UNIVERSIDADE FEDERAL DE RORAIMA
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
MESTRADO PROFISSIONAL EM MATEMÁTICA EM REDE NACIONAL - PROFMAT

FELIPE GOMES DA SILVA

FERRAMENTAS MATEMÁTICAS E ALGORITMOS PARA O PROCESSAMENTO DE IMAGENS DIGITAIS

Boa Vista, RR

2023

FELIPE GOMES DA SILVA

FERRAMENTAS MATEMÁTICAS E ALGORITMOS PARA O PROCESSAMENTO DE IMAGENS DIGITAIS

Dissertação apresentada ao Programa de Mestrado Profissional em Matemática em Rede Nacional - PROFMAT, da Sociedade Brasileira de Matemática - SBM e Universidade Federal de Roraima-UFRR, como parte dos requisitos para a obtenção do título de Mestre em Matemática.

Orientador: Prof. Dr. LINDEVAL F. DE LIMA

Boa Vista, RR

2023

Dados Internacionais de Catalogação na publicação (CIP)
Biblioteca Central da Universidade Federal de Roraima

S586f Silva, Felipe Gomes da.
Ferramentas matemáticas e algoritmos para o processamento de imagens digitais / Felipe Gomes da Silva. – Boa Vista, 2023.
89 f. : il.

Orientador: Prof. Dr. Lindeval Fernandes de Lima.
Dissertação (mestrado) – Universidade Federal de Roraima, Programa de Mestrado Profissional em Matemática.

1 – Matemática. 2 – Imagens. 3 – Filtros. 4 – Algoritmos. I – Título. II – Lima, Lindeval Fernandes de (orientador).

CDU – 519.688

Ficha Catalográfica elaborada pela Bibliotecária/Documentalista:
Mariede Pimentel e Couto Diogo - CRB-11-354 - AM

FELIPE GOMES DA SILVA

FERRAMENTAS MATEMÁTICAS E ALGORITMOS PARA O PROCESSAMENTO DE IMAGENS DIGITAIS

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Matemática - PROFMAT da Universidade Federal de Roraima-UFRR, como requisito para a obtenção do título de Mestre em Matemática. Defendida em 21 de julho de 2023 e avaliada pela seguinte banca examinadora.

Prof. Dr. LINDEVAL F. DE LIMA
Orientador - UFRR

Prof. Dr. Marcos André Fernandes Spósito
Membro - IFRR

Prof. Dr. Luciano Ferreira Silva
Membro - UFRR

Boa Vista, RR
2023

AGRADECIMENTOS

Agradeço primeiramente ao professor Dr. Lindeval Fernandes de Lima pela oportunidade de me orientar na conclusão deste trabalho, onde propiciou-me o primeiro contato com o Processamento de imagens digitais e com a interface gráfica do aplicativo criado.

Não posso deixar de agradecer ao meu pai, Luís Raimundo da Silva, por ensinar-me a nunca desistir e persistir mesmo com as adversidades. Deste modo aproveito para mencionar minha esposa Suelen Nascimento e minha filha Luna Felipa por está ao meu lado aliviando todo o estresse diário.

Agradeço também aos professores do programa PROFMAT que tiveram um olhar especial por eu ser o único aluno da turma, logo eles são responsáveis pela minha motivação de concluir esta jornada.

"Os dados são a espada do século XXI, e quem conseguir manejá-los, será o Samurai."

Jonathan Rosenberg

RESUMO

Com a crescente popularização das tecnologias digitais o ensino da Matemática tem a necessidade de adaptar-se e modificar-se, o Brasil tem atualmente mais de um celular por habitante, segundo levantamento anual divulgado pela FGV, logo grande parte da população já aplicou ou viu alguma foto com filtros. Partindo dessa premissa o presente trabalho buscou reunir ferramentas matemáticas e computacionais que explicasse como a matemática está ligada diretamente com a criação desses filtros. Nessa dissertação são apresentados ferramentas matemáticas como: matrizes, distâncias, funções, derivadas, vetores gradientes e transformadas de Fourier. Apresentamos o conceito de imagem digital como imagem matricial e mostramos a implementação de algoritmos tanto na construção de imagens pixel a pixel quanto na criação de filtros no domínio do espaço quanto no domínio das frequências. Finalizamos com a apresentação do aplicativo criado com o intuito de aproximar de maneira mais simples o professor ou estudante que queira ingressar nesse campo de processamento de imagens digitais.

Palavras-chave: Matemática, Imagens, Filtros, Algoritmos

ABSTRACT

Due to the growing popularization of digital technologies, the teaching of Mathematics needs to adapt and change. Currently, in Brazil, there is more than one cell phone per inhabitant, according to an annual survey released by FGV. Therefore, a large part of the population has already applied or seen a photo with filters. Taking this premise into account, the present work sought to gather mathematical and computational tools that explain how mathematics is directly linked to the creation of these filters. In this dissertation, mathematical tools such as matrices, distances, functions, derivatives, gradient vectors and Fourier transforms are presented. We present the concept of a digital image as a matrix image and show the implementation of algorithms both in the construction of pixel-by-pixel images and in the creation of filters in the space domain and in the frequency domain. We finish with the presentation of the application created with the objective of facilitating the approximation of the teacher or student who wants to enter this field of digital image processing.

Keywords: Mathematics, Images, Filters, Algorithms

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

A tecnologia no ensino de Matemática é um recurso que pode contribuir significativamente com a aprendizagem dos alunos, permitindo uma abordagem dinâmica e participativa dos conteúdos, o que foge do tradicional uso da lousa e do livro didático. Dessa forma, o uso de linguagens computacionais tem se mostrado como uma ferramenta que permite a construção de aplicativos de conceitos matemáticos de maneira mais interativa e compreensível (CUNHA, 2021).

A matemática é uma das matérias que mais tem se beneficiado das TICs (Tecnologias de Informação e Comunicação), uma vez que há muitas ferramentas computacionais voltadas para o estudo da matemática, como resolução de problemas, jogos e simulações, entre outras, que têm promovido processos de melhoria no ensino aprendizagem dos conteúdos.

A utilização de computadores e softwares em sala de aula tem facilitado o processo de transmissão de conhecimentos, possibilitando uma maior interatividade entre professores e alunos, além de oferecer recursos audiovisuais que auxiliam na compreensão de conceitos matemáticos mais complexos.

Em resumo, o uso de tecnologias no processo de ensino está possibilitando novos métodos de abordagem dos conteúdos matemáticos permitindo assim uma aprendizagem significativa para os educandos.

É quase impossível falar em tecnologias na educação sem antes falar de programação, fluxogramas e algoritmos pois é uma das habilidades citadas pela (BRASIL, 2018). É nesse contexto que a linguagem de computação Python tem sido amplamente utilizado no ensino da matemática por sua facilidade de uso e por possuir uma sintaxe simples e intuitiva. Sua capacidade de manipular dados e fazer cálculos numéricos o torna uma ferramenta útil para resolução de problemas matemáticos. Além de ser utilizado para diversas finalidades, como por exemplo, para a visualização de dados, resolução de problemas de otimização, simulações, entre outras aplicações. Ele permite a utilização de bibliotecas específicas para o ensino de matemática, como a NumPy, que permite a manipulação de arrays e a realização de cálculos matriciais. A utilização do Python na matemática também tem se mostrado eficaz para aprimorar a aprendizagem em outras disciplinas, como física e estatística, uma vez que é possível utilizar a programação para visualização e análise de dados.

Em síntese, o Python tem se destacado como uma ferramenta útil e versátil para a matemática, permitindo a resolução de problemas e a manipulação de dados de forma mais prática e interativa, o que pode contribuir significativamente para a

aprendizagem dos alunos.

O Processamento de Imagens Digitais (PDI) foi o campo de estudo onde usamos para aplicar tanto as ferramentas matemáticas quanto o estudo do pesamento computacional. O PDI lida com a aquisição, processamento, análise e interpretação de imagens digitais para extrair informações relevantes. O objetivo do PDI é melhorar a qualidade da imagem, facilitar sua interpretação e reconhecimento automático.

O PDI envolve várias etapas, incluindo pré-processamento, segmentação, extração de características e classificação. No pré-processamento, as imagens são limpas, normalizadas e corrigidas para remover ruídos e outros artefatos. Na segmentação, a imagem é dividida em várias partes com base em características como cor, textura e forma. A extração de características envolve a identificação de características específicas da imagem, como bordas, cantos e formas. Por fim, na classificação, a imagem é classificada em uma categoria específica com base em suas características.

Desta forma o PDI é amplamente utilizado em várias áreas, incluindo medicina, astronomia, segurança, automação industrial, entre outras. Na medicina, é usado para identificar tumores e outras anomalias em imagens de raios-x e ressonância magnética. Na astronomia, é usado para identificar estrelas, planetas e outros corpos celestes em imagens do espaço. Na segurança, é usado para reconhecimento facial e de placas de veículos em imagens de câmeras de segurança. Na automação industrial, é usado para inspeção de qualidade de produtos e controle de processos. Portanto, vemos a importância do processamento de imagens digitais para as mais diversas áreas da ciência e da tecnologia tanto no campo industrial com sua ampla aplicabilidade como na área da educação, desta forma, fica claro a importância de continuar pesquisando descobertas e melhorias para esta área tão importante que são as Tecnologias de Informação e Comunicação (TIC's).

1.1 Objetivos do Trabalho

O objetivo geral deste trabalho é de apresentar conteúdos e algoritmos relevantes para o ensino de matemática, alinhando conhecimentos matemáticos e computacionais. Usamos para alcançar esse objetivo usaremos a linguagem de programação python por ser amplamente utilizada por ser de livre utilização, todo professor e estuante será capaz de usá-la quando necessário.

Como objetivos específicos têm-se:

- Apresentar conteúdos (ferramentas) matemáticas importantes para criação de filtros de imagem digital.

- Apresentar algoritmos de implementação, na linguagem python, de filtros de imagem digital.

1.2 Estrutura do Trabalho

Esse trabalho encontra-se estruturado em seis capítulos.

Nesse capítulo abordamos a origem, os objetivos e a justificativa desse trabalho de pesquisa. Sua finalidade é introduzir a justificativa e a estrutura do trabalho.

No capítulo 2, apresentamos ferramentas matemáticas relevantes para a compreensão do estudo de processamento de imagens digitais.

No terceiro capítulo o foco é a imagem digital, apresentaremos os conceitos fundamentais, a construção e manipulação de imagens sintéticas.

O quarto capítulo dedica-se a apresentar os filtros tradicionais do processamento de imagens digitais e outros criados a partir da tentativa e erro.

No quinto capítulo apresenta-se um aplicativo criado com o intuito de explicar inicialmente a ligação dos filtros de imagens com a matemática.

Por fim, no último capítulo apresenta as conclusões obtidas através das pesquisas.

2 ELEMENTOS MATEMÁTICOS

2.1 Matrizes

A manipulação de dados, numéricos ou não, sempre foi interesse de estudo de diversos campos da ciência, sendo assim o estudo de matrizes tem seu papel de importância quando trata-se de agrupamento de dados.

2.1.1 Definição

Chama-se Matriz de ordem m por n um conjunto com $m \times n$ elementos (Números, funções, polinômios, vetores etc.), esses elementos são apresentados em m linhas e n colunas como mostrado abaixo:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ x_{31} & x_{32} & \dots & x_{3n} \\ x_{41} & x_{42} & \dots & x_{4n} \\ x_{51} & x_{52} & \dots & x_{5n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix}$$

Cada elemento da matriz X é identificado por uma dupla de índices i e j, assim identificamos o elemento x_{ij} o elemento de X que está na linha i e na coluna j.

2.1.2 Tipos de Matrizes

A matriz $[e]$ é uma matriz 1×1, ao passo que

$$\begin{bmatrix} 1 & 2 & 3 \\ 27 & 18 & 9 \end{bmatrix}$$

é uma matriz 2×3.

As matrizes se caracterizam em especial pelas suas dimensões ou pelos tipos de elementos em determinadas posições. Dependendo dos valores de m e n, uma matriz $m \times n$ recebe um nome especial.

Matriz Linha: Toda matriz $1 \times n$, ou seja $m = 1$. Também chamamos *vetor linha*. Por exemplo, a matriz

$$\begin{bmatrix} 8 & 2 & -7 \end{bmatrix}$$

é uma *matriz linha* 1×3.

Matriz Coluna: É aquela matriz do tipo $m \times 1$, ou seja onde $n = 1$. Também conhecida como *vetor coluna*. Por exemplo, a matriz

$$\begin{bmatrix} 0 \\ -3 \\ \sqrt{2} \end{bmatrix}$$

é uma *matriz coluna* 3×1.

Matrizes Quadradas: é aquela matriz onde temos $m = n$, ou seja, o número de linhas é igual ao número de colunas. Podendo também nos referir como *Matriz Quadrada de ordem* n

Dado as matrizes

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

e

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 2 & 3 \\ 5 & 8 & 13 & 21 \\ 34 & 55 & 89 & 144 \end{bmatrix}$$

temos que $\mathbf{A}$ é uma matriz quadrada de ordem 3 e $\mathbf{B}$ é uma matriz 3×4.

Considerando os índices, temos

Matriz Nula: Se $x_{ij} = 0 \; \forall i, j$, ou seja, quado todos os elementos da matriz são 0

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

2.1.2.1 Tipos Especiais de Matrizes Quadradas

Matriz Diagonal: É uma matriz quadrada onde $x_{ij} = 0 \; \forall \, i \neq j$. Dentro dessa definição se $x_{ij} = 1 \; \forall \, i = j$ então chamamos de ***Matriz identidade***. A matriz identidade de ordem n é representada por $\mathbf{I}_n$

$$\mathbf{I_3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Diagonal Principal: São os elementos x_{ij} de modo que $i = j$.

Diagonal Secundária: São os elementos x_{ij} de modo que $i + j = n + 1$.

Dada a matriz

$$\mathbf{D} = \begin{bmatrix} 1 & 1 & 2 & 3 \\ 5 & 8 & 13 & 21 \\ 34 & 55 & 89 & 144 \\ 233 & 377 & 610 & 987 \end{bmatrix}$$

a diagonal principal de **D** é $x_{11} = 1$, $x_{22} = 8$, $x_{33} = 89$ e $x_{44} = 987$

a diagonal secundária de **D** é $x_{41} = 233$, $x_{32} = 55$, $x_{23} = 13$ e $x_3 = 3$.

Matriz Triangular Superior: É uma matriz quadrada onde os elementos abaixo da diagonal principal são nulos, ou seja, $x_{ij} = 0$, sempre que $i < j$.

$$\mathbf{E} = \begin{bmatrix} 1 & 1 & 2 & 3 \\ 0 & 8 & 13 & 21 \\ 0 & 0 & 89 & 144 \\ 0 & 0 & 0 & 987 \end{bmatrix}$$

Matriz Triangular Inferior: É uma matriz quadrada onde os elementos acima da diagonal principal são nulos, ou seja, $x_{ij} = 0$, sempre que $i > j$.

