

# PSET 1: processamento de imagens

## — Linguagens de Programação —

Preparado pelo Staff da MIT 6.009

(tradução: Abrantes Araújo Silva Filho)

Data de Entrega: 06/11/2025

### LEIA COM ATENÇÃO!

Este PSET é uma tradução da primeira tarefa de programação que os alunos da disciplina “MIT 6.009: Fundamentals of Programming” recebem logo no primeiro dia de aula, feita para os alunos da disciplina “Linguagens de Programação” na Universidade Vila Velha (UVV). Originalmente, no MIT, esta tarefa deveria ser completada no prazo de uma semana mas, para os alunos aqui na UVV, o prazo será de duas semanas. Por favor, leia com atenção todas as instruções. Boa programação!

## Sumário

<b>1</b>	<b>Preparação</b>	<b>3</b>
<b>2</b>	<b>Introdução</b>	<b>4</b>
2.1	Representação de imagens e a codificação de cores . . . . .	5
2.2	Carregando, salvando e mostrando imagens . . . . .	6
<b>3</b>	<b>Filtragem de imagem por meio de transformações por pixel</b>	<b>6</b>
3.1	Adicionando um caso de teste . . . . .	7
3.2	Debugging . . . . .	8
<b>4</b>	<b>Filtragem de imagem por correlação</b>	<b>8</b>
4.1	Instruções de correlação . . . . .	8
4.2	Efeitos das bordas . . . . .	10
4.3	Kernels de exemplo . . . . .	11
4.3.1	Identidade . . . . .	11

4.3.2	Translação . . . . .	11
4.3.3	Média . . . . .	11
4.4	Correlação . . . . .	12
<b>5</b>	<b>Desfoque e nitidez</b>	<b>12</b>
5.1	Desfoque (borrado) . . . . .	12
5.2	Nitidez . . . . .	13
<b>6</b>	<b>Detecção de borda</b>	<b>14</b>
<b>7</b>	<b>Envio do seu código</b>	<b>15</b>
<b>8</b>	<b>Discussão do código</b>	<b>15</b>
<b>9</b>	<b>Tarefas adicionais (não obrigatório)</b>	<b>15</b>
9.1	Filtros adicionais . . . . .	16
9.2	Redimensionamento sensível ao conteúdo . . . . .	16
9.2.1	Algoritmo . . . . .	17
9.2.2	Implementação . . . . .	18
9.2.3	Seam carving . . . . .	18
9.3	Cores . . . . .	20

# 1 Preparação

Este laboratório pressupõe que você tenha o Python 3.5 ou posterior instalado em sua máquina (versões mais recentes, como a 3.11, são recomendadas). Caso você não tenha o Python em sua máquina, siga as instruções de instalação disponíveis na página de [download do Python](#)<sup>1</sup>.

Este laboratório também usará a biblioteca `pillow`, que servirá para carregar e salvar imagens. Consulte [esta página](#)<sup>2</sup> para obter instruções sobre como instalar a `pillow` (observe que, dependendo da sua configuração, pode ser necessário executar `pip3` em vez de `pip`).

Cada aluno deve fazer o download do arquivo `pset1.zip`, que contém código e outros arquivos que serão utilizados como ponto de partida para este PSET.

A maioria da codificação deve ser feita no arquivo `pset1.py`, que você enviará no final deste PSET. **IMPORTANTE:** você não deve adicionar nenhuma importação ao arquivo (não deve usar o comando `import`), nem deve usar o módulo `pillow` para nada além de carregar e salvar imagens (funções que já estão implementadas para você).

A pontuação neste PSET será baseada em:

- Respostas corretas para as perguntas discursivas que você responderá ao longo deste trabalho;
- Seu código passar em todos os testes do arquivo `test.py` que está disponível para você, e passar em todos os testes adicionais que serão executados pelo Autolab e pelo professor (você não terá acesso aos testes adicionais; e
- Uma apresentação ORAL e INDIVIDUAL, de 3 a 5 minutos, onde você deve explicar todo seu código para o professor. Essa explicação deve comprovar que você fez o trabalho por conta própria, estudou, programou e sabe de fato o que cada parte do código está fazendo.

## ATENÇÃO COM A INTEGRIDADE ACADÊMICA!

Você deve seguir todas as normas de integridade acadêmica da Universidade e da disciplina. Em especial, preste muita atenção à seguinte regra:

- Se, na apresentação oral ao professor, você não souber explicar seu código, linha por linha, função por função e módulo por módulo, e não conseguir explicar como seu programa de modo geral, seu trabalho **SERÁ CONSIDERADO PLÁGIO** e você ficará com nota 0 (zero) neste PSET.

