limite presente no vetor. Decidi por essa abordagem, pois essa operação realiza uma transformação linear das intensidades, que, embora caindo pela metade o número possível de valores, ainda buscará manter a mesma relação entre eles. Após esse processo, normalizei novamente os valores para o intervalo [0~255], para que melhorar a visualização (da mesma forma que seria feita por um visualizador padrão). Resultados:



Ao analisar os resultados, percebi dois pontos principais. O primeiro, é que a partir de 16 níveis de cor, não consegui mais perceber nenhuma alteração significativa na imagem, algo que pode ser relacionado à questão 6, onde percebemos que os 4 planos de bits menos significativos carregam majoritariamente, ruído. O segundo ponto, é que a imagem referente aos 256 níveis de cor deveria ser idêntica à imagem original, entretanto, ela aparenta estar com um contraste mais elevado. Acredito que isso seja devido à

normalização para [0~255], que forçou a amplitude máxima dos valores dos pixels, resultando no efeito citado.

Questão 10

A questão 10 solicita que sejam aplicados vários filtros em uma imagem monocromática. E também que seja apresentada a resposta combinada dos filtros h_3 e h_4 . Os filtros aplicados são os seguintes:

$$h_1 = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

$$h_2 = \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

$$h_3 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$h_4 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$h_5 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$h_6 = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$h_7 = \begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix}$$

$$h_8 = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$

$$h_9 = \frac{1}{9} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

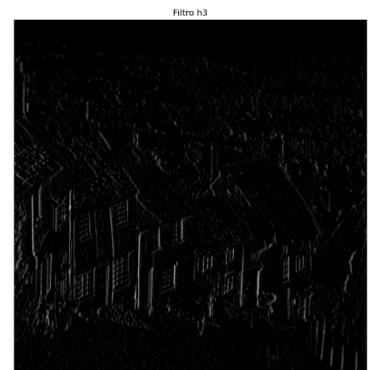
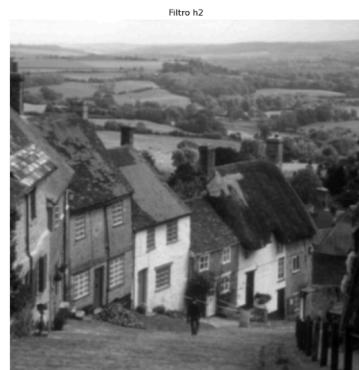
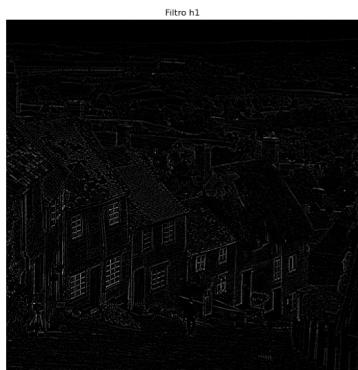
$$h_{10} = \frac{1}{8} \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & 2 & 2 & 2 & -1 \\ -1 & 2 & 8 & 2 & -1 \\ -1 & 2 & 2 & 2 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

$$h_{11} = \begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

E usei a seguinte imagem como origem:



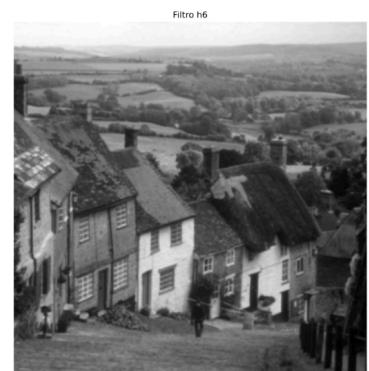
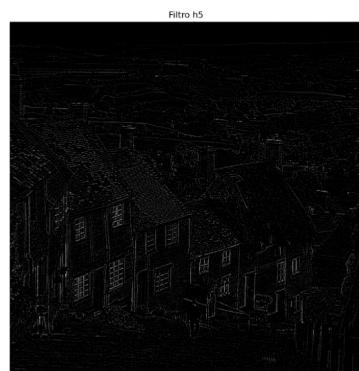
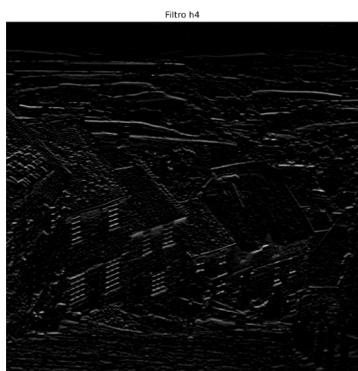
Nessa questão, decidi utilizar a função `cv2.filter2D()` que realiza a convolução da imagem com o filtro fornecido. Dentre outros parâmetros, indiquei pela manutenção da mesma profundidade da imagem original. Escolhi fazer através dessa função, pelos mesmos motivos que escolhi a `cv2.transform()` e a `cv2.normalize()` nos locais já apresentados, pois a função cumpre o objetivo proposto, e por ser implementada via biblioteca, é esperado que apresente grande otimização. Obtive os seguintes resultados:



Filtro h1: Como esperado de um filtro passa-alta, ressaltou as bordas da imagem em múltiplas direções, embora tenha feito isso de maneira suave, provavelmente devido ao fato de ser uma matriz 5x5.