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 5 & 8 & 0 & 0 \\ 34 & 55 & 89 & 0 \\ 233 & 377 & 610 & 987 \end{bmatrix}$$

Matriz Simétrica: É uma Matriz quadrada onde $x_{ij} = x_{ji}\ \forall\, i, j$. Por exemplo

$$\mathbf{G} = \begin{bmatrix} 1 & \sqrt{5} & 0 & -1 \\ \sqrt{5} & 2 & i & 8 \\ 0 & i & 3 & 10 \\ -1 & 8 & 10 & 4 \end{bmatrix}$$

Matriz Antissimétrica: É uma Matriz quadrada onde $x_{ij} = -x_{ji}\ \forall\, i, j$. Por exemplo

$$\mathbf{G} = \begin{bmatrix} 2 & 1 & 0 & 1 \\ -1 & 4 & 5 & 6 \\ 0 & -5 & 8 & 7 \\ -1 & -6 & -7 & 16 \end{bmatrix}$$

2.1.3 Operações com Matrizes

2.1.3.1 Adição

A soma de duas matrizes de mesma ordem, $\mathbf{A}_{m\times n} = [a_{ij}]$ e $\mathbf{B}_{m\times n} = [b_{ij}]$, é uma matriz $m \times n$, que denotamos por **A + B**,onde os elementos são as somas dos elementos correspondentes de **A** e **B**. Ou seja,

$$\mathbf{A + B} = [a_{ij} + bij]_{m\times n} \tag{2.1}$$

Propriedades: Dadas as Matrizes **A, B** e **C** de mesma ordem $m \times n$, temos:

- Comutatividade: **A + B = B + A**
- Associatividade: **(A + B) + C = A + (B + C)**
- Elemento Neutro: **A + 0 = A**, onde **0** é a matriz nula $m \times n$
- Simetria: dada a Matriz **A** $\exists$ a Matriz **-A** de modo que **A + (-A) = 0**

2.1.3.2 Multiplicação por Escalar

Se k é um número real ou complexo, o produto de uma matriz **A** = $[a_{ij}]$ por esse número é uma matriz **B** = $[b_{ij}]$, de modo que $b_{ij} = ka_{ij}$

Propriedades: Sejam k e k_1 números reais ou complexos e A e B matrizes, então as propriedades abaixo são válidas:

- Associatividade: (kk_1)**A**=k(k_1**A**)
- Distributividade: $(k + k_1)$**A** = k**A**+k_1**A** e k(**A** + **B**) = k**A**+k**B**
- 1**A**= **A**
- 0**.A = 0**, ou seja, se multiplicarmos uma matriz por 0 obtemos a matriz nula.

2.1.3.3 Transposição

Dada uma matriz $\mathbf{A}_{m\times n} = [a_{ij}]$, podemos obter uma outra matriz $\mathbf{A}^t_{n\times m} = [b_{ij}]$, ou seja, as linhas de **A** são as colunas de $\mathbf{A}^t$

Propriedades:

- Uma matriz é simétrica se e somente se **A = A**^{t}
- $(\mathbf{A}^t)^t = A$, ou seja, a transposta de uma transposta de uma matriz é ela mesma.
- $(\mathbf{A} + \mathbf{B})^t = \mathbf{A}^t + \mathbf{B}^t$, a transposta de uma soma é a soma das transpostas
- $(k\mathbf{A})^t = k\mathbf{A}^t$

2.1.3.4 Multiplicação de Matrizes

Sejam **A** = $[a_{ij}]_{m\times n}$ e **B** = $[b_{rs}]_{n\times p}$. Definimos **AB** = $[c_{uv}]_{m\times p}$ onde

$$c_{uv} = \sum_{k=1}^{n} a_{uk}b_{kv} = a_{u1}b_{1v} + a_{u2}b_{2v} + ... + a_{un}b_{nv} \tag{2.2}$$

é importante ressaltar que a *Multiplicação entre matrizes* só é possível quando o número de linhas da primeira matriz é igual ao número de colunas da segunda matriz, ou seja, de maneira geral a *Multiplicação entre matrizes* não é *Comutativa*. Além disso a matriz resultante **C = AB** é de ordem $m \times p$ sempre que **A** = $[a_{ij}]_{m\times n}$ e **B** = $[b_{rs}]_{n\times p}$

Veja o exemplo

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 4 \\ 0 & 1 & 2 \\ 1 & -1 & 1 \end{bmatrix} \qquad \text{e} \qquad \mathbf{B} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 3 \end{bmatrix}$$

Portanto, a multiplicação de **A** por **B** é:

$$\mathbf{C} = \begin{bmatrix} 9 & 14 \\ 4 & 7 \\ 3 & 2 \end{bmatrix}$$

Propriedades Sejam **A**, **B** e **C** matrizes que cumpram as condições da multiplicação

- Em geral **AB** $\neq$ **BA**

- Elemento Neutro: **AI = IA = A**, onde **I** é a matriz identidade e A uma matriz quadrada
- Matriz Nula: **A0 = 0A = 0**
- Associatividade:(**AB**)**C** = **A**(**BC**)
- Distributiva à direita (**A+B**)**C** = **AC + BC**
- Distributividade à esquerda: **A**(**B + C**) = **AB + AC**
- Transposição: $(\mathbf{AB})^t = \mathbf{B}^t\mathbf{A}^t$

2.2 Distâncias - Métricas

Segundo (LIMA, 2013) uma *métrica* num conjunto M é uma função $d : M \times M \to \mathbb{R}$, que associa a cada par ordenado de elementos $x, y \in M$ um número real $d(x, y)$, chamado distância de x a y, de modo que sejam satisfeitas as seguintes condições para quaisquer $x, y, z \in M$:

d1) $d(x, x) = 0$

d2) Se $x \neq y$ então $d(x, y) > 0$

d3) $d(x, y) = d(y, x)$

d4) $d(x, z) \leq d(x, y) + d(y, z)$

Nesse presente trabalho focaremos nossos conceitos a utilização a um conjunto $M \subset \mathbb{N} \times \mathbb{N}$ de modo que possamos usar nos elementos de uma matriz qualquer.

2.2.1 Distância Euclidiana

No conjunto dos números reais $\mathbb{R}$ a distância entre dois pontos $x, y \in \mathbb{R}$ é dado por $d(x, y) = |x - y|$. Usando essa definição podemos definir a distância entre dois pontos $P = (a, b)$ e $Q = (c, d)$ pertencentes a $\mathbb{R}^2$ como sendo

$$d(P, Q) = \sqrt{(a - c)^2 + (b - d)^2} \tag{2.3}$$

Com essa medida de distância todo elemento com uma distância menor ou igual a um certo r em relação (x, y), são os pontos contidos em uma circunferência de raio r e centrado em (x, y)

2.2.2 Distância D_4 (city-block)

Essa distância é chamada de *não-euclidiana* pois ela se baseia em uma somatória de distâncias perpendiculares, enquanto a distância *Euclidiana* é a hipotenusa de um triângulo retângulo essa distância é o somatório dos catetos de um triângulo retângulo. Ou seja,

$$d_4(P,Q) = |a-c| + |b-d| \tag{2.4}$$

onde | . | denota módulo (ou valor absoluto). Neste caso, os pontos tendo uma distância d_4 em relação a (x,y) menor ou igual a algum valor r formam um losango centrado em (x,y).

2.2.3 Distância D_8 (tabuleiro de xadrez)

Podemos adotar também como distância a maior diferença em módulo das respectivas coordenadas dos pontos, ou seja

$$d(P,Q) = max(|a-c|, |b-d|) \tag{2.5}$$

onde max é um operador que devolve o maior valor dentre um conjunto de valores entre parênteses. Neste caso os pontos com distância D8 em relação a (x,y) menor ou igual a algum valor r formam um quadrado centrado em (x,y).

Na seção seguinte abordaremos o conceito de função, função contínua, limites de função e derivadas.

2.3 Funções

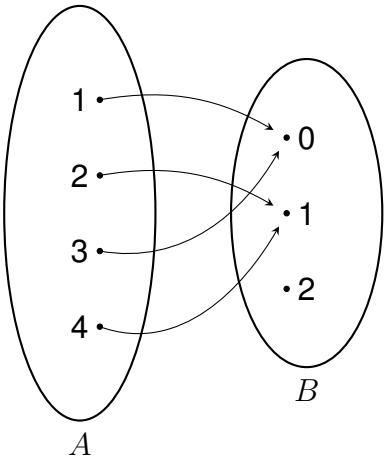
Definição:

Sejam A e B conjuntos numéricos. Uma função $f : A \to B$ é uma lei que associa a cada elemento de A um único elemento de B. O conjunto A chamamos de *Domínio de f* e B chamaremos de *contra-domínio de f*

Exemplos

Exemplo 1: Sejam $A = \{1,2,3,4\}$ e $B = \{0,1,2\}$. $f : A \to B$ onde $f(x) = \begin{cases} 1; \textit{se x é par} \\ 0; \textit{se x é impar} \end{cases}$

Deste modo temos:



Seja **A**=$[a_{ij}]_{2\times2}$ onde $a_{ij} = i + j$ e **B**=$[b_{ij}]_{2\times2}$.

Exemplo 2: Seja $f : A \to B$ onde $f(x) = x^2$ para $x \in A$, deste modo temos

$$\mathbf{A=}\begin{vmatrix} 2 & 3 \\ 3 & 4 \end{vmatrix} \rightarrow \mathbf{B = f(A) =} \begin{vmatrix} f(2) & f(3) \\ f(3) & f(4) \end{vmatrix} = \begin{vmatrix} 4 & 9 \\ 9 & 16 \end{vmatrix}$$

2.3.1 Definição de Função Contínua

Sejam f uma função real e p um ponto de seu domínio. Definiremos: f é contínua em p se, e somente se para todo $\epsilon > 0$ dado, existe $\delta > 0$ (δ depende de ϵ), tal que, para todo $x \in D_f$, onde

$$p - \delta < x < p + \delta \Rightarrow f(p) - \epsilon < f(x) < f(p) + \epsilon \tag{2.6}$$

2.3.2 Definição de Limite

Sejam f uma função e p um ponto do domínio de f ou extremidade de um dos intervalos que compõem o domínio de f. Dizemos que f tem limite L, em p, se, para todo $\epsilon > 0$ dado, existir um $\delta > 0$ tal que, para todo $x \in D_f$,

$$0 < |x - p| < \delta \Rightarrow |f(x) - L| < \epsilon$$

Tal número L, que quando existe é único, será indicado por $\lim_{x\to p} f(x)$. Assim

$$\lim_{x\to p} f(x) = L \Leftrightarrow \begin{cases} \forall\epsilon > 0, \exists\, \delta > 0\, tal\, que,\ para\, todo\, x \in D_f \\ 0 < |x - p| < \delta \Rightarrow |f(x) - L| < \epsilon \end{cases} \tag{2.7}$$

Observação: f é contínua em p se, e somente se, limite de $f(x)$ quando x tende a p é igual a $f(p)$

2.3.3 Derivadas

2.3.3.1 Definição de Derivada

Sejam f uma função e p um ponto de seu domínio. O limite

$$\lim_{x \to p} \frac{f(x) - f(p)}{x - p} \tag{2.8}$$

quando existe e é finito, denomina-se ***Derivada*** de f em p e indica-se por $f^{'}(p)$ Se f adimite derivada em p, então diremos que f é derivável em p ou diferenciável em p.

Fazendo $h = x - p$ temos que

$$f^{'}(p) = \lim_{h \to 0} \frac{f(h + p) - f(p)}{h} \tag{2.9}$$

Poderemos usar as notações $f^{'}(x)$ ou $\frac{\partial}{\partial x} f(x)$ para representar a derivada da função f em relação a x.

2.3.4 Funções de Duas Variáveis

Uma função f de duas variáveis é uma regra que associa a cada par ordenado de números reais (x, y) de um conjunto D um único valor real, denotado por $f(x, y)$. O conjunto D_f é o domínio de f e sua imagem é o conjunto de valores possíveis de f, ou seja, $Im\, f = \{f(x, y) \in \mathbb{R} | (x, y) \in D_f\}$.

2.3.4.1 Derivadas Parciais

Se f é uma função de duas variáveis, suas derivadas parciais são as funções f_x e f_y definidas por

$$f_x(x, y) = \lim_{h \to 0} \frac{f(x + h, y) - f(x, y)}{h} \tag{2.10}$$

$$f_y(x, y) = \lim_{h \to 0} \frac{f(x, y + h) - f(x, y)}{h} \tag{2.11}$$

2.3.4.2 Notações para Derivadas Parciais

$$f_x(x, y) = f_x = \frac{\partial f}{\partial x} = \frac{\partial}{\partial x} f(x, y) = f_1 = D_1 f = Dx f$$

$$f_y(x, y) = f_y = \frac{\partial f}{\partial y} = \frac{\partial}{\partial y} f(x, y) = f_2 = D_2 f = Dy f$$

2.3.5 Vetor Gradiente

Seja $f(x, y)$ uma função que adimite derivadas parciais no ponto $p = (x_0, y_0)$. O vetor

$$\nabla f(x_0, y_0) = \left(\frac{\partial}{\partial x} f(x_0, y_0), \frac{\partial}{\partial y} f(x_0, y_0) \right) \tag{2.12}$$

é chamado **Vetor Gradiente**

2.4 Transformada de Fourier

A transformada de Fourier é comumente usada para converter um sinal no tempo em um espectro de frequência. Esse método utiliza o fato de que toda função não-linear pode ser representada como uma soma de ondas senoidais (infinitas). A transformada separa um sinal de tempo e retorna informações sobre a frequência de todas as ondas senoidais necessárias para simular o sinal de tempo.

2.4.1 Série de Fourier

Segundo (FIGUEIREDO, 2018), qualquer função periódica pode ser decomposto em uma soma de ondas senoides e cossenoides. Nelas, a frequência de cada senoide é um múltiplo inteiro da frequência fundamental da onda distorcida, denominados "harmônicos". A função $f(x)$ periódica pode ser expressa como a série trigonométrica de Fourier:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \cos \frac{n\pi x}{L} + b_n sen \frac{n\pi x}{L} \right) \tag{2.13}$$

sendo os coeficientes de Fourier:

$$a_0 = \frac{1}{L} \int_{-L}^{L} f(x) dx \tag{2.14}$$

$$a_n = \frac{1}{L} \int_{-L}^{L} f(x) \cos \frac{n\pi x}{L} dx \tag{2.15}$$

$$b_n = \frac{1}{L} \int_{-L}^{L} f(x) sen \frac{n\pi x}{L} dx \tag{2.16}$$

Onde $n \in \mathbb{N}$ e L é o período da função. A equação (2.13) é conhecida com equação de síntese e as equações (2.14),(2.15) e (2.16) são conhecidas como coeficientes da série trigonométrica de fourier. Quando a função possui valores complexos é comum usar a ***série exponencial (ou complexa) de Fourier***. Assim podemos escrever uma função $f(x)$ como:

$$f(x) = \sum_{k=-\infty}^{\infty} c_n e^{\frac{in\pi x}{L}} \tag{2.17}$$

sendo

$$c_n = \frac{1}{2L}\int_{-L}^{L} f(x)e^{\frac{-i\pi nx}{L}}dx,\ para\, n \in \mathbb{Z} \tag{2.18}$$

onde $i = \sqrt{-1}$ Enquanto que a série de Fourier é usada para representar uma função periódica por uma soma discreta de exponenciais complexas (ou senos e cossenos), a transformada de Fourier é usada para representar uma função geral (não necessariamente periódica) por uma superposição contínua ou integral de exponenciais complexos. Dessa forma, a série pode ser representada no domínio das frequências.

2.4.2 Transformadas de Fourier

2.4.2.1 Transformada de Fourier para funções contínuas de uma variável

Dada uma função $f : \mathbb{R} \to \mathbb{R}$, definiremos como sua ***Transformada de Fourier***(FT) a expressão

$$\mathcal{T}\{f(x)\} = F(u) = \int_{-\infty}^{\infty} f(x)e^{-i2\pi ux}dx \tag{2.19}$$

Dado $F(u)$, $f(x)$ pode ser obtido através da transformada inversa de Fourier (IFT)

$$\mathcal{T}^{-1}\{F(u)\} = f(x) = \int_{-\infty}^{\infty} F(u)e^{i2\pi ux}du \tag{2.20}$$

As equações (2.19) e (2.20) são chamadas de par de Fourier, esse par exite se $f(x)$ for contínua e integrável e $F(u)$ for integrável.

Podemos observar que da forma que foi definida a FT de uma função real é um número complexo da forma

$$F(u) = R(u) + I(u)i \tag{2.21}$$

onde $R(u)$ e $I(u)$ são, respectivamente, a parte real e a parte imaginária da função $F(u)$. Na forma exponencial podemos representar $F(u)$ por

$$F(u) = |F(u)|e^{i\phi(u)} \tag{2.22}$$

onde temos um componente de magnitude $|F(u)|$ que é denominado *espectro de Fourier* e $\phi(u)$ seu ângulo de fase.