**ATENÇÃO:** o seu código deve conter muitos comentários explicativos, que mostrem para quem está lendo o que você pensou e porque codificou da maneira

<sup>1</sup><https://www.python.org/downloads/>


<sup>2</sup><https://pillow.readthedocs.io/en/stable/>

realizada. Em resumo, seu código deve estar TOTALMENTE COMENTADO, com explicações a respeito de tudo. Além disso, preste atenção ao seguinte:

- Comentários em inglês são forte indício de que o código foi simplesmente copiado da internet. Trabalhos com comentários em inglês terão nota zerada.
- Nomes de variáveis ou funções em inglês são forte indício de que o código foi simplesmente copiado da internet. Trabalhos com identificadores em inglês (exceto as funções que já estão prontas no arquivo original) terão a nota zerada.
- Todo código que você escrever será enviado para sistemas automatizados de detecção de plágio (MOSS e compare50). Alunos que copiarem código (da internet ou de outros alunos) terão a nota zerada.

## 2 Introdução

Você já assistiu *CSI: Crime Scene Investigation* ou outro programa de fantasia investigativa processual? Se sim, as palavras “melhore essa imagem” (*enhance this image*) devem soar familiar. Assista ao vídeo a seguir para saber do que estamos falando:

 [Let's Enhance<sup>3</sup>](#)

O problema é que a realidade não é tão simples como a ficção. Uma imagem não pode ser melhorada de modo tão espetacular e tão simples como nos filmes<sup>4</sup>. No mundo real, temos muitas técnicas para analisar e tratar imagens. O Photoshop e o GIMP são conjuntos de ferramentas cheios de algoritmos de manipulação de imagens que dão ao usuário grandes habilidades para avaliar, manipular e transformar qualquer imagem digital.

Neste PSET você ajudará nosso próprio laboratório “CSI: Computer Science Investigation” a construir suas ferramentas para manipulação de imagens. Ao final do PSET você terá escrito o código para inverter, desfocar, aumentar a nitidez e encontrar as bordas em imagens em tons de cinza. Também fornecemos sugestões de outras manipulações de imagens que se baseiam neste PSET (incluindo como trabalhar com imagens coloridas), se você quiser expandir ainda mais seu conjunto de ferramentas. Muitos filtros de imagem do mundo real são implementados usando as mesmas ideias que desenvolveremos ao longo deste trabalho.

---

<sup>3</sup><https://www.youtube.com/watch?v=Vxq9yj2pVWk>

<sup>4</sup>Embora o MIT esteja trabalhando para criar alguma coisa parecida: <http://people.csail.mit.edu/billf/project%20pages/sresCode/Markov%20Random%20Fields%20for%20Super-Resolution.html>

## 2.1 Representação de imagens e a codificação de cores

Para ajudar nosso laboratório CSI a começar a **manipular** imagens com código, primeiro precisamos aprender a **representar** digitalmente as imagens em código<sup>5</sup>.

Embora as imagens digitais possam ser representadas de inúmeras maneiras, a mais comum resistiu ao teste do tempo: um mosaico retangular de **pixels** — pontos coloridos, que juntos formam a imagem. Uma imagem, portanto, pode ser definida especificando uma **largura**, uma **altura** e uma **matriz de pixels**, cada um dos quais é um valor de cor. Essa representação surgiu desde os primórdios da televisão analógica e sobreviveu a muitas mudanças tecnológicas. Embora os formatos de arquivo individuais empreguem diferentes codificações, compactação e outros truques, a representação de matriz de pixels permanece central para a maioria das imagens digitais.

Nosso laboratório CSI está apenas começando, então eles querem manter as coisas simples e focar apenas em imagens em tons de cinza. Cada pixel é codificado como um único inteiro no intervalo  $[0, 255]$  (1 byte pode conter 256 valores diferentes), sendo 0 o preto mais profundo e 255 o branco mais brilhante que podemos representar. A gama completa é mostrada abaixo:



Para nosso laboratório, representaremos imagens como instâncias de uma classe já fornecida para você, a classe `Imagem`, que tem três variáveis de instância:

- `largura`: a largura da imagem (em pixels);
- `altura`: a altura da imagem (em pixels); e
- `pixels`: uma lista Python de pixels armazenados em *row-major order*<sup>6</sup> (listando a linha superior da esquerda para a direita, depois a próxima linha e assim por diante).

Por exemplo, considere esta imagem  $2 \times 3$  pixels (ampliada aqui para maior clareza):



Essa imagem seria codificada como a seguinte instância:

```
i = Imagem(2, 3, [0, 50, 50, 100, 100, 255])
```

<sup>5</sup>A representação de imagens e cores digitais como dados é um tema rico e interessante. Você já notou, por exemplo, que suas fotos de flores roxas são incapazes de capturar o roxo vibrante que você vê na vida real? Aqui está o motivo: [https://en.wikipedia.org/wiki/CIE\\_1931\\_color\\_space](https://en.wikipedia.org/wiki/CIE_1931_color_space)

<sup>6</sup>[https://en.wikipedia.org/wiki/Row-\\_and\\_column-major\\_order](https://en.wikipedia.org/wiki/Row-_and_column-major_order)

## 2.2 Carregando, salvando e mostrando imagens

Fornecemos três métodos próximos à parte inferior da classe `Imagem` que podem ser úteis para depuração: `carregar`, `salvar` e `mostrar`. Cada uma dessas funções é explicada por meio de uma [docstring](https://en.wikipedia.org/wiki/Docstring#Python)<sup>7</sup>.

