Trabalho 1

MO443/MC920 - Introdução ao Processamento de Imagem Digital

Lucas de Oliveira Silva - 220715

1 Introdução

Neste trabalho o objetivo foi aplicar filtros simples a imagens monocromáticas por meio da operação de convolução. A filtragem foi feita no domínio espacial usando a função *convolve* do scipy. A solução foi feita com Python 3.8.5 em um único arquivo .py. Foram utilizadas as seguintes bibliotecas: numpy (1.19.1), scikit-image (0.16.2) e scipy (1.5.2).

2 Como executar

A implementação está no *script* trabalho1.py, as imagens de teste na raiz e as saídas obtidas na pasta output. O programa precisa necessariamente de uma imagem de entrada (-i img) em formato png, o resto dos parâmetros são opcionais e alguns possuem valores padrões. Os parâmetros opcionais são os seguintes:

- -o Indica a imagem de saída. O padrão é out.png.
- -num Um inteiro que indica qual máscara será usada dentre as listadas no enunciado.
- -m Indica como as bordas deveram ser tratadas conforme esta lista. O padrão é "nearest"
- -s Indica o formato da máscara customizada caso esse modo for utilizado. Ex: -s 2,2
- -k Lista de valores que representam a máscara. Ex: -k -1,0.9,-1.1,1
- -bit Um inteiro, que caso fornecido, filtra aquele plano de bit antes de aplicar a convolução.
- -r Indica se a imagem de saída terá seu intervalo de intensidades remapeado para [0, 255] usando uma transformação linear.

- -h3h4 Indica se a combinação $\sqrt{h3^2 + h4^2}$ deve ser usada ao invés das máscaras simples.

Ex: python3 trabalho1.py -i ./seagull.png -o ./saida.png -num 3 -bit 7 -m mirror -r true

Apesar de todos os parâmetros acima serem opcionais, ao menos um dentre -num, -k ou -h3h4 precisa ser fornecido. Para mais detalhes é possível executar o *script* com o parâmetro -h.

A imagem é lida de forma monocromática usando a função *io.imread(..., as_gray=True)* do skimage, mas como essa função pode retornar imagens de intensidades fracionarias no intervalo [0, 1] a função *img_as_ubyte* é aplicada logo em seguida para garantir valores em [0, 255]. Se -bit for definido a plano de bit correspondente é filtrado. Por fim os valores são convertidos para *float* antes de aplicar os filtros. Após a aplicação o redimensionamento do intervalo é feito se -r for verdadeiro. Antes de salvar a imagem em arquivo é feito um arredondamento seguido de um *clipping* para o intervalo [0, 255].

3 Avaliação dos resultados

Nesta seção será avaliado o resultado de cada filtro e uma possível aplicação será proposta. As imagens ilustrativas foram feitas usando o método nearest de tratamento de bordas onde o valor atribuído é aquele mais próximo da posição a ser considerada.

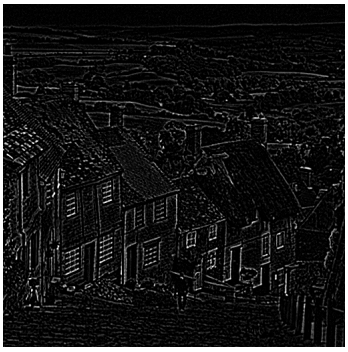
- h1: É um filtro de detecção de bordas, mas ele é sensível a ruídos da imagem. Na figura 1 fica claro essa sensibilidade em que uma das imagens teve seu ruido reduzido aplicando o filtro h6.
- h2: Esse é um filtro Gaussiano que "borra"a imagem fazendo uma média ponderada que dá mais peso a valores próximos ao centro. Ele elimina picos de intensidades associados a ruídos pois reduz a variância dos valores da imagem.
- h3 e h4: O primeiro realça linhas verticais da imagem e o segundo as horizontais. Fica mais fácil de ver o efeito em imagens com pouca textura como na figura 3 (c) e (d).
- h5: Corresponde a um filtro passa-alta que realça detalhes da imagem como bordas, curvas e linhas. Mas como pode ser visto na figura 2 (d) o ruído da imagem é também destacado.
- h6: Como já mencionado esse é um filtro de suavização de imagem que reduz o ruído original. O efeito obtido é de borrar a imagem original.