2.4.2.2 Transformada de Fourier para funções contínuas de duas variáveis

O conceito de FT e IFT para funções de duas variáveis é semelhante as de uma variável, se $f(x, y)$ é contínua e integrável e $F(u, v)$ é integrável então o par de Fourier

a seguir existe:

$$\mathcal{T}\{f(x,y)\} = F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-i2\pi(ux+vy)}dxdy \tag{2.23}$$

$$\mathcal{T}^{-1}\{F(u,v)\} = f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)e^{i2\pi(ux+vy)}dxdy \tag{2.24}$$

Da mesma forma temos que $F(u,v)$ é complexa e a sua amplitude $|F(u,v)|$ é denominado como espectro de Fourier.

2.4.2.3 Transformada de Fourier Discreta para funções de uma variável

Dada uma função contínua $f(x)$, seja a sequência $\{f(x_0), f(x_0+\Delta x), f(x_0+2\Delta x), ..., f(x_0+[N-1]\Delta x)\}$ obtida através de uma discretização de N pontos da função f, igualmente espaçados por Δx.Denominaremos $f(n)$ de função discreta de $f(x)$, deste modo a Transformada Discreta de Fourier (DFT) de $f(n)$ será:

$$F(u) = \frac{1}{N}\sum_{n=0}^{N-1} f(n)e^{\frac{-i2\pi un}{N}} \tag{2.25}$$

para $u = 0, 1, 2..., N-1$ e a IFT de $F(u)$ será:

$$f(n) = \frac{1}{N}\sum_{u=0}^{N-1} f(n)e^{\frac{i2\pi un}{N}} \tag{2.26}$$

para $n = 0,\ 1,\ 2, ...,\ N-1$.

Os valores $u = 0, 1, 2..., N-1$ na DFT da (2.25) correspondem a amostras dos valores da transformada do sinal contínuo nos pontos $0, \Delta u, 2\Delta u, ..., (N-1)\Delta u$. Em outras palavras, $F(u)$ representa $F(u\Delta u)$. Portanto, os intervalos de espaçamento entre as amostras do sinal e de sua transformada estão relacionados através da expressão

$$\Delta u = \frac{1}{N\Delta x} \tag{2.27}$$

2.4.2.4 Transformada de Fourier Discreta para funções de duas variáveis

Seja $f(x,y)$ uma função contínua de duas variáveis discretizada em M pontos de x e N pontos de y. Deste modo temos que a DFT será:

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-i2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \tag{2.28}$$

para $u = 0,\ 1,\ 2, ...,\ M-1$ e $v = 0,\ 1,\ 2, ...,\ N-1$. A IFT nesse caso é dada por:

$$f(x,y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) e^{i2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \tag{2.29}$$

O processo de discretização da função bidimensional contínua f(x,y) pode ser visto como uma grade 2-D, com impulsos unitários espaçados de Δx e Δy, nos eixos x e y respectivamente. A função $f(x,y)$ discretizada (utilizada nas (2.28) e (2.29)) representa as amostras de f(x,y) contínua original em pontos espaçados entre si de Δx e Δy, nos respectivos eixos. Os intervalos de amostragem nos domínios espacial e da frequência estão relacionados entre si por:

$$\Delta u = \frac{1}{M\Delta x} \tag{2.30}$$

e

$$\Delta v = \frac{1}{N\Delta y} \tag{2.31}$$

Para um caso em particular onde $M = N$, as equações (2.28) e (2.29) poderão ser reescrita como:

$$F(u,v) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) e^{-i2\pi\left(\frac{ux}{N}+\frac{vy}{N}\right)} \tag{2.32}$$

e

$$f(x,y) = \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u,v) e^{i2\pi\left(\frac{ux}{N}+\frac{vy}{N}\right)} \tag{2.33}$$

Note que $\frac{1}{MN}$ da equação (2.28) foi desmembrado para $\frac{1}{N}$ em cada uma das equações mas esse desmembramento é arbitrário.

3 IMAGENS

Uma imagem é uma representação visual de um objeto, pessoa ou coisa, criada por meio de diferentes meios, como pintura, escultura, desenho, gravura, fotografia ou outros meios das artes visuais. Pode ser uma figura formada pelo conjunto de pontos onde convergem os raios provenientes de determinadas fontes, graças à sua interação com um sistema óptico, sendo assim uma imagem óptica. (FILHO; NETO, 1999)

Além disso, uma imagem também pode ser uma reprodução visual de um objeto, obtida por meio de um espelho, um instrumento de óptica ou outros meios. Pode ser uma pequena estampa que representa um assunto religioso ou qualquer outro tema.

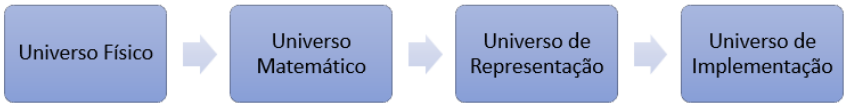
As imagens têm uma ampla variedade de usos e aplicações em diferentes áreas, como arte, comunicação, publicidade, ciência, medicina, entre outras. Elas são utilizadas para transmitir informações visuais, expressar emoções, comunicar ideias e contar histórias. As imagens podem ser capturadas, criadas ou modificadas de diversas maneiras, dependendo do meio e do propósito específico.

Em matemática aplicada necessitamos modelar esse conceito de imagem usando uma hierarquia de abstrações descrita em um paradigma apresentado em (GOMES; VELHO, 2015), a saber necessitamos estabelecer 4 universos (conjuntos): o universo físico F, o universo matemático M, o universo de representação R e o universo de implementação

Segundo (GOMES; VELHO, 2015) o *universo físico* contém os objetos do mundo real que pretendemos estudar, o *universo matemático* contém uma descrição abstrata dos objetos do mundo físico, o *universo da representação* é construído por descrições simbólicas e finitas associadas do universo matemático e no *universo de implementação* associamos as descrições criadas no universo de representação com estruturas de dados, com finalidade de obter uma representação do objeto no computador.

O processo de criação de imagem digital percorre todo esse paradigma pois a imagem, do ponto de vista do universo matemáticos é um objeto contínuo mas nossa

Figura 1 – Paradigma dos 4 universos de abstração.



Fonte: Imagem adaptada do livro (GOMES; VELHO, 2015).

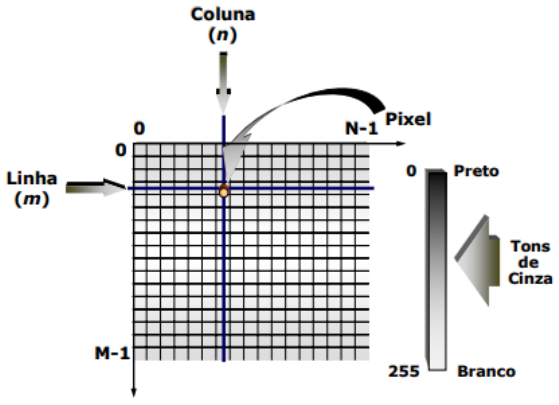
representação e implementação computacional é idealizado através do conceito de *Imagem Matricial* que seria a decomposição em uma união de pontos discretos de cor única, ou seja, para a implementação no computador a imagem é tratada como uma matriz de dados.

3.1 Imagens Sintéticas

3.1.1 Modelo de Imagem Digital

Uma imagem monocromática é uma função bidimensional contínua $f(x, y)$, na qual x e y são coordenadas espaciais e o valor de f em qualquer ponto (x,y) é proporcional à intensidade luminosa (brilho ou nível de cinza) no ponto considerado, (FILHO; NETO, 1999). Como os computadores não são capazes de processar imagens contínuas, mas apenas arrays de números digitais, é necessário representar imagens como arranjos bidimensionais de pontos. Cada ponto na grade bidimensional que representa a imagem digital é denominado elemento de imagem ou pixel. Na figura 2 apresenta-se a notação matricial usual para a localização de um pixel no arranjo de pixels de uma imagem bidimensional. O primeiro índice denota a posição da linha, m, na qual o pixel se encontra, enquanto o segundo, n, denota a posição da coluna. Se a imagem digital contiver M linhas e N colunas, o índice m variará de 0 a M-1, enquanto n variará de 0 a N-1. Observe-se o sentido de leitura (varredura) e a convenção usualmente adotada na representação espacial de uma imagem digital.

Figura 2 – Representação Matricial da imagem digital.



Fonte: Desenvolvida por (FILHO; NETO, 1999).

É importante ressaltar que dependendo da biblioteca usada pela linguagem de programação para tratar imagens, a localização do elemento a_{ij} é dado como uma função de duas variáveis $f(x, y)$ onde x representa a quantidade de colunas e y

representa a quantidade de linhas, ou seja, de modo geral ou trataremos a imagem como uma matriz A ou como a transposta da matriz A.

3.1.2 Modelo de imagem com 1 canal

Os canais de imagens digitais referem-se aos meios pelos quais as imagens digitais são armazenadas, transferidas, exibidas e processadas. Deste modo, quando falamos que uma imagem possui apenas um canal, significa que cada pixel $f(x, y)$ terá um valor numérico inteiro tal que $0 \leq f(x, y) \leq 255$.

Esses valores correspondem a luminância no pixel ou escala de cinza, onde 0 corresponde a cor preta e 255 corresponde a cor branca. Arquivos PGM (Portable Gray Map) armazenam imagens 2D em escala de cinza. Cada pixel na imagem contém só um ou dois bytes de informação (8 ou 16 bits). Talvez não pareça muita coisa, mas os arquivos PGM podem armazenar dezenas de milhares de tons, de preto a branco, com todos os tons de cinza entre eles. Uma das vantagens das imagens PGM é a sua flexibilidade de uso. Esses arquivos podem ser utilizados como camadas em outras imagens, permitindo combinar diferentes elementos visuais de forma simples e eficiente. Além disso, devido ao seu tamanho compacto, os arquivos PGM são fáceis de armazenar e podem ser inseridos em documentos do Word ou outros tipos de arquivos sem aumentar significativamente o tamanho total. Dentro da Biblioteca ***Pillow*** o sistema usado para trabalhar com imagens PGM é o sistema **L** que referi-se a luminância do pixel. A figura 3 é um exemplo de imagens PGM que iremos trabalhar muitos dos nossos algoritmos.

Figura 3 – Lena em formato PGM.

Fonte: Disponível em https://people.sc.fsu.edu/

3.1.3 Modelo de imagem com 3 canais

Representações mais modernas de imagem utilizam 3 canais para representar as cores do pixel. Nesse trabalho será adotado o sistema de cores RGB, abreviação de "Red, Green, Blue"(Vermelho, Verde, Azul), é um sistema de cores aditivas utilizado na reprodução de cores em dispositivos eletrônicos, como monitores de TV, computadores, mídias digitais e celulares. Ele constrói todas as cores a partir da combinação dessas três cores primárias.

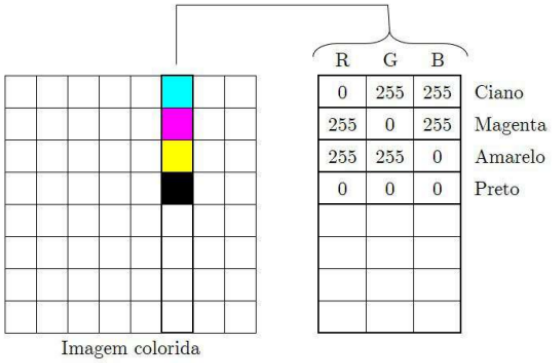
No sistema RGB, cada uma das cores primárias - vermelho, verde e azul - é representada por um valor de 8 bits, variando de 0 a 255. Esses valores determinam a intensidade luminosa de cada cor e, quando combinados em diferentes proporções, permitem a reprodução de uma ampla gama de cores. Com 8 bits para cada cor, o sistema RGB pode representar um total de 16.777.216 cores possíveis $(256 * 256 * 256)$.

No caso das imagens coloridas, ao invés de um valor numérico de cada pixel, utilizamos uma tripla ordenada; A estrutura dessa tripla depende de onde essa imagem está inserida. Sendo mais específico: nos aparelhos emissores de luz (televisão, projetores, etc) são utilizados frequentemente o padrão RGB, mas quando estas precisam ser utilizadas em dispositivos que não emitem luz (como impressoras, por exemplo) o computador converte para o padrão CMYK (Cyan, Magenta, Yellow, Black) (IGNACIO, 2013). Após avanços no sistema de captura de imagens digitais, novos padrões de cores foram formados, como o HSL (Tonalidade, Saturação e Luminância) e o HSV (Matiz, Saturação e Valor) para citar alguns.(IGNáCIO, 2013).

Sendo assim um pixel $f(x, y)$ como dito anteriormente será representado por uma tupla de 3 valores, ou seja, $f(x, y) = (R, G, B)$ onde $0 \leq R, G, B \leq 255$

.

Figura 4 – Representação Matricial RGB.



Fonte: (GONZALES; WOODS, 2010).

Na Figura 5 é mostrado como a união dos 3 canais de cores produz uma imagem

com as cores próximas o suficiente para representar a realidade.

Figura 5 – Separando os canais de uma imagem colorida RGB.

Fonte: O autor.

3.1.4 Construção pixel a pixel

Nessa sessão apresentaremos a criação de imagens pixel a pixel, ou seja, trabalharemos alguns algoritmos matemáticos para criar formas usando as cores disponíveis no sistema RGB.

O primeiro que apresentaremos é simplesmente uma imagem sem cor definida, como mostrado no exemplo abaixo:

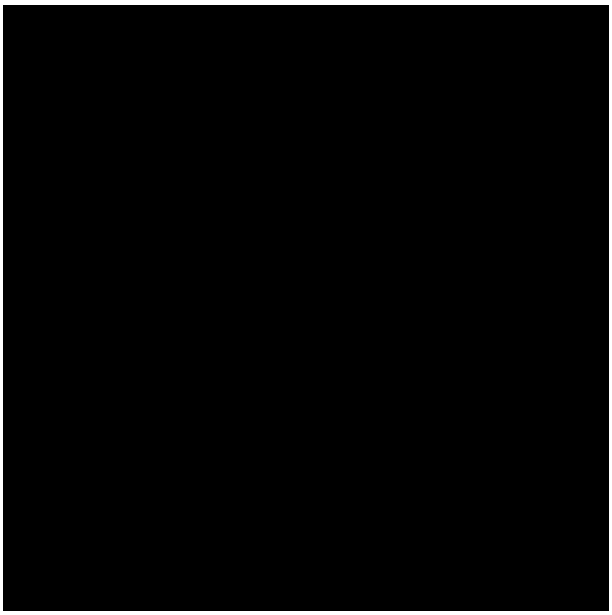
```
from PIL import Image

imagem = Image.new('RGB',(100,100))

imagem.show()
```

Observe que usamos a biblioteca Pillow (PIL) e importamos dela a clase Image e usamos a função new() para criar uma matriz bidimensional que receberá uma cor informada, como não informamos a cor, a imagem gerada por *imagem = Image.new('RGB',(100,100))* e apresentada por *imagem.show()* é:

Para modificar a cor desse quadrado de 100 pixel lado basta definir a cor com 3 coordenadas RGB nesse caso parra ilustração criaremos um quadrado azul. No modelo de cor RGB a composição fica de 0% vermelho, 0% verde e 100% azul, ou seja, (0,0,255), além disso ampliaremos a imagem para 200 pixel´s de lado. observe:

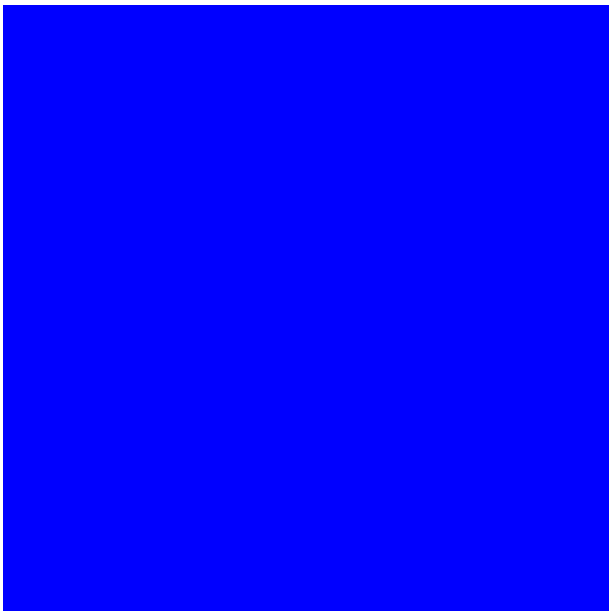
```
from PIL import Image

AZUL = (0,0,255)
image = Image.new('RGB',(100,100), AZUL)

image.show()
```

Figura 6 – Imagem sem cor.