Você não precisa se aprofundar no entendimento desses códigos, mas dê uma olhada nas docstrings e tente usar os métodos para:

- Carregar uma imagem;
- Salvar a imagem com um nome diferente; e
- Mostrar a imagem.

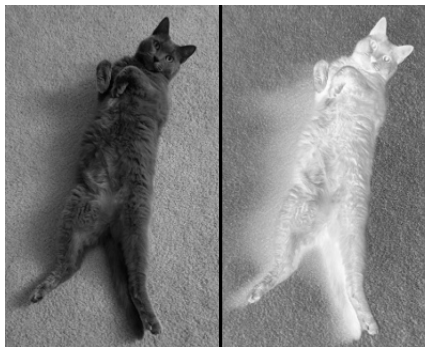
Existem várias imagens de exemplo no diretório `test_imagens`, dentro do arquivo de código do PSET. Você também pode usar suas próprias imagens para testar.

À medida que você implementa os vários filtros, essas funções podem fornecer uma maneira de visualizar sua saída, o que pode ajudar na depuração, e também facilita a exibição de seus resultados interessantes para amigos e familiares.

Você pode adicionar código para carregar, manipular e salvar imagens no bloco `if __name__ == '__main__':`, no arquivo `pset1.py`, que será executado quando você executar o arquivo diretamente diretamente (mas não quando for importado pela suíte de teste).

## 3 Filtragem de imagem por meio de transformações por pixel

Nosso laboratório decidiu que a primeira ferramenta que eles gostariam de construir é um filtro de **inversão**, que reflete pixels sobre o valor de cinza médio (0 preto se torna 255 branco e vice-versa). Por exemplo, aqui está uma fotografia do gato de Adam Hartz. Do lado esquerdo está a imagem original, e do lado direito está uma versão invertida.



---

<sup>7</sup><https://en.wikipedia.org/wiki/Docstring#Python>

Infelizmente, os programadores CSI não esperaram pelo especialista do MIT (você) e, em vez disso, tentaram concluir o trabalho eles mesmos. A maior parte da implementação do filtro de inversão foi concluída para você (ele é invocado chamando o método chamado `invertida`), mas algumas partes não foram implementadas corretamente. Sua primeira tarefa é corrigir a implementação com erros do filtro de inversão.

Antes de fazer isso, no entanto, vamos adicionar um caso de teste simples para que possamos testar se nosso código está funcionando.

### 3.1 Adicionando um caso de teste

Vamos começar com uma imagem  $4 \times 1$  que é definida com os seguintes parâmetros:

- `largura`: 4
- `altura`: 1
- `pixels`: [29, 89, 136, 200]

**QUESTÃO 01:** se você passar essa imagem pelo filtro de inversão, qual seria o output esperado? Justifique sua resposta.

Vamos também adicionar este caso de teste aos testes regulares do laboratório para que ele seja executado quando executarmos o arquivo `test.py`. Se você abrir o arquivo `test.py` em um editor de texto, verá que é um arquivo Python que usa o módulo `unittest` do Python para realizar [testes unitários](#)<sup>8</sup>.

Cada classe neste arquivo serve como uma coleção de casos de teste e cada método dentro de uma classe representa um caso de teste específico.

A execução de `test.py` fará com que o Python execute e relate todos os testes no arquivo. No entanto, você pode fazer o Python executar apenas um subconjunto dos testes executando, por exemplo, o seguinte comando de um terminal<sup>9</sup> (sem incluir o cifrão):

```
$ python3 test.py TestImagem
```

Isso executará todos os testes na classe `TestImagem`. Se você executar este comando, deverá obter um breve relatório indicando que o único caso de teste foi aprovado. Se você quiser ser ainda mais específico, diga ao `unittest` para executar apenas um único caso de teste, por exemplo:

```
$ python3 test.py TestInvertida.test_invertida_1
```

(Observe que este caso de teste deve falhar agora porque a implementação dada do filtro de inversão tem bugs!)

---

<sup>8</sup><https://docs.python.org/3/library/unittest.html>

<sup>9</sup>Não execute este comando de dentro do Python, execute a partir de um terminal de comandos. Se estiver com dificuldade, procure um monitor ou o professor.

Para adicionar um caso de teste, você pode adicionar um novo método a uma das classes. É importante ressaltar que para que seja reconhecido como um caso de teste, seu nome deve começar com a palavra `test`<sup>10</sup>.

No arquivo que você recebeu há um caso de teste, `test_invertida_2`, no grupo `TestInvertida`. Modifique este método para que ele implemente o teste de cima (invertendo a pequena imagem  $4 \times 1$ ). Nesse caso de teste, você pode definir o resultado esperado como uma instância da classe `Imagem` e pode compará-lo com o resultado da chamada do método `invertida` da imagem original.

Por enquanto, este caso de teste deve falhar, mas podemos esperar que ele passe assim que todos os bugs no filtro de inversão forem corrigidos por você.

Em todo o PSET você pode (se achar útil) adicionar seus próprios casos de teste para outras partes do código, pois está depurando `pset1.py`, e quaisquer extensões ou funções de utilitário que escrever.