- h7 e h8: São filtros de realce de borda assim como os filtros h3 e h4, mas nesse caso há um destaque de linhas diagonais.
- h9: Tem um efeito de borrar a imagem no sentido da diagonal principal do filtro.
- h10: Produz um realce de detalhes aumentando o contraste.
- h11: É mais um detector de bordas diagonais, dessa vez na direção sudeste.
- $\sqrt{h3^2 + h4^2}$: Realiza uma combinação das bordas horizontais e verticais detectadas e portanto o resultado é um realce de todas as bordas da imagem original.

É interessante notar que foi possível melhorar a detecção de bordas aplicando um filtro Gaussiano ou h6 antes, já que ao borrar a imagem apenas bordas "fortes"são mantidas. Além disso foi obtido um resultado muito mais "limpo"aplicando a detecção apenas no plano de bit 7 (o mais significativo) como visto na figura 4.

(a) Imagem original

(b) h1

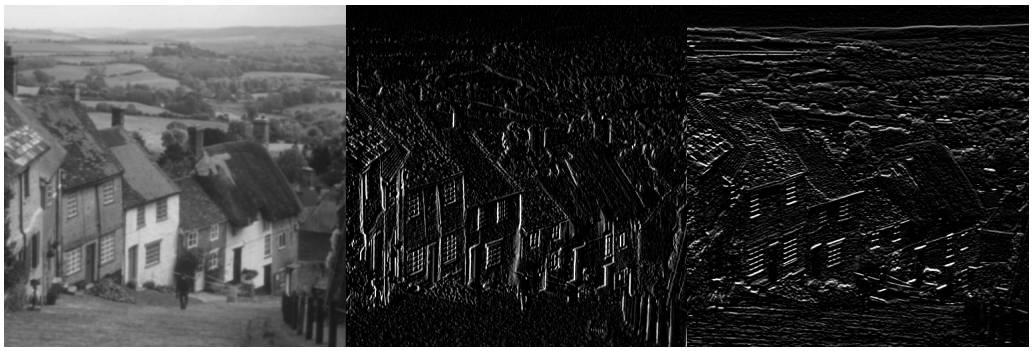


(c) h1 com h6 antes

Figura 1: Aplicação do filtro h1 com e sem redução de ruídos antes

4 Conclusão

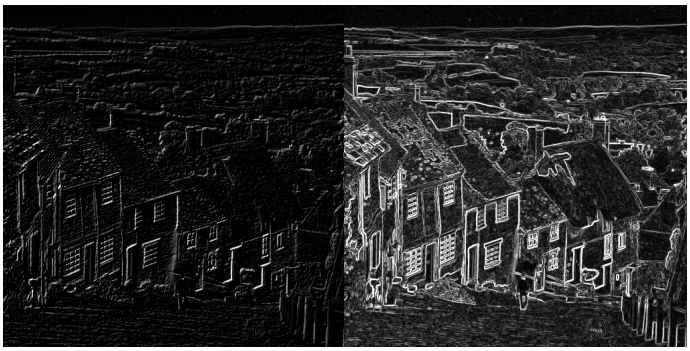
Apesar de serem muito simples, os filtros aplicados nesse trabalho produzem resultados interessantes e que podem ser utilizados como entrada para análises mais profundas de imagens. Os devidos arredondamentos e tratamentos foram aplicados e portando não limitações quanto a valores na imagem de entrada/saída.