Fonte: O autor.

Figura 7 – Imagem Azul.



Fonte: O autor.

Assim, podemos usar a criatividade para manipular esses dados. Por exemplo, podemos criar um triângulo usando a definição de matriz triangular como sugere o algoritmo abaixo:

```
from PIL import Image

def triangulo(size):
    WHITE = (255,255,255)
    BLACK = (0,0,0)
    image = Image.new("RGB",(size,size), WHITE)
    for x in range(size):
        for y in range(size):
            if x<y:
                image.putpixel((x,y),BLACK)
    return image


if __name__ == "__main__":
    t=triangulo(700)
    t.show()
```

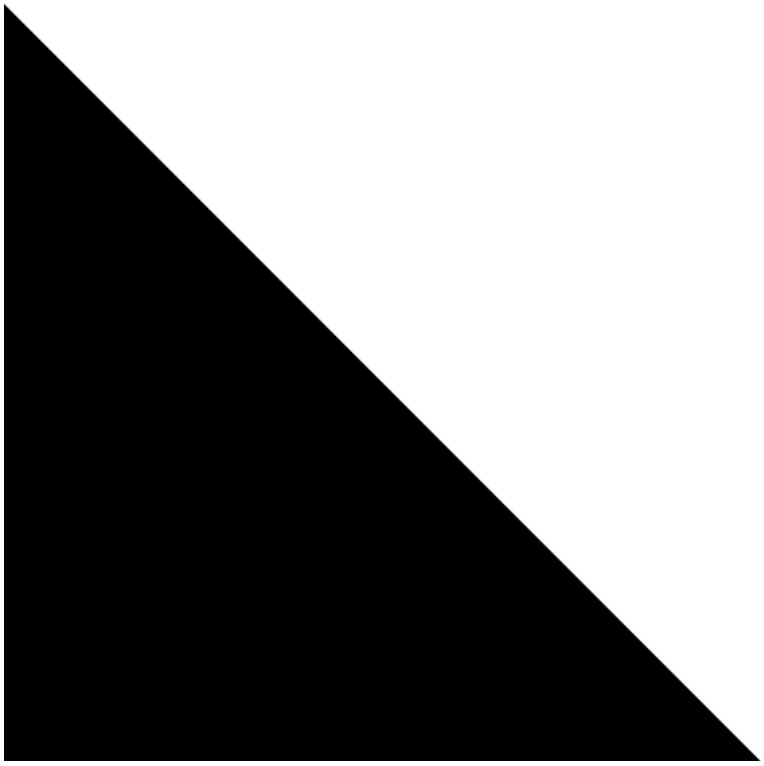
Nesse algoritmo definimos uma função chamada *triangulo* que recebe um valor numérico que determinará a dimensão da imagem quadrada, após isso definimos duas tuplas de 3 entradas para representar numericamente nossas cores branca (*WHITE = (255,255,255)*) e preta (*BLACK = (0,0,0)*). O algoritmo segue criando uma matriz quadrada com todos os pixel brancos com os laços de repetição percorrendo toda a matriz e "pintando"cada pixel caso a primeira coordenada seja maior que a segunda, isso é possível usando a função *putpixel((x,y),"cor desejada")* .Deste modo a imagem exibida no monitor será apresentada na figura 8

Usando conceitos geométricos podemos criar circunferências, elipses e hipérboles.

3.1.4.1 Circunferência

Para construir nossa circunferência precisamos determinar seus elementos principais como centro e o raio. O centro faremos com que seja localizado no meio da imagem ou seja o pixel que fique exatamente na divisão inteira da dimensão da imagem por 2, o raio por sua vez faremos com que seja $\frac{1}{4}$ do lado pois nesse primeiro momento estamos considerando uma imagem quadrada mas poderíamos considerar um raio diferente (essas escolhas são arbitrárias). Agora basta percorrer todos os pixel e pintar aqueles que estão a uma distância euclidiana menor que o raio, ou seja, um pixel (x, y) receberá a cor preta se $(x - x_0)^2 + (y - y_0)^2 < (raio)^2$ onde (x_0, y_0) é o centro da circunferência. Observe o algoritmo abaixo:

Figura 8 – Triângulo.

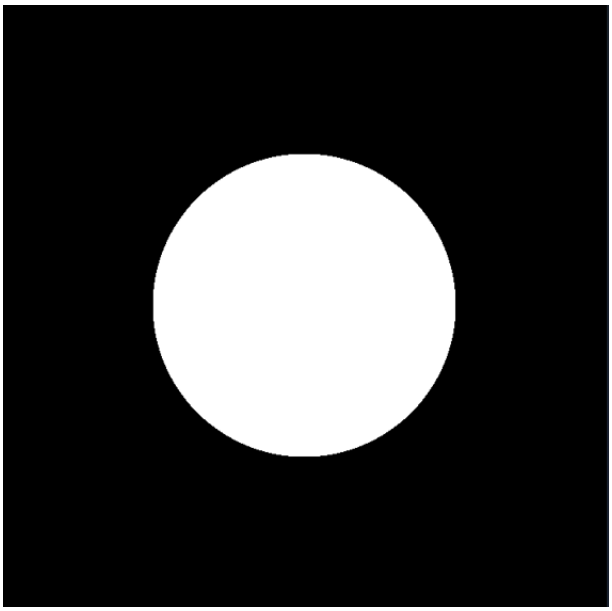


Fonte: O autor.

```
from PIL import Image

def circulo(size):
    WHITE = (255, 255, 255)
    BLACK = (0, 0, 0)
    image = Image.new("RGB", (size, size), BLACK)

    x0=size//2
    y0=size//2
    raio = size//4
    for x in range(size):
        for y in range(size):
            if (x-x0)**2+(y-y0)**2<(raio)**2:
                image.putpixel((x,y), WHITE)
    return image
```

```
if __name__ == "__main__":
    circulo = circulo(700)
    circulo.show()
```

Figura 9 – Círculo.



Fonte: O autor.

3.1.4.2 Elipse

Uma **elipse** ε de **focos** F_1 e F_2 é o conjunto de pontos P do plano cuja a soma das distâncias a F_1 e F_2 é igual a uma constante $2a > 0$, maior do que a distância entre os focos $2c \geq 0$. Ou seja, sendo $0 \leq c \leq a$ e $d(F_1, F_2) = 2c$

A **equação geral da elipse** é

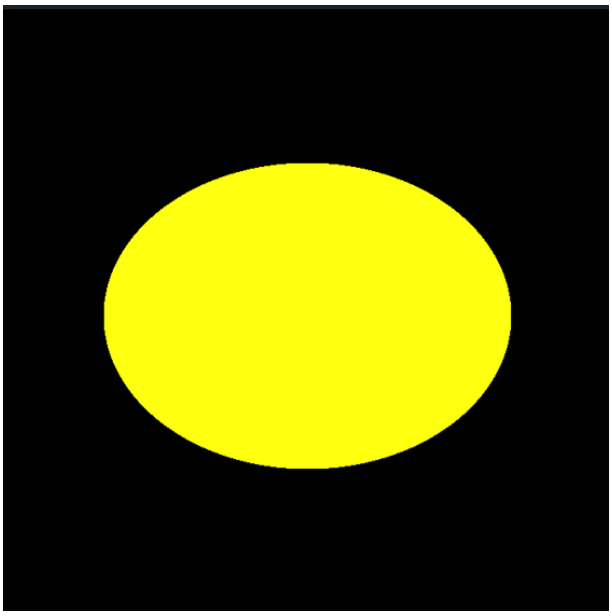
$$\frac{(x - x_0)^2}{a^2} + \frac{(y - y_0)^2}{b^2} = 1 \tag{3.1}$$

onde (x_0, y_0) é o ponto médio entre os dois focos em um eixo horizontal e $b^2 = a^2 - c^2$

Usaremos esse conceito para construir a imagem de uma elipse com interior preenchido pela cor amarelo que no sistema **RGB** fica $(255, 255, 0)$. A seguir é apresentado a implementação do código em python

```
from PIL import Image

def elipse(size):
    amarelo = (255,255,16)
    preto = (0, 0 , 0)
    img = Image.new("RGB",(size,size), preto)

    x0 = size // 2
    y0 = size // 2
    a = size//3
    b = size//4

    for x in range(size):
        for y in range(size):
            if ((x-x0)**2)/(a)**2+((y-y0)**2)/(b)**2<1:
                img.putpixel((x,y), amarelo)
    return img


if __name__ == "__main__":
    elipse = elipse(700)
    elipse.show()
```

Figura 10 – Elipse.



Fonte: O autor

3.1.4.3 Hipérbole

Uma **hipérbole** ω de dois **focos** F_1 e F_2 é o conjunto de pontos P do plano para os quais o módulo da diferença de suas distâncias euclidianas a F_1 e a F_2 é igual a constante $2a > 0$, menor que a distância entre os focos $2c > 0$:

$$\omega = \{P; |d(P, F_1) - d(P, F_2)| = 2a\},\ 0 < a < c,\ d(F_1, F_2) = 2c \tag{3.2}$$

A **equação geral da hipérbole** é

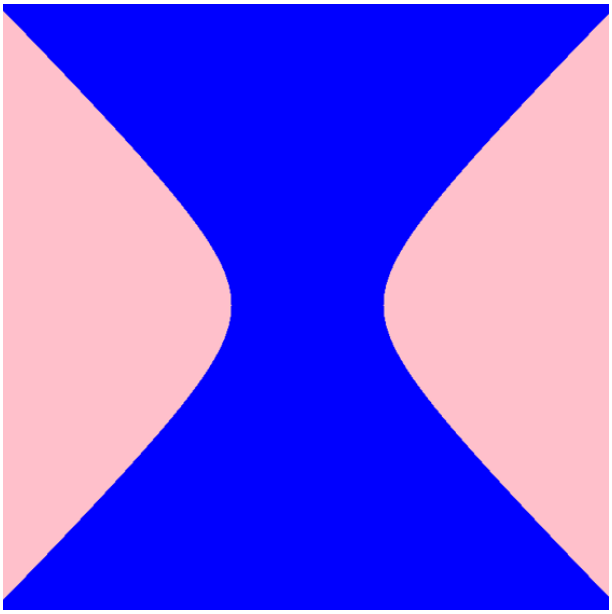
$$\frac{(x - x_0)^2}{a^2} - \frac{(y - y_0)^2}{b^2} = 1 \tag{3.3}$$

onde (x_0, y_0) é o ponto médio entre os dois focos em um eixo horizontal e $b^2 = c^2 - a^2$

Usaremos esse conceito para construir a imagem de uma hipérbole com interior preenchido pela cor azul que no sistema **RGB** fica $(0, 0, 255)$ e com o exterior com rosa $(255, 192, 203)$. A seguir é apresentado a implementação do código em python.

```
from PIL import Image

def hiperbole(size):
    rosa = (255,192,203)
    azul = (0, 0 , 255)
    img = Image.new("RGB",(size,size), rosa)

    x0 = size // 2
    y0 = size // 2
    a = size//8
    b = size//8

    for x in range(size):
        for y in range(size):
            if ((x-x0)**2)/(a)**2-((y-y0)**2)/(b)**2<1:
                img.putpixel((x,y), azul)
    return img


if __name__ == "__main__":
    hiperbole  = hiperbole(700)
    hiperbole.show()
```

Figura 11 – Hipérbole.



Fonte: O autor.

3.2 Propriedades de uma Imagem Digital

3.2.1 Vizinhança

Dado um pixel $p = (x, y)$ denotaremos de vizinhança 4 pixels, 2 horizontais e 2 verticais cujas as coordenadas são $(x + 1, y), (x - 1, y), (x, y + 1)$ e $(x, y - 1)$,ou seja, todos aqueles pixels com distância euclidiana igual a 1. Estes pixels formam a chamada "4-vizinhança"de p, que será designada $N_4(p)$.

Os quatro vizinhos diagonais de p são os pixels de coordenadas $(x - 1, y - 1), (x - 1, y + 1), (x + 1, y - 1)$ e $(x + 1, y + 1)$, ou seja, os pixels com distância $d_8 = 2$ e que não pertençam a 4-vizinhança, constituem o conjunto $N_d(p)$. A "8-vizinhança"de p é definida como:

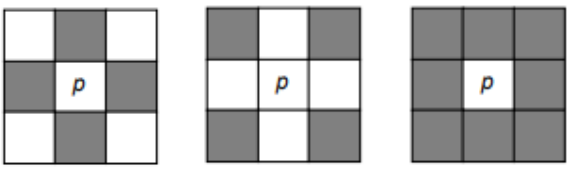
$$N_8(p) = N_4(p) \cup N_d(p) \tag{3.4}$$

As vizinhanças são ilustradas na figura 12.

3.2.2 Operações Aritméticas e Lógicas

Segundo (FILHO; NETO, 1999) após adquirir e digitalizar uma imagem passamos tratar ela como uma matriz, portanto é comum pensar em operações aritméticas

Figura 12 – Vizinhança do pixel P.



Fonte: (FILHO; NETO, 1999)

que podem também ser formuladas de acordo com operações que sejam efetuadas pixels a pixel ou orientadas a vizinhança de um pixel qualquer.

No primeiro caso, elas podem ser descritas pela seguinte notação:

$$X\, op\, Y = Z \tag{3.5}$$

Onde X e Y são matrizes e op é uma operação aritmética $(+, -, *, /)$ ou uma operação lógica (Conjunção "e", Disjunção Inclusiva "ou", Disjunção Exclusiva "ou...ou", Negação).

Vamos considerar duas imagens de dimensões iguais para efeito de estudo mas caso as imagens tiverem dimensões distintas poderemos por exemplo centraliza-las, reduzir as dimensões, maximizar e etc.

Inicialmente vamos usar uma imagem de uma circunferência e uma imagem de um quadrado, usado as medidas de distância euclidiana e a D_4. Uma observação importante que devemos fazer é que como nossa escala de cinza é fixada na faixa de [0,...,255] então precisamos converter os valores após as operações com a seguinte adequação:

$$g = \frac{255}{f_{max} - f_{min}}(f - f_{min}) \tag{3.6}$$

Outra forma de adequar essas valores é simplesmente truncando os valores acima de 255 pois é uma solução mais simples em relação ao tempo operacional.