## 3.2 Debugging

Agora é hora de trabalhar para encontrar e corrigir os erros no código fornecido para o filtro de inversão. Boa depuração!

**QUESTÃO 02:** faça a depuração e, quando terminar, seu código deve conseguir passar em todos os testes do grupo de teste `TestInvertida` (incluindo especificamente o que você acabou de criar). Execute seu filtro de inversão na imagem `test_images/bluegill.png`, salve o resultado como uma imagem PNG e salve a imagem.

# 4 Filtragem de imagem por correlação

O pessoal do nosso laboratório CSI ficou impressionado! Eles também aprenderam a lição e não tentarão mais codificar suas próprias ferramentas de manipulação de imagens. Uau!

O laboratório agora solicitou um recurso um pouco mais avançado, envolvendo a correlação de imagens com vários kernels. Abaixo estão as transcrições de suas instruções.

## 4.1 Instruções de correlação

Dada uma imagem de entrada  $I$  e um kernel  $k$ , aplicar  $k$  à  $I$  resulta em uma nova imagem  $O$  (talvez com pixels não-inteiros e/ou fora dos limites), com altura e largura iguais à  $I$ , mas com pixels que são calculados de acordo com as regras descritas por  $k$ .

---

<sup>10</sup>Você também pode criar grupos de novos testes criando uma nova classe cujo nome também comece com a palavra `Test`.



O processo de aplicar o kernel  $k$  à uma imagem  $I$  é realizado como uma **correlação**: o pixel na posição  $(x, y)$  na imagem resultante, que vamos denominar de  $O_{x,y}$  (com  $O_{0,0}$  sendo o canto superior esquerdo), é expresso como uma combinação linear dos pixels ao redor da posição  $(x, y)$  na imagem de entrada, onde os pesos são dados pelo kernel  $k$ .

Como exemplo, vamos começar com um kernel  $3 \times 3$ :

	0	1	0	
	0	0	0	
	0	0	0	

Quando nós aplicamos esse kernel à uma imagem  $I$ , cada pixel  $O_{x,y}$  na imagem resultante é a combinação linear dos 9 pixels mais próximas à  $(x,y)$  em  $I$ , onde o valor de cada pixel de entrada é multiplicado pelo valor associado no kernel:

35	40	41	45	50	×						=					
40	40	42	46	52			0	1	0							
42	46	50	55	55			0	0	0							
48	52	56	58	60			0	0	0					42		
56	60	65	70	75												

Em particular, para um kernel  $3 \times 3$  nós temos:

$$O_{x,y} = I_{x-1,y-1} \times k_{0,0} + I_{x,y-1} \times k_{1,0} + I_{x+1,y-1} \times k_{2,0} + \quad (1)$$

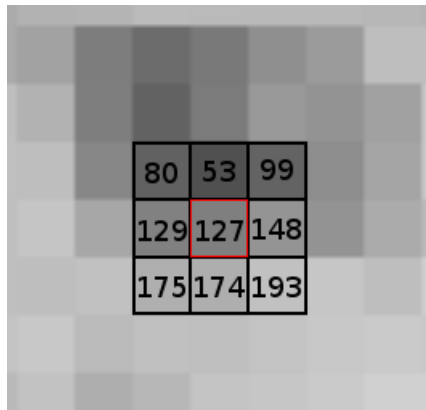
$$I_{x-1,y} \times k_{0,1} + I_{x,y} \times k_{1,1} + I_{x+1,y} \times k_{2,1} + \quad (2)$$

$$I_{x-1,y+1} \times k_{0,2} + I_{x,y+1} \times k_{1,2} + I_{x+1,y+1} \times k_{2,2} \quad (3)$$

**QUESTÃO 03:** considere uma etapa de correlacionar uma imagem com o seguinte kernel:

0.00	-0.07	0.00
-0.45	1.20	-0.25
0.00	-0.12	0.00

Aqui está uma parte de uma imagem de amostra, com as luminosidades específicas de alguns pixels:



Qual será o valor do pixel na imagem de saída no local indicado pelo destaque vermelho? Observe que neste ponto ainda não arredondamos ou recortamos o valor, informe exatamente como você calculou. Observação: demonstre passo a passo os cálculos realizados.

## 4.2 Efeitos das bordas

Ao calcular os pixels no perímetro de  $O$ , menos de 9 pixels de entrada estão disponíveis. Para um exemplo específico, considere o pixel superior esquerdo na posição  $(0,0)$ . Nesse caso, todos os pixels acima e à esquerda de  $(0,0)$  estão fora dos limites. Uma opção que temos para lidar com esses efeitos de borda é tratar cada pixel fora dos limites como tendo um valor de 0. No entanto, isso pode levar a artefatos infelizes nas bordas de nossas imagens.