Filtro h2: A filtragem provocou um desfoque geral na imagem, conforme esperado de um filtro gaussiano passa-baixas.

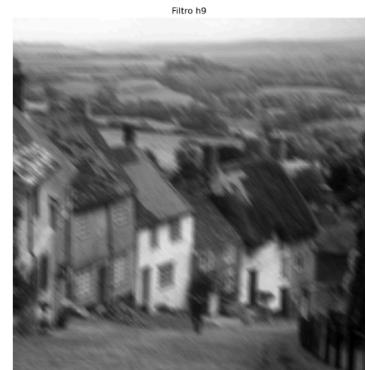
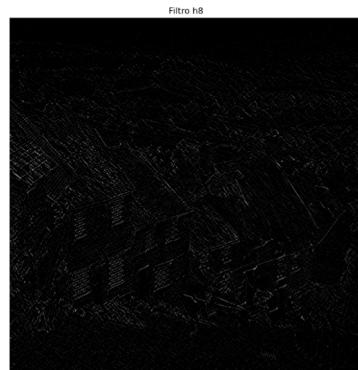
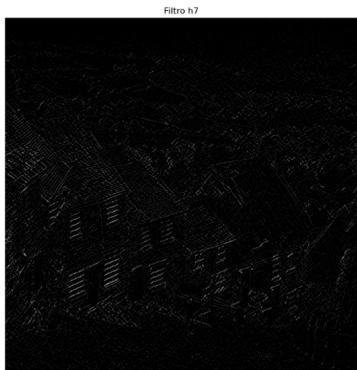
Filtro h3: O filtro realçou as bordas verticais da imagem, bordas na diagonal também foram ressaltadas, mas com menor intensidade. É um exemplo da atuação de um filtro passa-alta.



Filtro h4: Complementarmente ao filtro h3, essa filtragem realçou as bordas horizontais da imagem, algumas diagonais também foram ressaltadas. Também é um exemplo da atuação de um filtro passa-alta.

Filtro h5: Assim como o filtro h1, destacou bordas em várias direções e, por ser uma matriz menor, de forma um pouco mais agressiva, destacando também bordas mais tênuas.

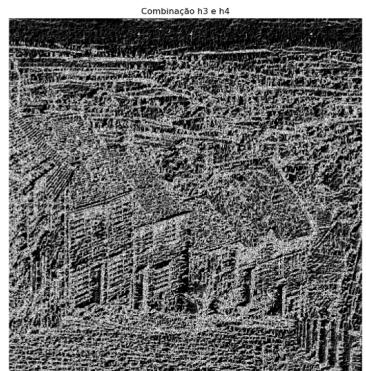
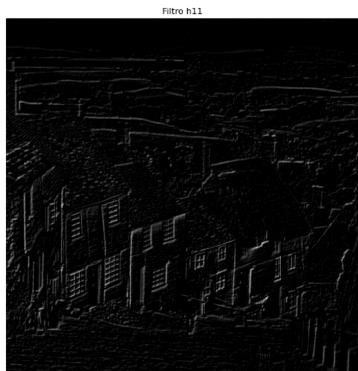
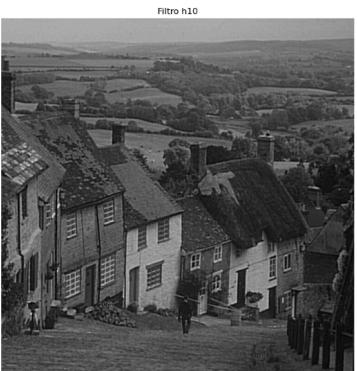
Filtro h6: O filtro provocou um pequeno desfoque na imagem, conforme esperado de um filtro passa-baixa de média.



Filtro h7: O filtro também destacou algumas bordas, principalmente as que seguem a diagonal secundária da imagem.

Filtro h8: Complementarmente ao filtro h7, também destacou bordas na diagonal, mas nesse caso, principalmente as que seguem a diagonal principal da imagem.

Filtro h9: Esse filtro gerou um “arraste” no sentido da diagonal principal da imagem, como se a foto fosse batida com a câmera em movimento nessa direção.



Filtro h10: O filtro aparentemente tornou a imagem um pouco mais nítida para grandes objetos, entretanto, perdeu detalhamento e em alguns pontos, parece estar um pouco mais ruidosa.

Filtro h11: O filtro destacou bordas da imagem, principalmente as pertencentes ao canto superior esquerdo, embora algumas bordas horizontais e verticais também aparecem.

Combinação das respostas aos filtros h3 e h4: Nota-se que foram destacadas diversas bordas, em praticamente todas as direções, por mais tênuas que sejam na imagem original.

Entretanto, a imagem resultante se torna tão poluída, que o número de objetos identificáveis se torna bastante diminuto.