Uma maneira de unir duas imagens é calculando a média aritmética de seus pixels, veremos abaixo o resultado das operações de multiplicação, divisão e conjunção com essas imagens matriciais:

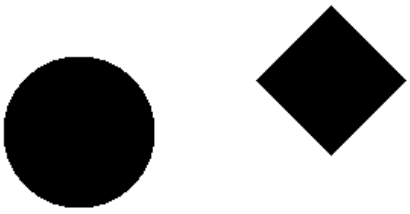
```
from PIL import Image

circulo = Image.new("L",(200,200),255)
losango = Image.new("L",(200,200),255)

media = Image.new("L",(200,200),255)
```

```
mutiplicacao = Image.new("L",(200,200),255)
divisao = Image.new("L",(200,200),255)

conjucao = Image.new("L",(200,200),255)
dijuncao = Image.new("L",(200,200),255)



x0 = 200//2
y0 = 200//2
x1 = 200//3
y1 = 200//3
raio = 200//4
for x in range(200):
    for y in range(200):
        if (x-x0)**2+(y-y0)**2<raio**2:
            circulo.putpixel((x,y),0)
        if abs(x-x1)+abs(y-y1)<raio:
            losango.putpixel((x, y), 0)

        pxlmedia = (circulo.getpixel((x, y))
                    +losango.getpixel((x, y)))//2
        pxlmul = min(255,circulo.getpixel((x,y))
                    *losango.getpixel((x,y)))
        pxldiv = int(circulo.getpixel((x,y))
                    /(losango.getpixel((x,y))+1))
        pxlcon = circulo.getpixel((x,y))
                    and losango.getpixel((x,y))
        pxldij = circulo.getpixel((x,y))
                    or losango.getpixel((x,y))


        media.putpixel((x, y), pxlmedia)
        mutiplicacao.putpixel((x,y),pxlmul)
        divisao.putpixel((x,y),pxldiv)

        conjucao.putpixel((x,y),pxlcon)
        dijuncao.putpixel((x,y),pxldij)
```

```
circulo.show()
losango.show()

media.show()
mutiplicacao.show()
divisao.show()
conjucao.show()
dijuncao.show()
media.show()
```

Na Figuras 13, 14 e 15 vemos a imagem do círculo e do losango criados através das distâncias e ação dos operadores de média, multiplicação, divisão, conjunção e disjunção, respectivamente.

Figura 13 – círculo e losango.



Fonte: O autor.

Figura 14 – Imagens unidas pela média, multiplicação e divisão.

Fonte: O autor.

3.2.3 União de Imagens

Dada duas imagens digitais uniremos elas pelo operador da média e pelo operador soma, como uma imagem RGB tem 3 canais então devemos calcular a média aritmética de cada um desses canais, uma observação importante que não devemos negligenciar é que somando os valores podemos ter canais acima de 255

Figura 15 – Imagem unidas pela conjunção e disjunção.

Fonte: O autor.

portanto trucaremos esses valores para 255, é importante também ressaltar que em geral imagens tem tamanhos distintos logo temos que fazer uma adequação dessas dimensões ou centralizar essas imagens no exemplo que mostraremos abaixo as imagens "Orinoco"e "pedra do machado"possuem as seguintes dimensões $(1079, 1020)$ e $(1079, 1078)$, respectivamente, portanto centralizaremos a menor de maneira simples fazendo ela começar no pixel $(\frac{1079-1079}{2}, \frac{1078-1020}{2}) = (0, 29)$, ou seja vamos sobrepor as imagens e o pixel (0,0) da imagem menor ficará sobre o pixel (0,16) da maior. Observe

```
from PIL import Image
from utils import show_horizontal1

orinoco = Image.open("input/orinoco.jpg")
pedra = Image.open("input/pedra do machado.jpg")
unida_media = Image.open("input/orinoco.jpg")
unida_soma = Image.open("input/orinoco.jpg")

largura_o,altura_o = orinoco.size
largura_p,altura_p= pedra.size
a= (altura_o-altura_p)//2
ImagemUnida = Image.new("RGB",(largura_o,altura_o))
for x in range(largura_o):
    for y in range(a,altura_o):
        pxl1 = orinoco.getpixel((x,y))
        pxl2 = pedra.getpixel((x,y-a))
        pxl=((pxl1[0]+pxl2[0])//2,
             (pxl1[1]+pxl2[1])//2,
             (pxl1[2]+pxl2[2])//2)
        pxl1 = (int(min(255,pxl1[0]+(pxl2[0]+1))),
                int(min(255,pxl1[1]+(pxl2[1]+1))),
```

```
                int(min(255,pxl1[2]+(pxl2[2]+1))))
        unida_media.putpixel((x,y),pxl)
        unida_soma.putpixel((x, y), pxll)




unida_media.show()
unida_soma.show()


orinoco.show()
pedra.show()
```

Muitas outras aplicações podemos obter através de operadores aritméticos e operadores lógicos na união de duas imagens. Na próxima seção abordaremos com mais formalidade os filtros de imagens digitais, usando o conceito de vizinhança e máscaras.

Figura 16 – Foto do rio Orinoco.

Figura 17 – Foto da Pedra do Machado.

Figura 18 – União de Imagens pelo operador Média

Figura 19 – União de Imagens pelo operador Soma

Fonte: O autor

4 FILTROS DE IMAGEM DIGITAL

4.1 Editando uma imagem pixel a pixel

A partir dessa seção trabalharemos com a edição de imagens onde faremos o algoritmo percorrer cada pixel e sob determinada condição a nova imagem é formada

4.1.1 Imagens de um canal

Arquivos PGM (Portable Gray Map) armazenam imagens 2D em escala de cinza. Cada pixel na imagem contém só um ou dois bytes de informação (8 ou 16 bits). Talvez não pareça muita coisa, mas os arquivos PGM podem armazenar dezenas de milhares de tons, de preto a branco, com todos os tons de cinza entre eles.

Na imagem abaixo os pixel são definidos com uma tonalidade diferente de cinza que vai de 0 a 255. Ou seja, cada pixel além de corresponder a uma localização de duas dimensões também tem um valor associado dos 256 tonalidades de cinza.

Figura 20 – Lena em formato PGM.

Fonte: Disponível em https://people.sc.fsu.edu/

4.1.1.1 Filtro Preto e Branco de um canal

O filtro preto e braco analisa cada tonalidade de cinza e relaciona com a cor preto ou branco, ou seja, das 256 tonalidades 128 delas receberá a cor **0** que representa o preto e outras 128 receberão a cor **255** que representa o branco.

O algoritmo do filtro consiste em "pintar"uma nova imagem,com as mesmas dimensões da imagem original, de branco e percorrer toda imagem pintando de preto todo pixel que tem valor menor que 128.

```
from PIL import Image


lena = Image.open("input/lena.pgm"))
largura, altura = lena.size

def nova_imagem(largura):
    BRANCO = 255
    PRETO = 0

    image = Image.new("L", (largura, altura), BRANCO)
    for x in range(largura):
        for y in range(altura):
            if lena.getpixel((x,y))<128:
                image.putpixel((x, y),PRETO)
    return image


if __name__ == "__main__":
    imagem=nova_imagem(largura)
    imagem.show()
```

Note que na segunda linha do código foi usado a fução *Image.open()* que abre um arquivo com o nome descrito nele, nesse caso a imagem estava com o nome de *lena.pgm* e estava dentro de uma pasta criada com o nome de *input*. Na terceira linha usamos o comando *.size* que extrai as dimensões da imagem. Além disso, criamos uma imagem no sistema *"L"* que significa luminância e assim podemos trabalhar somente com um valor inteiro para representar a tonalidade do pixel. O comando usado para extrair o valor do pixel é *.getpixel((x,y))*.

Assim obtemos a seguinte modificação da imagem:

Figura 21 – Lena aplicado filtro Preto e Branco.

Fonte: O autor.

4.1.1.2 Filtro Negativo com 1 canal

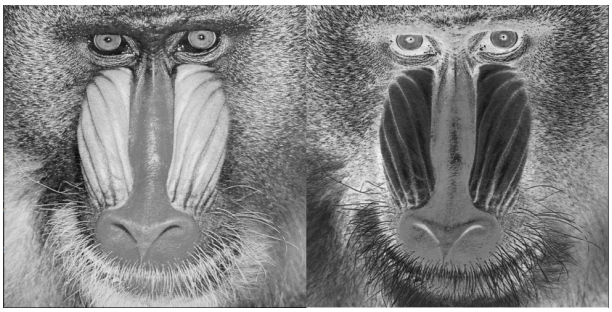
O filtro negativo consiste em inverter os valores de cada pixel segundo uma função bijetora onde onde a lei de formação da função é dada por $f(x) = 255 - x$, ou seja, a função inverterá os valores de x no intervalo de 0 a 255.

```
from PIL import Image

baboon = Image.open("input/baboon.pgm")
l, a = baboon.size

def nova_imagem(l):

    image = Image.new("L", (l, a))
    for x in range(l):
        for y in range(a):
            nova_cor = 255 - baboon.getpixel((x, y))
            image.putpixel((x, y),nova_cor)
    return image


if __name__ == "__main__":
    imagem=nova_imagem(l)
    imagem.show()
```

4.1.2 Filtro em escalas de cinza com 3 canais

A seguir serão apresentados alguns métodos usados para conversão de uma imagem em RGB para imagem em escala de cinza.

Figura 22 – Baboon aplicado o filtro Negativo.



Fonte: O autor.

4.1.2.1 Método da escolha de um canal

No sistema RGB as tonalidades de cinza são representadas por uma tupla de três valores iguais, ou seja, um pixel $f(x, y) = (k, k, k)$ onde $0 \leq k \leq 255$. Deste modo uma forma de transformar um pixel colorido em escala de cinza é escolhendo um dos canais para representá-lo.

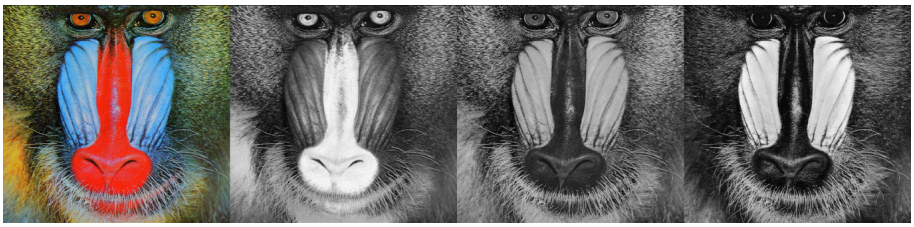
Assim, dado um pixel $f(x, y) = (25, 80, 3)$ sua representação por esse método seria $(25, 25, 25)$ ou $(80, 80, 80)$ ou $(3, 3, 3)$ a depender da escolha da melhor representação, logo abaixo temos o algoritmo que nos apresentará a imagem original e as três representações em escala de cinza.

```
from PIL import Image


original = Image.open("input/Baboon_color.png")
largura, altura = original.size

def escolha_de_canal(largura):
    img1 = Image.new("RGB",(largura,altura))
    img2 = Image.new("RGB", (largura,altura))
    img3 = Image.new("RGB", (largura,altura))
    for x in range(largura):
        for y in range(altura):
            cor = original.getpixel((x,y))
            img1.putpixel((x,y),(cor[0],cor[0],cor[0]))
            img2.putpixel((x, y), (cor[1],cor[1],cor[1]))
            img3.putpixel((x, y), (cor[2],cor[2],cor[2]))
    original.show()
    img1.show()
```

```
    img2.show()
    img3.show()

if __name__ == "__main__":
    cinza= escolha_de_canal(largura)
```

Na figura 23 observamos as diferenças da escolha de canais diferentes, da esquerda para direta temos a imagem original, a imagem produzida escolhendo o número do canal "RED", a imagem produzida escolhendo o número do canal "GREEN"e a imagem produzida escolhendo o número do canal "BLUE".

Figura 23 – Baboon aplicado ao filtro de escolha de canal.



Fonte: O autor.

4.1.2.2 Método do Máximo entre os 3 canais

Tendo uma imagem em RGB, cada pixel da matriz da imagem resultante será o maior dentre os pixels da mesma posição dos três canais do RGB. Por exemplo, dado um pixel na localização $a_{xy} = (152, 85, 162)$ o pixel resultante será $(162, 162, 162)$ e percorreremos toda imagem com esse processo.

Observe o algoritmo,

```
from PIL import Image

original = Image.open("input/Baboon_color.png")
largura, altura = original.size

def maximo(largura):
    imagem = Image.new("RGB",(largura,altura))
    for x in range(largura):
        for y in range(altura):
            cor = original.getpixel((x,y))
            m=max(cor[0],cor[1],cor[2])
```

```
                imagem.putpixel((x,y),(m,m,m))

    return imagem




if __name__ == "__main__":
    cinza= maximo(largura)
    cinza.show()
```

Figura 24 – Baboon, filtro de escala de cinza usando o método máximo dos canais.

Fonte: O autor.

4.1.2.3 Método da Média entre os 3 canais

Quando trabalhamos com vários valores é comum usamos como representante a média aritmética desses valores mesmo as vezes não sendo o melhor representante deste modo calcularemos a média aritmética entre os canais e como precisamos que essa média seja um número inteiro então tomaremos a divisão inteira por 3 da soma dos valores.

Observe que no algoritmo abaixo a única mudança é que agora calcularemos usando a seguinte operação $(cor[0] + cor[1] + cor[2])//3$. Porém, existe uma outra fórmula que é utilizada em processamento de imagens, que leva em consideração a capacidade de absorção do olho humano da luz emitida por cada componente de cor. É uma média ponderada das componentes RGB. Cinza = cor[0]*0,3 + cor[1]*0,59 + cor[2]*0,11 Onde cor[0] é a representação do "RED", cor[1] é a representação do "GREEN"e cor[2] é a representação do "BLUE".

```
from PIL import Image



original = Image.open("input/Baboon_color.png")
largura, altura = original.size

def cinza_media(largura):
    imagem = Image.new("RGB",(largura,altura))
    for x in range(largura):
        for y in range(altura):
            cor = original.getpixel((x,y))
            m= (cor[0]+cor[1]+cor[2])//3
            """use essa linha para média ponderada
            m = int(cor[0]*0.3+cor[1]*0.59+cor[2]*0.11) """

            imagem.putpixel((x,y),(m,m,m))

    return imagem


if __name__ == "__main__":
    cinza= cinza_media(largura)
```

Na figura 25 observamos a diferença entre a imagem colorida das frutas, o filtro da média aritmética e o filtro da média ponderada aplicados a ela. Onde conseguimos perceber diferenças sutis entre as imagens.

Figura 25 – Imagens de Frutas aplicado o filtro de escala de cinza em média aritmética e em média ponderada.

Fonte: O autor.

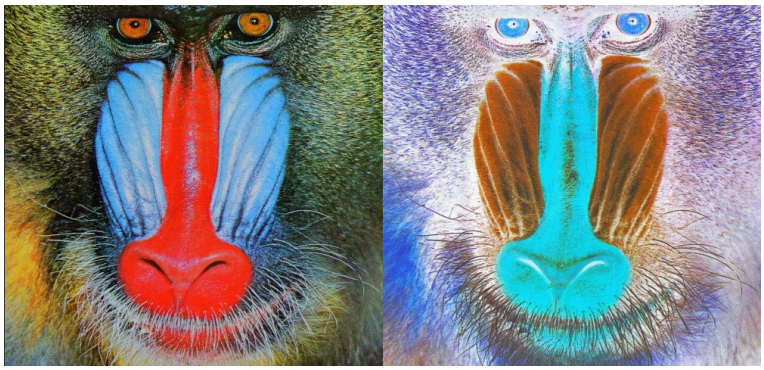
4.1.2.4 Filtro Negativo em 3 canais

Como vimos esse filtro para apenas um canal as cores de cada pixel devem revecebr um valor que seja oposto, deste modo é importante salientar que cada pixel resultante terá o valor desubtraído de 255, ou seja, dado um pixel $(x, y) = (r, g, b) \rightarrow f(x, y) = (255 - r, 255 - g, 255 - b)$ como é mostrado no algoritmo abaixo

```
from PIL import  Image

baboon = Image.open("input/Baboon_color.jpg")

largura,altura= baboon.size
for x in range(largura):
    for y in range(altura):
        r,g,b = baboon.getpixel((x,y))
        baboon.putpixel((x,y),(255-r,255-g,255-b))
baboon.show()
```

observe que nesse algoritmo não criamos outra imagem para receber os novos valores mas apenas modificamos os valores da matriz nela mesmo, pois essas operações são pontuais diferentemente de outros filtros que utilização a vizinhança do pixel para transformá-lo. Na figura 26 vemos como fica uma imagem negativa de 3 canais

Figura 26 – Representação da imagem negativa de uma imagem colorida.