Ao implementar a correlação, consideraremos esses pixels fora dos limites em termos de uma **versão estendida** da imagem de entrada. Os valores à esquerda da imagem devem ser considerados como tendo os valores da coluna 1, os valores na parte superior da imagem devem ser considerados como tendo os valores da linha 1, etc., conforme ilustrado no diagrama a seguir<sup>11</sup> (observe, no entanto, que a imagem deve ser estendida em todas as quatro direções, não apenas para o canto superior esquerdo):



<sup>11</sup>Esta imagem, licenciada sob *Creative Commons Attribution-Share Alike 3.0 Unported License*, foi criada por Michael Plotke e obtida através da [Wikimedia Commons](#)

Para fazer isso, implemente uma alternativa para a função `get_pixel`, que retorna valores dos pixels de dentro da imagem normalmente, mas que lida com pixels fora dos limites retornando valores apropriados conforme discutido acima, em vez de gerar uma exceção. Seu código de correlação, então, não terá que se preocupar se os pixels estão dentro dos limites ou não.

### 4.3 Kernels de exemplo

Existe um mundo de operações interessantes que podem ser expressas como kernels de imagem (alguns exemplos podem ser vistos nas seções abaixo), e muitos programas científicos também usam esse padrão, então sinta-se à vontade para experimentar.

Mas atenção: note que a saída de uma correlação **não necessariamente** é uma imagem “legal” (os pixels podem estar fora do intervalo  $[0, 255]$  ou podem ser floats). Portanto, a etapa final em cada função de processamento de imagem é recortar valores de pixel negativos para 0 e valores acima de 255 para o limite de 255, e garantir que todos os valores na imagem sejam inteiros. Dica: você pode querer implementar uma função para realizar esta operação em uma imagem.

#### 4.3.1 Identidade

O kernel abaixo representa uma transformação de **identidade**: aplicar esse kernel à uma imagem de entrada, resulta na mesma imagem de saída, sem alterações.

```
0 0 0
0 1 0
0 0 0
```

#### 4.3.2 Translação

O kernel abaixo desloca a imagem de entrada dois pixels para a direita, descarta as duas colunas de pixels mais à direita e duplica as duas colunas mais à esquerda.

```
0 0 0 0 0
0 0 0 0 0
1 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```

#### 4.3.3 Média

O kernel abaixo resulta em uma imagem de saída onde cada pixel é a média dos 5 pixels mais próximos da entrada.

```
0.0 0.2 0.0
0.2 0.2 0.2
0.0 0.2 0.0
```

## 4.4 Correlação

Se você ainda não fez isso, você vai querer neste ponto escrever uma função para lidar com essas correlações em um sentido geral (para uma imagem arbitrária e um kernel arbitrário). Observe que sua função não deve modificar a imagem de entrada durante a correlação, deve criar uma nova. Em relação ao kernel, cabe a você escolher como representá-lo dentro do seu código.

Observação: para ajudar no debugging, você pode escrever alguns casos de teste correlacionando `test_images/centered_pixel.png` ou alguma outra imagem simples com alguns kernels.

Você pode usar os kernels acima para ajudar a testar se seu código produz os resultados esperados. Você pode querer primeiro escrever seu código especificamente para kernels  $3 \times 3$  e então generalizar para kernels de tamanho arbitrário (no entanto, você pode assumir que kernels sempre serão quadrados e que terão um número ímpar de linhas e colunas).

**QUESTÃO 04:** quando você tiver implementado seu código, tente executá-lo em `test_images/pigbird.png` com o seguinte kernel  $9 \times 9$ :