(a) h2 (b) h3 (c) h4

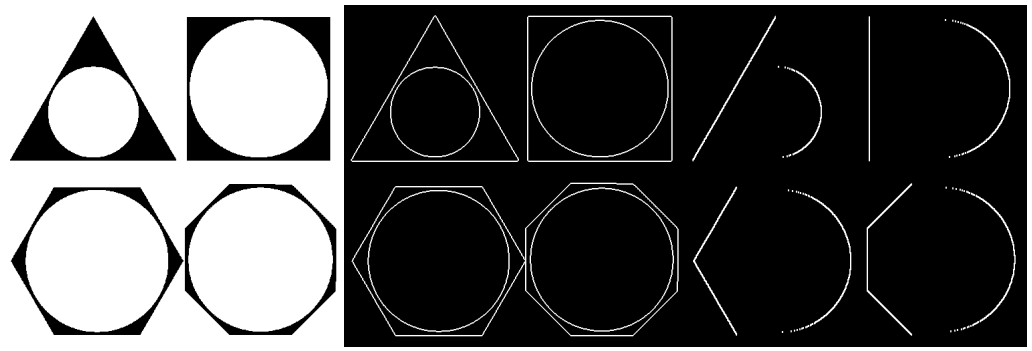
(d) h5 (e) h6 (f) h7

(g) h8 (h) h9 (i) h10

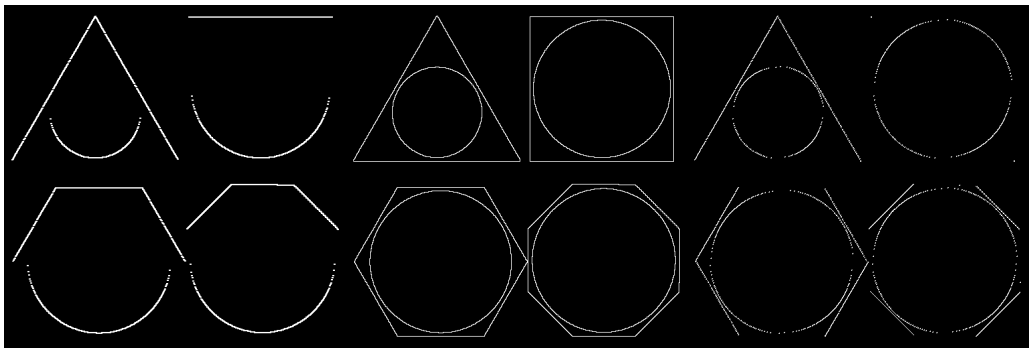


(j) h11 (k) $\sqrt{h3^2 + h4^2}$

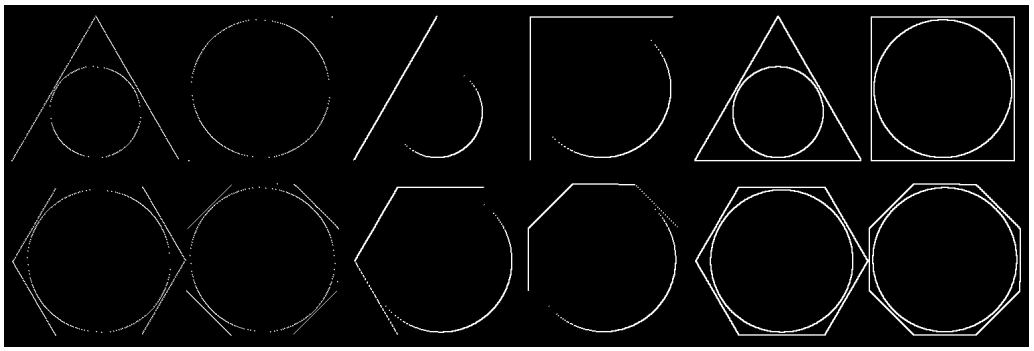
Figura 2: Resultado da aplicação de cada filtro a imagem city.png



(a) Imagem original (b) h1 (c) h3



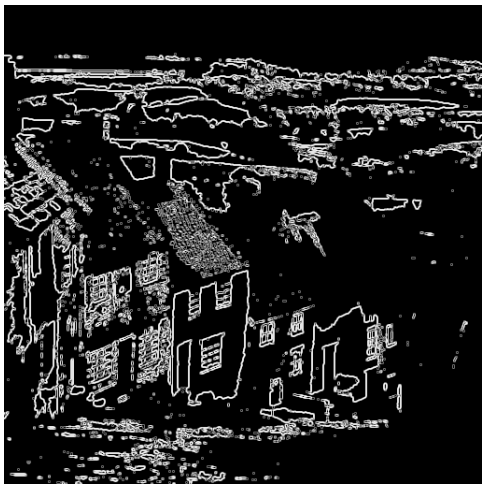
(d) h4 (e) h5 (f) h7



(g) h8 (h) h11 (i) $\sqrt{h3^2 + h4^2}$

Figura 3: Resultado da aplicação de cada filtro de detecção de borda a imagem poly.png

(a) Plano de bit 7



(b) $\sqrt{h3^2 + h4^2}$ aplicado ao plano de bit 7

Figura 4: Diferença entre aplicar a detecção a imagem original e ao plano de bit 7