Fonte: O autor.

4.2 Filtragem Espacial

Segundo (FILHO; NETO, 1999) o uso de máscaras espaciais no processamento de imagens é normalmente denominado filtragem espacial (em contraste com

a expressão “filtragem no domínio da frequência", utilizada quando se opera sobre a transformada de Fourier da imagem original) e as máscaras são conhecidas como filtros espaciais. Nesta seção trabalharemos alguns filtros espaciais de imagem como Correlação e Convolução. Filtros passa-baixa, passa-alta e passa-banda. Exemplos de filtros digitais de suavização (média, filtro de Gauss, mediana, suavização, conservativa, Kuwahara). Noção de gradiente. Filtros derivativos (Roberts, Prewitt e Sobel). Unsharp. Laplaciano. Laplaciano do gaussiano.

As técnicas de filtragem no domínio espacial são aquelas que atuam diretamente sobre a matriz de pixels que é a imagem digitalizada. Logo, as funções de processamento de imagens no domínio espacial podem ser expressas como:

$$g(x, y) = T[f(x, y)] \tag{4.1}$$

onde: g(x,y) é a imagem processada, f(x,y) é a imagem original e T é um operador em f, definido em uma certa vizinhança de (x,y).

Os filtros espaciais são operadores que permitem alterar a frequência espacial de uma imagem (sinal), modificando o valor do tom de cinza de cada pixel em função dos valores dos tons de cinza dos pixels da sua vizinhança.

Os filtros podem ser lineares ou não-lineares. Nos filtros lineares cada pixel resulta de uma combinação linear entre os pixels da sua vizinhança, com coeficientes que correspondem aos pesos a atribuir às parcelas. Quaisquer outros filtros são designados por filtros não lineares.

4.2.1 Filtros Digital em duas dimensões

Por conveniência definiremos um filtro de imagem digital M de duas dimensões por uma matriz que iremos chamar de ***Máscara*** ou ***Kernel***. Essa Máscara respeitará as seguintes condições:

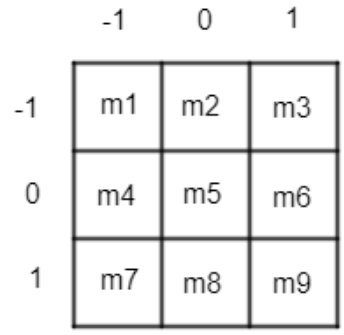
m1) M é uma matriz quadrada.

m2) Se M tem um número ímpar de elementos, ou seja, tem (2N+1)×(2N+1) elementos então estes estão indexados desde -N até N, tal que o elemento central de H é H(0,0).

m3) Se M tem um número par de elementos, ou seja, tem (2N)×(2N) elementos então estes estão indexados desde 0 até 2N, tal que o elemento central de H é H(0,0).

m4) Os valores de M designam-se por “coeficientes”.

Observe o modelo de filtro abaixo:



4.3 Correlação e Convolução

Os conceitos apresentados nesse tópico serão sempre tratados de maneira discreta por conta da natureza matricial das operações e transformações feitas com as imagens.

4.3.1 Correlação

Dado uma imagem Matricial $f(x, y)$ e uma máscara $H(x, y)$, defini-se a operação de correlação $(\otimes)$ entre H e f por:

$$H \otimes f(x, y) = \sum_{i=-N}^{N} \sum_{j=-N}^{N} H(i, j) \times f(x + i, y + j) \tag{4.2}$$

4.3.2 Convolução

A operação de Convolução $(*)$ é semelhante ao de Correlação pois aplicar a convolução é o mesmo que rotacionar a máscara em $180°$ e aplicar a Correlação. Contudo podemos definir a Convolução de uma máscara H com uma imagem f por

$$H * f(x, y) = \sum_{i=-N}^{N} \sum_{j=-N}^{N} H(i, j) \times f(x - i, y - j) \tag{4.3}$$

Observamos que se a máscara H for uma matriz simétrica a correlação e convolução serão operações idênticas.

Além disso, diferenciamos a correlação e a convolução, em que a última respeita a propriedade associativa. Ou seja, se G e H são filtros e I é uma imagem, então, $G * (H * I) = (G * H) * I$

A importância desta propriedade torna-se bastante conveniente quando pretende-se aplicar mais do que um filtro a uma imagem. Ou seja, aplica-se a convolução entre os filtros e após a convolução do filtro resultante com a imagem original.

No contexto do processamento de imagem, uma das matrizes de entrada é geralmente uma imagem de níveis de cinzento (I). A segunda matriz, geralmente bem menor e igualmente bidimensional (apesar de poder também ser apenas um vector), corresponde ao filtro H, dentro do qual se estabelece uma posição de referência como sendo o seu pixel central (como já antes referido)

Como trabalhamos com operações que envolvam a vizinhança do pixel temos que tomar alguns cuidados quantos aos pixels da borda da imagem para isso temos algumas soluções para um filtro de ordem $2N + 1 \times 2N + 1$ e uma imagem I de ordem $M \times M$, onde $M > 2N + 1$:

- Usa-se apenas a convolução em um subconjunto de pixel da imagem, ou seja, o filtro H irá percorrer os pixels da imagem I somente se a distância euclidiana dos pixel da imagem for maior ou igual a N.
- A imagem matricial é ampliada adicionando colunas e linhas com valor 0 nas bordas, essa solução pode distorcer a intensidade dos pixels da borda da imagem.
- A imagem matricial é ampliada adicionando colunas e linhas com valor igual aos pixels nas bordas, essa solução pode distorcer a intensidade dos pixels da borda da imagem.

4.4 Filtros de Passa - Baixa

Filtro Passa-Baixa é um filtro espacial linear que atenua as frequências espaciais mais altas de uma imagem e acentua as mais baixas. Normalmente é utilizado para remover pequenos detalhes indesejados, eliminar algum tipo de ruído ou suavizar imagens. Ou seja, o objetivo desse filtro é:

- Suavizar a imagem pela redução das variações nos de níveis de cinza que dão à aparência de “serrilhado” nos patamares de intensidade.
- Atenuar as altas frequências, que correspondem às transições abruptas.
- Minimizar ruídos.

Ruído é uma palavra usada principalmente na linguagem das para designar erros. Em imagens, é uma variação dos valores nos níveis de cinza dos pixels na imagem, causados por erros na transmissão de dados, ou eventuais distorção introduzida na

fase de aquisição de uma imagem em geral. Os pontos ruidosos aparecem distribuídos aleatoriamente e é geralmente expresso como uma distribuição de probabilidade com uma média específica e um desvio padrão.

Dois tipos de ruídos aparecem com mais frequência. O ruído Gaussiano (aleatório) que é um ruído aditivo, distribuído sobre a imagem, com média 0 (zero) e desvio padrão m e o ruído sal e pimenta (impulsivo) com densidade s onde aparecem pixels com valores alternadamente modificados para 0 (pimenta) e para o valor máximo da imagem (sal).

Para retirar esses ruídos são utilizados os filtros de suavização.

4.4.1 Filtro de Média

Esse filtro é uma máscara de convolução $n \times n$ com todos seus coeficientes iguais a 1 e depois dividindo-se o valor obtido pelo número de pixels da máscara (n^2)

Abaixo temos os filtros 3×3, 5×5 e 7×7

$$H_3 = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad H_5 = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad H_7 = \frac{1}{49}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Para implementar nosso algoritmo desse filtro temos que usar o conceito de bola de um ponto onde o raio será a distância D_8 de um pixel, ou seja, para implementação desse algoritmo faremos dois laços de repetição que percorrerão toda vizinhança de um pixel dado um raio r. Assim a máscara de raio r terá dimensão de $2r + 1 \times 2r + 1$, ou seja, raio 1 é uma máscara 3×3 por exemplo. Além disso temos que tratar as bordas que nesse caso alimentará a dimensão da matriz em cada um dos lados r linhas ou r colunas.

O algoritmo abaixo de implementação no python é feito para imagens PGM, ou seja, um canal porém podemos fazer de maneira análoga para imagens de 3 canais basta repetir o processo para cada um dos canais.

```
from PIL import Image
```

```
lena = Image.open("input/lena_sp.pgm")
l, a = lena.size
raio = int(input("Raio: "))
imagem_borda = Image.new("L", (l+2*raio,a+2*raio),0)
imagem_filtrada = Image.new("L", (l,a))

'''Abaixo aumentamos as bordas dependo do raio requerido'''

for x in range(l):
    for y in range(a):
        cor = lena.getpixel((x,y))
        imagem_borda.putpixel((x+raio,y+raio),cor)


for x in range(raio,l+raio):
    for y in range(raio,a+raio):
        m=255
        for i in range(x-raio,x+raio+1):
            for j in range(y-raio,y+raio+1):
                m=m+imagem_borda.getpixel((i,j))
        m=m//((2*raio+1)**2)
        imagem_filtrada.putpixel((x-raio,y-raio),m)

lena.show()
imagem_filtrada.show()
```

No algoritmo usamos uma variável m para receber o somatório dos valores da convolução da máscara com o setor da imagem, como os valores da máscara são todos iguais a 1 logo ficou o somatório dos pixels vizinhos a uma distancia menor ou igual ao raio.

É importante ressaltar que nessa implementação foi usada uma imagem com ruído de sal e pimenta para testarmos a funcionalidade do filtro, nas imagens seguintes observamos o resultado da aplicação para raio 1, 2, 3, 4 e 5.

Observamos que quanto maior o raio do filtro mais borrada fica a imagem.

O filtro da média limita-se quando devemos remover ruídos em imagens e preservar bordas e detalhes finos. Para contorná-la, uma técnica alternativa é o filtro da mediana.

Figura 27 – Lena com ruído sal e pimenta.

Fonte: Disponível em https://people.sc.fsu.edu/

Figura 28 – Filtro média aplicado com raio igual a 1.

Fonte: O autor.

4.4.2 Filtro da Mediana

Esse filtro é não linear e sua técnica consiste em substituir o nível de cinza do pixel central da janela pela mediana dos pixels situados em sua vizinhança a um raio r de distância.

A mediana m de um conjunto de n elementos é o valor tal que metade dos n

Figura 29 – Filtro média aplicado com raio igual a 2.

Fonte: O autor.

Figura 30 – Filtro média aplicado com raio igual a 3.

Fonte: O autor.

elementos do conjunto situam-se abaixo de m e a outra metade acima de m. Quando n é ímpar, a mediana é o próprio elemento central do conjunto ordenado. Nos casos em que n é par, a mediana é calculada pela média aritmética dos dois elementos mais próximos do centro.

Como o cálculo da mediana requer uma ordenação dos valores analisados para após calcular o valor da mediana optamos por usar um módulo chamado *median* de

Figura 31 – Filtro média aplicado com raio igual a 4.

Fonte: O autor.

Figura 32 – Filtro média aplicado com raio igual a 5.

Fonte: O autor.

uma biblioteca chamada *statistics* pois fazer essa ordenação implicaria uma grande demanda computacional que não é o foco desse trabalho mas o leitor pode consulta vários algoritmos de ordenação na obra de (CORMEN et al., 2012).

O algoritmo abaixo é semelhante ao algoritmo do filtro da média mas para calcular a mediana guardamos os valores do somatório da convolução em uma matriz vetor e chamamos o método *median* para extrair esse valor de mediana no vetor.

```
from PIL import Image
import statistics

lena = Image.open("input/lena_sp.pgm")
l, a = lena.size
raio = int(input("Raio: "))
imagem_borda = Image.new("L", (l+2*raio,a+2*raio),0)
imagem_filtrada = Image.new("L", (l,a))

'''Abaixo aumentamos as bordas dependo do raio requerido'''

for x in range(l):
    for y in range(a):
        cor = lena.getpixel((x,y))
        imagem_borda.putpixel((x+raio,y+raio),cor)


for x in range(raio,l+raio):
    for y in range(raio,a+raio):
        m=[]
        for i in range(x-raio,x+raio+1):
            for j in range(y-raio,y+raio+1):
                m.append(imagem_borda.getpixel((i,j)))
        """o valor da mediana tem que ser inteiro"""
        mediana=int(statistics.median(m))

        imagem_filtrada.putpixel((x-raio,y-raio),mediana)

lena.show()
imagem_filtrada.show()
```

Deste mesmo modo podemos definir **Filtro de ordem**, as intensidades dos pontos da vizinhança do pixel f(x,y), dentro de uma janela da imagem, são ordenadas e é tomado o valor máximo ou o valor de uma ordem qualquer desta ordenação, como novo valor para g(x,y), ou seja, da imagem filtrada.

Figura 33 – Lena com ruído sal e pimenta.

Fonte: Disponível em https://people.sc.fsu.edu/

Figura 34 – Filtro mediana aplicado com raio igual a 1.

Fonte: O autor.

4.4.3 Filtro da Moda

Dado uma imagem $f(x, y)$ as intensidades dos pontos da vizinhança do pixel (x, y), dentro de uma janela da imagem, são ordenadas e é tomado como novo valor para $f(x, y)$, o valor $g(x, y)$ mais frequente da vizinhança, onde $g(x, y)$ é o vetor que guardou os valores da vizinhança de (x, y).

Figura 35 – Filtro mediana aplicado com raio igual a 2.

Fonte: O autor.

Figura 36 – Filtro mediana aplicado com raio igual a 3.

Fonte: O autor.

4.4.4 Filtro Gaussiano

Os filtros apresentados até aqui são todos de suavização da imagem, ou seja, sua principal utilidade é para eliminar valores muito dispersos de uma amostra. Pensando nesse objetivo iremos usar a função distribuição gaussiana de duas dimensões apresentada por (MORETTIN; BUSSAB, 2010) para criar os coeficientes de uma máscara. Os valores para o filtro gaussiano em uma dimensão são dados pela função:

Figura 37 – Filtro mediana aplicado com raio igual a 4.

Fonte: O autor.

Figura 38 – Filtro mediana aplicado com raio igual a 5.

Fonte: O autor.