```
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
```

Ao rodar esse kernel, salve a imagem resultante.

## 5 Desfoque e nitidez

### 5.1 Desfoque (borrado)

Nosso laboratório CSI está mais uma vez impressionado! Eles estão animados para trabalhar com a nova ferramenta que você criou e querem incluí-lo nas primeiras aplicações: **desfoque de caixa** (*box blur*)!

O filtro de **desfoque de caixa** pode ser implementado alimentando um kernel apropriado para nossa função de correlação recém-criada.

Para um desfoque de caixa, o kernel é um quadrado  $n \times n$  de valores idênticos que somam 1. Como você pode ser solicitado a experimentar diferentes valores de  $n$ , talvez queira definir uma função que receba um único argumento  $n$  e retorne um kernel de desfoque de caixa  $n \times n$ .

Implemente o método `borrada` na classe `Imagem`, que recebe um número  $n$  e retorna uma nova `Imagem` com altura e largura iguais às dimensões da imagem de entrada que foi desfocada com um kernel de desfoque de caixa de tamanho apropriado. Tome cuidado para produzir valores de brilho inteiros (`round(valor)`, por exemplo) no intervalo  $[0, 255]$ . Ou seja, em muitos casos, você precisará recortar valores inteiros explicitamente para esse intervalo.

Quando você terminar e seu código passar em todos os testes relacionados ao desfoque, execute seu filtro na imagem `test_images/cat.png` com um kernel de desfoque de caixa de tamanho 5, salve o resultado como uma imagem PNG.

## 5.2 Nitidez

Agora, é óbvio, nenhum laboratório CSI estaria completo sem uma operação de “Melhorar imagem!” (*enhance!*). Nosso laboratório ainda se contenta em começar pequeno e, portanto, solicita que você faça isso implementando um filtro de **nitidez** (*sharpen*).

A operação de “nitidez” geralmente tem outro nome que é mais sugestivo do que isso significa: muitas vezes é chamada de **máscara de não nitidez** (*unsharp mask*) porque resulta da subtração de uma versão “não nítida” (embaçada) da imagem de uma versão em escala da imagem original.

Mais especificamente, se tivermos uma imagem ( $I$ ) e uma versão borrada dessa mesma imagem ( $B$ ), o valor da imagem nítida  $S$  em um determinado local é:

$$S_{x,y} = \text{round}(2I_{x,y} - B_{x,y})$$

Uma maneira de implementar essa operação é computar uma versão borrada da imagem e, então, para cada pixel, computar o valor dado pela equação acima. No entanto, é realmente possível realizar esta operação com uma única correlação (com um kernel apropriado).

**QUESTÃO 05:** se quisermos usar uma versão desfocada  $B$  que foi feita com um kernel de desfoque de caixa de  $3 \times 3$ , que kernel  $k$  poderíamos usar para calcular toda a imagem nítida com uma única correlação? Justifique sua resposta mostrando os cálculos.

Implemente uma máscara de não nitidez como o método `focada` da classe `Imagem`, onde  $n$  denota o tamanho do kernel de desfoque que deve ser usado para gerar a cópia desfocada da imagem. Este método deve retornar uma nova imagem mais nítida. Você pode implementar isso como uma correlação única ou usando

uma subtração explícita, mas se você usar uma subtração explícita, certifique-se de não fazer nenhum arredondamento até o final (a versão desfocada intermediária não deve ser arredondada ou cortada de forma alguma).

Quando terminar e seu código passar nos testes relacionados à nitidez, execute seu filtro de nitidez na imagem `test_images/python.png` usando um kernel de tamanho 11, salve o resultado como uma imagem PNG.

## 6 Detecção de borda

Nosso laboratório CSI tem seu primeiro caso! Eles não podem nos dizer o quê, é algo sobre “segurança nacional”, mas aparentemente precisam de uma ferramenta de **detecção de bordas**, e rápido!

Para esta parte do laboratório, implementaremos um [operador Sobel](#), que é útil para detectar bordas em imagens.

Este detector de borda é mais complexo do que um kernel simples de imagem, mas é uma combinação de dois kernels de imagem  $K_x$  e  $K_y$ :

Kernel  $K_x$ :

```
-1 0 1
-2 0 2
-1 0 1
```

Kernel  $K_y$ :

```
-1 -2 -1
0 0 0
1 2 1
```

Após calcular  $O_x$  e  $O_y$  correlacionando a entrada com  $K_x$  e  $K_y$  respectivamente, cada pixel da saída é a raiz quadrada da soma dos quadrados dos pixels correspondentes em  $O_x$  e  $O_y$ :

$$O_{x,y} = \text{round} \left( \sqrt{O_{x,y}^2 + O_{y,y}^2} \right)$$

Novamente, tome cuidado para garantir que a imagem final seja composta de pixels inteiros no intervalo  $[0, 255]$ . Mas apenas recorte a saída depois de combinar  $O_x$  e  $O_y$ . Se você cortar os resultados intermediários, o cálculo de combinação estará incorreto.

**QUESTÃO 06:** explique o que cada um dos kernels acima, por si só, está fazendo. Tente executar `mostrar` nos resultados dessas correlações intermediárias para ter uma noção do que está acontecendo aqui.

Implemente o detector de bordas como o método `bordas` dentro da classe `Imagem`. O método deve retornar uma nova instância de `Imagem` resultante das operações acima.

Quando terminar e seu código passar nos testes de detecção de borda, execute seu detector de borda na imagem `test_images/construct.png`, salve o resultado como uma imagem PNG.

## 7 Envio do seu código

O professor especificará a forma de envio e explicará nas aulas.

Testaremos os filtros que você implementou (inversão, desfoque, nitidez e bordas) em outras imagens para verificar se estão funcionando corretamente.

Obviamente **todos os códigos enviados pelos alunos SERÃO VERIFICADOS CONTRA PLÁGIO!** Se identificarmos códigos plagiados entre os alunos e/ou copiados de sites da internet, o PSET será zerado e o aluno encaminhado à coordenação acadêmica para aplicação das sanções previstas no código de Integridade Acadêmica da instituição.

## 8 Discussão do código

Depois que enviar as respostas e o código final, você precisa passar por uma prova oral de discussão de código com seu professor. Atenção: **você deve estar pronto para discutir seu código em detalhes!** Se você fez todo o PSET por conta própria, mesmo que não tenha conseguido fazer tudo corretamente, essa discussão não será problema para você. Você também deverá trazer as respostas das questões, por escrito, em papel almaço.

Você deve estar preparado para demonstrar seu código (que deve ser bem comentado, deve evitar repetições e deve fazer bom uso de funções auxiliares). Em particular, esteja preparado para discutir:

- Seu caso de teste para inversão.
- Sua implementação de correlação.
- Sua implementação da máscara de nitidez.
- O funcionamento geral e detalhado do código.

## 9 Tarefas adicionais (não obrigatório)

Se você terminou com antecedência, há muitos filtros de imagem interessantes que podem ser adicionados, e nós encorajamos você a explorar (desculpe, mas sem pontos adicionais!). Algumas ideias que você pode querer considerar são mostradas nas seções abaixo.

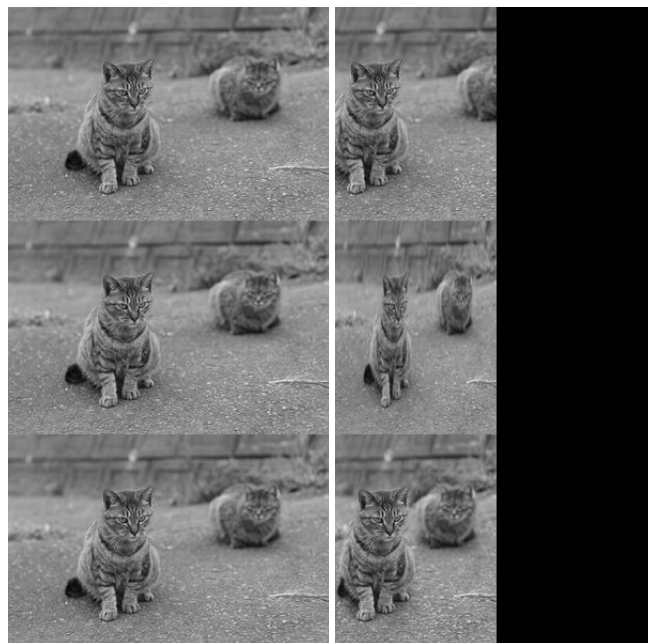
## 9.1 Filtros adicionais

- Implemente um filtro de **normalização** que dimensionará os valores dos pixels linearmente, de modo que o pixel mais escuro na saída tenha um valor de 0 e o mais claro tenha um valor de 255.
- Adicione capacidade para um **desfoque gaussiano** com valor arbitrário de  $\sigma$  ([saiba mais](#)).
- Crie um filtro mais complexo com efeitos dependentes da posição (como ondulação ou vinheta). Se você estudou álgebra linear (e se lembra da matéria!), tente girar a imagem.

## 9.2 Redimensionamento sensível ao conteúdo

Como um exercício opcional adicional para este PSET, podemos percorrer a implementação de uma versão de redimensionamento de imagens com reconhecimento de conteúdo (geralmente chamado de *retargeting*). O objetivo desta técnica é reduzir a escala de uma imagem, preservando as partes perceptivelmente importantes (por exemplo, removendo o fundo, mas preservando os assuntos).

Duas abordagens comuns para redimensionar uma imagem são **cortar** e **dimensionar** (*cropping* e *scaling*). No entanto, em certas situações, ambos os métodos podem levar a resultados indesejáveis. As figuras abaixo mostram esses vários tipos de redimensionamento em uma imagem de dois gatos, que foi reduzida de 300 pixels de largura para 150 pixels de largura. O corte é mostrado na primeira linha, o dimensionamento ingênuo na segunda linha, e uma forma de *retargeting* na terceira linha:





As duas primeiras estratégias têm deficiências: o corte faz com que um dos gatos seja quase completamente excluído e a escala distorce ambos os gatos. Ao se concentrar na remoção de partes de “baixa energia” da imagem, no entanto, a técnica chamada de *seam carving* consegue manter os dois gatos quase completamente intactos enquanto remove principalmente o fundo.

Aqui está outro exemplo, em uma foto de várias árvores. Observe novamente que as árvores são preservadas quase exatamente, apesar da diminuição da largura da imagem em 75 pixels:



Essas imagens foram geradas usando uma técnica chamada *seam carving* (algo como “escultura de costura”). Começaremos implementando uma variante mais simples desse algoritmo e, em seguida, descreveremos uma implementação completa logo em seguida.

### 9.2.1 Algoritmo

Enquanto o corte normal de imagens funciona removendo colunas de pixels dos lados esquerdo e direito da imagem indiscriminadamente, essa abordagem funciona removendo colunas de pixels com baixa ‘energia’ (ou seja, onde não há muita mudança na imagem).

Cada vez que queremos diminuir o tamanho horizontal da imagem em um pixel, começamos encontrando a coluna que possui a energia total mínima e removendo o pixels nela contidos. Para reduzir ainda mais a imagem, podemos aplicar esse processo repetidamente.