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4.4}$$

e os valores para o filtro gaussiano para duas dimensão são dados pela função:

$$G(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} exp\{-\frac{1}{2(1-\rho^2)}[(\frac{x-\mu_x}{\sigma_x})^2 - 2\rho\frac{(x-\mu_x)(y-\mu_y)^2}{\sigma_x\sigma_y} + (\frac{y-\mu_y}{\sigma_y})^2]\} \tag{4.5}$$

Onde $exp\{w\} = e^w$, μ a média, ρ é o coficiente de correlação entre dois conjunos

de dados e σ é o desvio padrão dos dados, como nossa os indices (x, y) dos elementos da máscara variam entre $-N \leq x \leq N$ e $-N \leq y \leq N$ temos que $\rho = \mu_x = \mu_y = 0$ e $\sigma_x = \sigma_y$ assim nossa função gaussiana para duas dimensões fica

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4.6}$$

Para $\sigma = 1$ temos os coeficientes do kernel igual a

$$\frac{1}{273}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 7 & 4 & 1 \\ \hline 4 & 16 & 26 & 16 & 4 \\ \hline 7 & 26 & 41 & 26 & 7 \\ \hline 4 & 16 & 26 & 16 & 4 \\ \hline 1 & 4 & 7 & 4 & 1 \\ \hline \end{array}$$

O algoritmo de implementação dessa mascára é:

```
from PIL import Image

def gaussiano(foto):
    largura, altura = foto.size
    img_borda = Image.new("L",(largura+4,altura+4),128)
    img_filtrada = Image.new("L",(largura,altura))
    mask = [[1,4,7,4,1],
            [4,16,26,16,4],
            [7,26,41,26,7],
            [4, 16, 26, 16, 4],
            [1, 4, 7, 4, 1]
    ]


    for x in range(largura):
        for y in range(altura):
            cor = imagem.getpixel((x,y))
            img_borda.putpixel((x + 2, y + 2), cor)
    for x in range(2,largura+2):
        for y in range(2,altura+2):
            pxl=0
            for i in range(-2,3):
                for j in range(-2, 3):
                    cor = img_borda.getpixel((x+i,y+j))
                    pxl = pxl + cor*mask[i+2][j+2]
```

```
                pxl=pxl//273
                img_filtrada.putpixel((x-2,y-2),pxl)
    return img_filtrada




if __name__ == "__main__":
    imagem = Image.open("input/lena.pgm")
    foto = gaussiano(imagem)
    foto.show()
```

Para aplicar o filtro para uma imagem de 3 canais basta repetir o processo para cada um dos canais e tratar a imagem com coloração RGB, abaixo podemos vê as modificações necessárias para a aplicação do filtro gaussiano:

```
from PIL import Image

def gaussiano(foto):
    largura, altura = foto.size
    img_borda = Image.new("RGB",(largura+4,altura+4),128)
    img_filtrada = Image.new("RGB",(largura,altura))
    mask = [[1,4,7,4,1],
            [4,16,26,16,4],
            [7,26,41,26,7],
            [4, 16, 26, 16, 4],
            [1, 4, 7, 4, 1]
    ]

    for x in range(largura):
        for y in range(altura):
            cor = imagem.getpixel((x,y))
            img_borda.putpixel((x + 2, y + 2), cor)
    for x in range(2,largura+2):
        for y in range(2,altura+2):
            pxl1=0
            pxl2=0
            pxl3=0
            for i in range(-2,3):
```

```
                for j in range(-2, 3):
                    cor = img_borda.getpixel((x+i,y+j))
                    pxl1 = pxl1 + cor[0]*mask[i+2][j+2]
                    pxl2 = pxl2 + cor[1] * mask[i + 2][j + 2]
                    pxl3 = pxl3 + cor[2] * mask[i + 2][j + 2]

            pxl1=pxl1//273
            pxl2 = pxl2 // 273
            pxl3 = pxl3 // 273
            img_filtrada.putpixel((x-2,y-2),(pxl1,pxl2,pxl3))
    return img_filtrada




if __name__ == "__main__":
    imagem = Image.open("input/Baboon_color.jpg")
    foto = gaussiano(imagem)
    foto.show()
```

Figura 39 – Lena aplicado o filtro gaussiano 5 por 5.

Fonte: O autor.

Figura 40 – Lena aplicado o filtro gaussiano 5 por 5.

Fonte: O autor.

4.5 Filtros de Passa-Alta

O principal objetivo das técnicas de filtragem Passa-Alta é evidenciar as altas frequências das imagens, ou seja, os traços finos e mudanças abruptas nos níveis de cinza da imagem. A máscara do filtro de passa-alta dever ter pesos de tal forma que a soma dos coeficientes seja zero.

4.5.1 Filtro de Passa-Alta Básico

Segundo (FILHO; NETO, 1999) os coeficientes de uma máscara de um filtro de passa-alta deverá ter os coeficientes nas proximidades de seu centro positivos e os restantes negativos respeitando sempre que a somatória dos coeficientes seja zero. Em uma máscara 3×3 por exemplo podemos usar os coeficientes abaixo:

$\frac{1}{9}$

-1	-1	-1
-1	8	-1
-1	-1	-1

ou $\frac{1}{5}$

0	-1	0
-1	4	-1
0	-1	0

Deste modo significa que quando a máscara for aplicada a regiões homogêneas de uma imagem, o resultado será zero ou um valor muito baixo, o que é consistente com o princípio da filtragem que é capitar somente as altas frequências da imagem. Outro fator a se considerar é que em filtros de passa-altas podem ou não ter as frações normalizadoras por conta de da soma dos seus coeficientes ser zero. Para a implementação desse filtro é de extrema importância mostrar ele aplicado com o operador soma com a imagem original, então no algoritmo que apresentaremos abaixo veremos 3 imagens, a original, o resultado do filtro de passa-alta e a soma dessas duas primeiras. Desta vez apresentaremos o algoritmo para uma imagem com os 3 canais RGB mas mostraremos também o resultado para uma imagem em escalas de cinza.

```
from PIL import Image

def passaalta(foto):
    largura, altura = foto.size
    img_borda = Image.new("RGB",(largura+2,altura+2),128)
    img_filtrada = Image.new("RGB",(largura,altura))
    img_pa = Image.new("RGB", (largura, altura))

    # MÁSCARA DE PASSA-ALTA
    mask = [[-1,-1,-1],
            [-1,8,-1],
```

```
            [-1,-1,-1]
    ]
    for x in range(largura):
        for y in range(altura):
            cor = imagem.getpixel((x,y))
            img_borda.putpixel((x + 1, y + 1), cor)
    for x in range(1, largura + 1):
        for y in range(1, altura + 1):
            pxl1 = 0
            pxl2 = 0
            pxl3 = 0
            for i in range(-1, 2):
                for j in range(-1, 2):
                    cor = img_borda.getpixel((x + i, y + j))
                    pxl1 = pxl1 + cor[0] * mask[i + 1][j + 1]
                    pxl2 = pxl2 + cor[1] * mask[i + 1][j + 1]
                    pxl3 = pxl3 + cor[2] * mask[i + 1][j + 1]
            img_filtrada.putpixel((x - 1, y - 1),
                                  (pxl1, pxl2, pxl3))
    img_filtrada.show()

    #APLICAÇÃO DO OPERADOR SOMA
    for x in range(largura):
        for y in range(altura):
            pxl_f = img_filtrada.getpixel((x,y))
            pxl_0= imagem.getpixel((x,y))
            pxl=(pxl_0[0]+pxl_f[0],
                pxl_0[1]+pxl_f[1],
                pxl_0[2]+pxl_f[2])
            img_pa.putpixel((x,y),pxl)
    img_pa.show()

if __name__ == "__main__":
    imagem = Image.open("input/PeppersRGB.jpg")
    foto = passaalta(imagem)
```

Figura 41 – imagem original, filtro de passa-alta básico e imagem com o operador soma das imagens anteriores

Fonte: O autor.

Figura 42 – imagem original, filtro de passa-alta básico e imagem com o operador soma das imagens anteriores

Fonte: O autor.

4.5.2 Realce de imagem por derivação

Segundo (GONZALES; WOODS, 2010) o cálculo da média dos pixels em um trecho de imagem produz como efeito a suavização de altas frequências e que o conceito de média é análogo à operação de integração, é razoável esperar que a diferenciação produza o efeito oposto e assim as alta frequência presentes em uma imagem fiquem em evidência . O método mais usual de diferenciação em aplicações de processamento de imagens é o gradiente. Em termos contínuos, o gradiente de $f(x, y)$ em um certo ponto (x, y) foi definido como vetor em (2.12) como

$$\nabla f(x_0, y_0) = \left(\frac{\partial}{\partial x} f(x_0, y_0), \frac{\partial}{\partial y} f(x_0, y_0) \right) \tag{4.7}$$

O comprimento desse vetor é dado por:

$$mag(\nabla f) = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2} \tag{4.8}$$

$$\nabla f(x,y) \approx |f(x,y) - f(x+1,y)| + |f(x,y) - f(x,y+1)| \tag{4.9}$$

ou

$$\nabla f(x,y) \approx |f(x,y) - f(x+1,y+1)| + |f(x+1,y) - f(x,y+1)| \tag{4.10}$$

Essas duas aproximações calculadas em (4.9) e (4.10) geram algumas máscaras como:

4.5.2.1 Roberts

Este filtro executa a equação (4.10), isto é, em vez de calcular as diferenças de valores de brilho na direcção vertical e horizontal, fá-lo numa direcção rodada de 45^o, onde as janelas de convolução são as seguintes:

1	0
0	-1

e

0	1
-1	0

4.5.2.2 Sobel

Este filtro realça linhas verticais e horizontais mais escuras que o fundo, sem realçar pontos isolados. Consiste na aplicação de duas máscaras, descritas a seguir, que compõem um resultado único por outro lado cada uma das máscaras evidenciará um tipo de linha diferente da imagem, as máscaras são:

Eixo X

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

e Eixo Y

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

4.5.2.3 Prewitt

Assim como Sobel este operador realça linhas verticais e horizontais mais escuras que o fundo, sem realçar pontos isolados, a diferença é sutil nos coeficientes e no resultado.

Eixo X

-1	0	1
-1	0	1
-1	0	1

e Eixo Y

-1	-1	-1
0	0	0
1	1	1

Para a implementação em python é necessário fazer uma pequena edição no algoritmo de passa-alta básico em relação aos coeficiente mas não é necessário usar o operador soma.

4.6 Filtragem Referente as Frequências

Segundo (MENESES et al., 2012) em qualquer imagem digital observamos uma forte interdependência da vizinhança do pixel e dos valores dos pixels, porque os alvos da captura da imagem tendem a mostrar uma homogeneidade dentro de certos espaços.

Essa interdependência pode ser utilizada para realçar de forma seletiva detalhes geométricos da imagem, como as bruscas mudanças ou bordas entre áreas aparentemente homogêneas, que na realidade se configuram como limites.

É observado, em geral, nas imagens a presença de características que marcam, por assim dizer, o rompimento desta interdependência, e que não parecem estar associadas a imagem real do objeto, e que neste caso são identificadas como ruídos.

Nas seções anteriores trabalhamos filtros que estão relacionados diretamente com as vizinhanças de um pixel e trabalhamos também com filtros que captam as baixas e altas frequências. Nessa seção trabalharemos diretamente com as frequências de brilho (níveis de cinza) de uma imagem através da Transformada de Fourier (FT).

Como uma imagem digital é função bidimensional discreta com valor $f(x, y)$ associados a cada pixel, a Transformada Discreta de Fourier (FDT) e sua inversa apresentadas nas equações (2.28) e (2.29) é uma opção para filtrar imagens no domínio das frequências.

4.6.1 Transformada de Sinais Espaciais

Para ilustrar o processo que a FDT provoca em uma função bidimensional trataremos a imagem de um retângulo como uma função do tipo

$$f(x,y) = \begin{cases} 1,\ se\,(x,y) \in [-a,a] \times [-b,b] \\ 0, \textit{caso contrário} \end{cases} \tag{4.11}$$

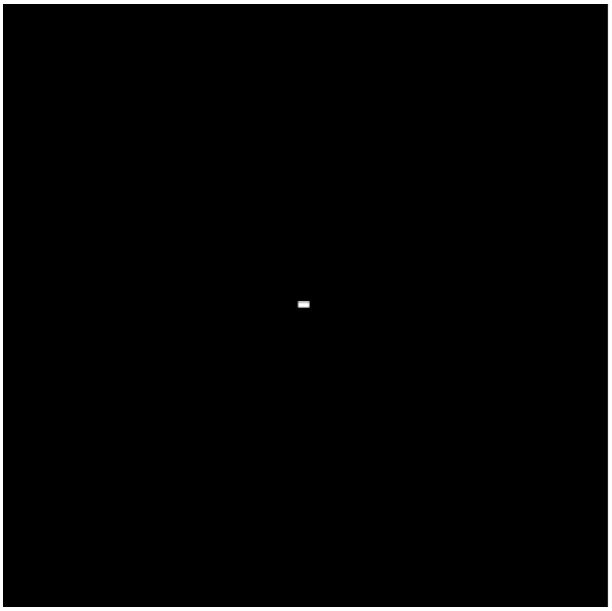
Para a implementação no código em python não usaremos mais o Pillow pois para trabalhar com a FT é melhor usar bibliotecas que tratam as imagens como matrizes propriamente no sentido matemático pra fazer as operações pertinentes.

Abaixo segue um algoritmo que cria a imagem de dimensões $2^{10} \times 2^{10}$ com um retângulo de 20 pixels $\times$ 10 pixels de dimensão, o fato do retângulo ser pequeno foi proposital pois com menores picos de frequência a FT cria impulsos mais oscilantes logo visivelmente fica de melhor compreensão.

```
import numpy as np
import scipy.fftpack as fp
import matplotlib.pyplot as plt

N = 2**10
#CRIA UMA MATRIZ ONDE TODOS OS TERMOS SÃO 0
f=np.zeros((N, N))
l = 10
a = 5

#DETERMINA QUE TODOS OS TERMOS NESSAS DIMESÕES SEJAM 1
f[N//2 - a: N//2 + a, N//2 - l: N//2 + l] = 1

#ESTILO DE COLORAÇÃO
colormap = 'gray'

#ELEMENTO DA IMAGEM QUE APARECERÁ EM FORMA DE GRÁFICO
plt.figure()
plt.title('Imagem original')
plt.imshow(f, cmap=colormap)
plt.colorbar()

#CALCULANDO A DFT EM 2D
F= fp.fft2(f)
Fm = np.absolute(F)
Fm /= Fm.max() #NORMALIZA EM [0,1]

#APRESENTANDO A |FT|
plt.figure()
plt.title('Imagem da FT')
plt.imshow(Fm, cmap=colormap, vmax=0.2)
plt.colorbar()

#CENTRALIZANDO O IMPULSO DA |FT|
Fm = fp.fftshift(Fm)
plt.figure()
plt.title('FT COM SHIFT')
plt.imshow(Fm, cmap=colormap, vmax=0.2)
plt.colorbar()
```

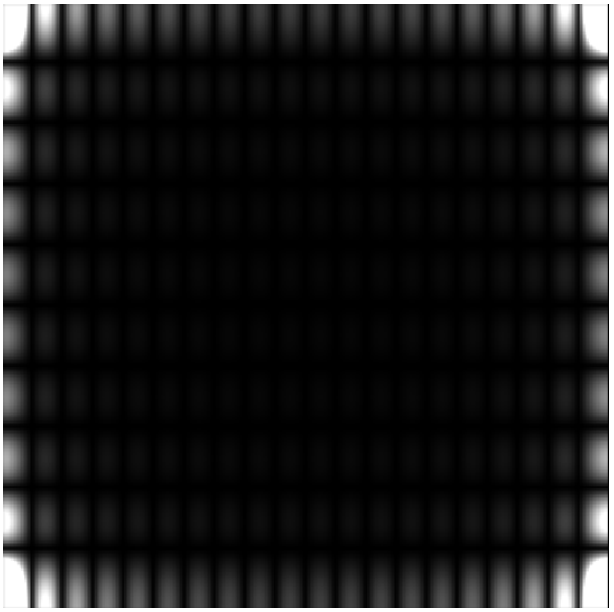
```
plt.show()
```

Figura 43 – Imagem original de 1024 pixel de dimensão com um pequeno retângulo no centro.



Fonte: O autor.

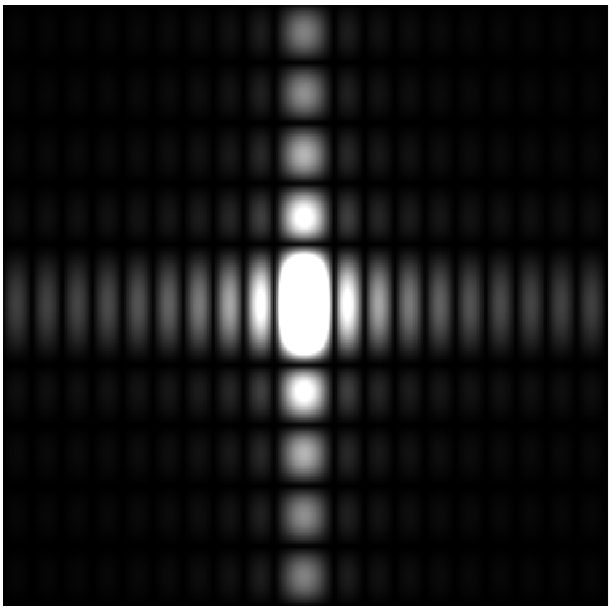
Figura 44 – Impulso gerado pela DFT na imagem 43



Fonte: O autor.