Aqui, definiremos a ‘energia’ com base no algoritmo de detecção de borda anterior: a energia de qualquer pixel será o valor associado na saída da detecção de borda; e a energia de um caminho é a soma total das energias dos pixels que ele contém.

Isso pode ser implementado repetindo as três etapas a seguir até que a largura da imagem atinja o tamanho desejado:

1. **Calcular o mapa de energia.** Como estamos usando nosso detector de bordas como nossa função de ‘energia’, isso deve envolver simplesmente chamar o método de bordas da imagem.

2. **Encontre a coluna de energia mínima** somando todas as energias em cada coluna e escolhendo o mínimo.
3. **Remova o caminho calculado.** Por fim, precisamos remover os pixels da coluna escolhida da imagem.

Se precisarmos remover mais colunas, podemos repetir todas as três etapas desse processo (incluindo recalcular o mapa de energia).

### 9.2.2 Implementação

Agora é hora de implementar o algoritmo de *retargeting*.

Você pode implementar isso usando qualquer nome que preferir e em qualquer local do arquivo. Pode fazer sentido, também, definir algumas funções “auxiliares” (talvez correspondendo às etapas acima) em vez de implementar tudo em uma função (por exemplo, você pode desejar definir uma função auxiliar para encontrar a coluna de energia mais baixa em uma imagem e outro para remover um determinado número de coluna de uma imagem).

Observe que esse processo deve envolver o cálculo de um mapa de energia, a remoção de uma única coluna, o cálculo do mapa de energia novamente e a remoção de uma segunda coluna.

Para verificar a si mesmo, execute seu código para fazer o seguinte:

- Remova 75 colunas de `arvore.png`
- Remova 100 colunas de `gatos.png`
- Remova 100 colunas de `porco.png`

Salve as imagens resultantes e compare-as com os originais. Observe que esses testes podem levar muito tempo (vários minutos) para serem executados.

Note que você pode notar alguns artefatos nestas imagens (particularmente no céu em `arvores.png`). Se você estiver interessado em melhorar isso, continue lendo nossa discussão sobre *seam carving*.

### 9.2.3 Seam carving

Se você der uma olhada no resultado de usar nosso código de *retargeting* original para redimensionar `arvore.png` em 75 pixels, você notará alguns artefatos no horizonte. É possível produzir resultados muito melhores usando uma variação dessa técnica chamada *seam carving*. Para mais informações, não perca o vídeo a seguir:

▶ [Image Resizing by Seam Carving.](#)

Você não receberá nenhum ponto adicional por implementar essa técnica, mas é **muito legal!**

Essa técnica é muito semelhante à que usamos anteriormente, exceto que, em vez de remover a **coluna** de menor energia em uma imagem, encontramos e removemos **‘caminhos’ de pixels** conectados de cima para baixo da imagem, com um pixel em cada linha. Cada vez que queremos diminuir o tamanho horizontal da imagem em um pixel, começamos encontrando o caminho conectado de cima para baixo que possui a energia total mínima e removendo os pixels contidos nele. Para reduzir ainda mais a imagem, podemos aplicar esse processo repetidamente.

Como acima, definiremos a ‘energia’ com base no algoritmo de detecção de borda anterior: a energia de qualquer pixel será o valor associado na saída da detecção de borda; e a energia de um caminho é a soma total das energias dos pixels que ele contém.

O objetivo de encontrar o caminho de energia mínima é alcançado por um algoritmo que é descrito brevemente abaixo e que é descrito em detalhes na página da [Wikipedia para o seam carving](#).

O algoritmo funciona repetindo as três etapas a seguir até que a largura da imagem atinja o tamanho desejado:

1. **Calcular o mapa de energia.** Como estamos usando nosso detector de bordas como nossa função de ‘energia’, isso deve envolver simplesmente chamar o método de bordas da imagem.
2. **Encontre o caminho de custo mínimo de cima para baixo.** Podemos começar criando um “mapa de energia cumulativa”, onde o valor em cada posição é o custo total do caminho de menor custo desde o topo da imagem até aquele pixel. Para a linha superior, isso será igual à linha superior do mapa de energia. As linhas subsequentes podem ser calculadas usando o seguinte algoritmo (em pseudocódigo):

Para cada linha de pixels na imagem:

Para cada pixel em uma linha:

Encontre o caminho de menor custo para alcançar esse pixel tomando o mínimo dos custos para alcançar cada um dos três possíveis pixels anteriores e adicionando isso ao custo do atual pixel. O custo atualizado do pixel atual agora reflete o custo total do caminho de custo mínimo para alcançar esse pixel.

O *seam* mínimo é então calculado retrocedendo da borda inferior até a borda superior. Primeiro, o pixel de valor mínimo na linha inferior do mapa de custo acumulado é localizado. Este é o pixel inferior da *seam* mínima. A *seam* é então rastreada até a linha superior do mapa de custo acumulado seguindo os pixels vizinhos com os menores custos cumulativos.

3. **Remova o caminho calculado.** Finalmente, precisamos remover os pixels na emenda computada da imagem.

Tente executar seu código em `porco.png`, `gatos.png` e `obra.png`, removendo 100 *seam* de cada.

### 9.3 Cores

Estenda seu código para trabalhar com imagens coloridas. Isso pode ser feito carregando imagens com `pillow` no modo “RGB” em vez do modo “L”. Neste modo, os pixels serão uma lista de tuplas  $(r, g, b)$ .

A maioria dos filtros descritos aqui podem ser implementados para imagens coloridas aplicando as versões em preto e branco aos canais vermelho, verde e azul separadamente e depois recombinação. A exceção notável é que a detecção de borda deve ser baseada na luminosidade, que pode ser calculada como:

$$L = \frac{299}{1000}R + \frac{587}{1000}G + \frac{114}{1000}B$$