De maneira geral alguns comprimentos de ondas das frequências das imagens não são vistas na representação dos impulsos da FT logo é sempre indicado mostrar a

Figura 45 – Impulso centralizado na imagem 44



Fonte: O autor.

FT na escala logarítmica para efeito didático, porém para os cálculos trabalharemos apenas com o valor absoluto pois tanto a FT como a IFT de funções bidimensionais geram valores complexos.

A seguir apresentaremos 4 filtros usando a FT usados no processamento de imagens de sensoriamento remoto apresentados no livro (MENESES et al., 2012).

4.6.2 Filtro Passa Alta Circular

Corta as frequências externas ao circulo central das frequências e deixa passar as frequências internas ao circulo (região clara). Pode-se concluir que a região clara da Transformada de Fourier está associada às altas frequências e a região cinza às baixas frequências. É necessário que dependendo da imagem defina alguns parâmetros, tais como o raio em pixel do círculo. Utiliza-se esse tipo de filtro para obter as feições de detalhe na imagem.

Logo abaixo obtermos um algoritmo que apresenta 4 imagens para efeitos didáticos, a saber: a imagem original, o impulso da FT, o corte feito no centro do impulso e a imagem filtrada.

```
 from PIL import Image
import numpy as np
import scipy.fftpack as fp
import matplotlib.pyplot as plt
```

```
f=Image.open("input/lena.pgm")



#ESTILO DE COLORAÇÃO
cmap='gray'
#ELEMENTO DA IMAGEM QUE APARECERÁ EM FORMA DE GRÁFICO
plt.figure()
plt.title('Imagem original')
plt.imshow(f, cmap=cmap)
plt.colorbar()

#CALCULANDO A DFT EM 2D
F= fp.fft2(f)

#CALCULO PARA VISUALIZAÇÃO DA FT
Fm = np.absolute(F)
Fm /= Fm.max() #NORMALIZA EM [0,1]
Fm = fp.fftshift(Fm)
Fm = np.log(Fm)

#VIZUALIZAÇÃO DA FT
plt.figure()
plt.title('Imagem da FT em escala Log')
plt.imshow(Fm, cmap=cmap)
plt.colorbar()

#FILTRANDO O CIRCULO CENTRAL QUE CONTEM AS ALATS FREQUENCIAS
x0=256
y0=256
FT=F
FT=fp.fftshift(FT)
for x in range(512):
    for y in range(512):
        if (x-x0)**2+(y-y0)**2>1000:
            FT[x][y]=0

FTT = np.absolute(FT)
FTT /= FTT.max() #NORMALIZA EM [0,1]
FTT = np.log(FTT)
```

```
#APRESENTANDO A |FT| FILTRADA
plt.figure()
plt.title('Imagem da FT filtrada')
plt.imshow(FTT, cmap=cmap)
plt.colorbar()

IFT=fp.ifft2(FT)
IFT=np.absolute(IFT)

plt.figure()
plt.title('Imagem resultante')
plt.imshow(IFT, cmap=cmap)
plt.colorbar()

plt.show()
```

Figura 46 – Espectro de Fourier

Imagem original **Espectro de Fourier**

Fonte: O autor.

4.6.3 Filtro Passa Baixa Circular

Corta as frequências internas ao círculo (região clara) e deixa passar as frequências externas ao circulo (região cinza). Nesse caso, utiliza-se esse filtro para obter as feições de regionais na imagem. É muito semelhante ao filtro anterior, inclusive as

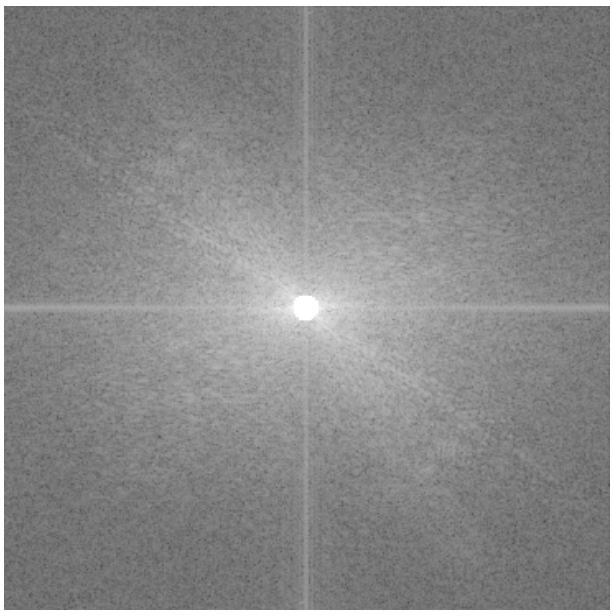
Figura 47 – Filtro Passa-Alta Circular

Espectro filtrado **Imagem filtrada**

Fonte: O autor.

únicas modificações no algoritmo é trocar o sinal de maior o sinal de menor e diminuir o raio desse modo ficamos com as imagens.

Figura 48 – Filtro Passa-Baixa Circular: Processo de filtragem no domínio da frequência



Espectro filtrado: passa-baixa circular **Imagem filtrada**

Fonte: O autor.

4.6.4 Filtro Banda Passante Circular Interno

O filtro passa banda passante circular interno cria dois círculos e corta as frequências internas ao círculo menor e externas ao círculo maior e deixa passar as frequências internas ao circulo maior e externa ao menor.

Pode-se concluir que o filtro banda passante deixa passar as frequências na região entre os círculos e corta as demais frequências. Utiliza-se esse filtro para obter as feições de detalhes limitados a determinadas frequências que se queira capitar.

```
 from PIL import Image
import numpy as np
import scipy.fftpack as fp
import matplotlib.pyplot as plt

f=Image.open("input/lena.pgm")


#ESTILO DE COLORAÇÃO
cmap='gray'
#ELEMENTO DA IMAGEM QUE APARECERÁ EM FORMA DE GRÁFICO
plt.figure()
plt.title('Imagem original')
plt.imshow(f, cmap=cmap)
plt.colorbar()

#CALCULANDO A DFT EM 2D
F= fp.fft2(f)

#CALCULO PARA VISUALIZAÇÃO DA FT
Fm = np.absolute(F)
Fm /= Fm.max() #NORMALIZA EM [0,1]
Fm = fp.fftshift(Fm)
Fm = np.log(Fm)

#VIZUALIZAÇÃO DA FT
plt.figure()
plt.title('Imagem da FT em escala Log')
plt.imshow(Fm, cmap=cmap)
plt.colorbar()
```

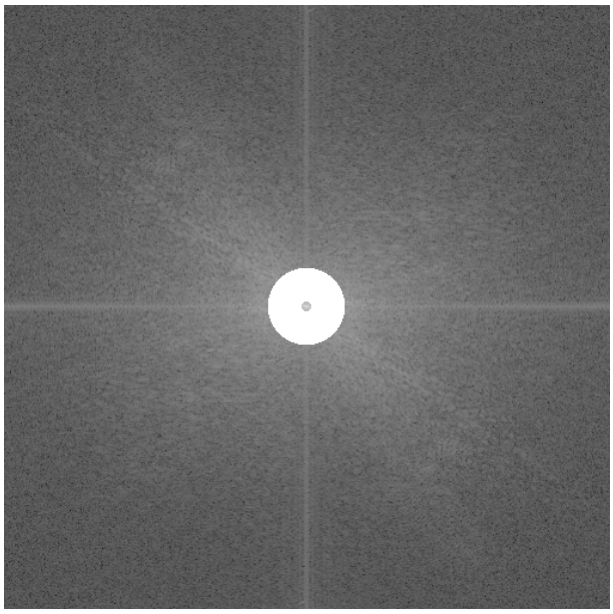
```
#FILTRANDO O CIRCULO CENTRAL QUE CONTEM AS ALATS FREQUENCIAS
x0=256
y0=256
FT=F
FT=fp.fftshift(FT)
for x in range(512):
    for y in range(512):
        if (x-x0)**2+(y-y0)**2>20 and (x-x0)**2+(y-y0)**2<1000:
            FT[x][y]=0

FTT = np.absolute(FT)
FTT /= FTT.max() #NORMALIZA EM [0,1]
FTT = np.log(FTT)

#APRESENTANDO A |FT| FILTRADA
plt.figure()
plt.title('Imagem da FT filtrada')
plt.imshow(FTT, cmap=cmap)
plt.colorbar()

IFT=fp.ifft2(FT)
IFT=np.absolute(IFT)

plt.figure()
plt.title('Imagem resultante')
plt.imshow(IFT, cmap=cmap)
plt.colorbar()

plt.show()
```

4.6.5 Filtro Banda Passante Circular Externo

Esse filtro faz o oposto do Filtro Banda Passante Circular Interno ele corta as frequências entre os dois círculos e deixa passar as demais frequências. Pode-se concluir que as frequências entre os círculos são de frequências baixas e muito altas.

Quando utilizamos esse filtro é para obtermos as feições de maiores detalhes e regionais.

Figura 49 – Imagem com o Filtro Banda Passante Circular Interno



Espectro filtrado: Banda Passante Circular Interno

Imagem filtrada

Fonte: O autor.

Para implementação do algoritmo basta trocar os sentidos dos sinais de menor para maior e de maior para menor, além de trocar o conectivo condicional de *"and"* para *"or"*.

Figura 50 – Imagem com o Filtro Banda Passante Circular Externo

Espectro filtrado: Banda Passante Circular Externo

Imagem filtrada

Fonte: O autor.

5 APLICATIVO DE FILTROS

Um dos principais objetivos desse trabalho foi a criação de um aplicativo que auxiliasse o ensino de processamento de imagens digitais nos níveis mais básicos de ensino. O conteúdo a ser trabalhado pode ser tanto média aritmética como nos filtros de conversão para escalas de cinza quanto no ensino de matrizes usando por exemplos os operadores matemáticos e lógicos, por esse motivo desenvolvemos um aplicativo para levar os primeiros contatos com o universo do PDI.

Este aplicativo visa intermediar o primeiro contato de professores e alunos aprender uma aplicabilidade da matemática aliada com as TIC's.

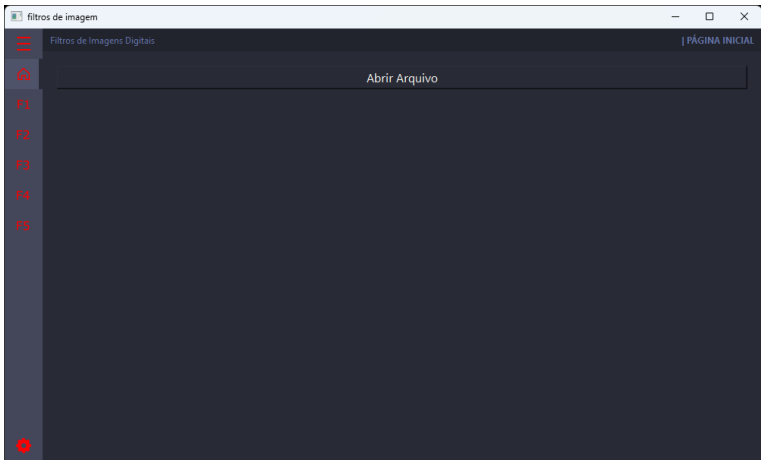
5.1 Construção do Aplicativo

Decidimos utilizar a linguagem de programação Python em toda a estrutura do aplicativo pois esse tem uma linguagem simplificada e de fácil entendimento, a interface gráfica foi planejada dentro do ambiente QT Designer.

5.2 Funcionalidades do aplicativo

O aplicativo conta com uma barra de menu lateral onde encontra-se 7 botões, a saber: o botão de expansão e contração do menu lateral, um botão de página inicial e 5 botões que representam filtros a serem aplicados.

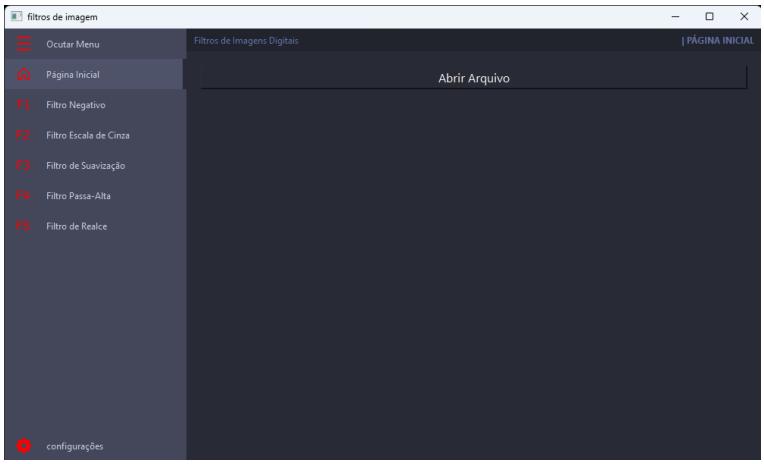
Figura 51 – Página Inicial do Aplicativo.



Fonte: O autor.

Na página inicial do aplicativo contém um botão de diálogo que pesquisa um arquivo JPG do computador, essa imagem é exposta logo abaixo do botão de diálogo.

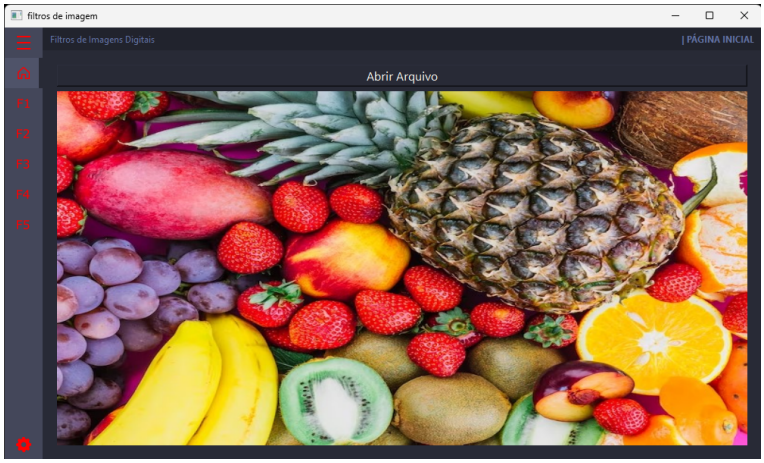
Figura 52 – Página Inicial do Aplicativo - Menu lateral expandido.



Fonte: O autor.

Ao clicar no botão **Abrir Arquivo** na página inicial é selecionada a imagem e esta aparece logo abaixo ao botão.

Figura 53 – Página Inicial do Aplicativo - Imagem selecionada.



Fonte: O autor.

5.2.1 Botão F1 - Filtro Negativo

O Botão **F1** no menu lateral abre uma página com um botão que aplica o filtro negativo na imagem apresentado em 4.1.2.4, ao lado da imagem tem um pequeno texto de explicação do filtro, falando o seguinte: Cada pixel tem 3 valores que representam o RGB, cada um desses valores vai de 0 a 255, para que a imagem fiquei com tons negativos basta fazer a seguinte operação matemática: cada valor será subtraído de 255. Logo dado um pixel p = (9, 150, 250) esse filtro mudará esse pixel para p = (255-9, 255-150, 255-250) = (246,155,5). Fazendo isso com todos os temos o efeito negativo.

Figura 54 – Página do Filtro Negativo.

Fonte: O autor.

5.2.2 Botão F2 - Escala de Cinza

O Botão **F2** no menu lateral abre uma página com um botão que aplica o filtro de conversão da imagem para a escala de cinza, apresentando também em 4.1.2.4, com o seguinte texto: Cada pixel é composto por 3 valores que representam as cores RGB, ou seja, dado um pixel p = (25, 60, 120) esse filtro faz a média aritmética desses 3 valores (30+60+120)//3=70. Logo o pixel ficará com os valores p = (70, 70, 70). Fazendo isso com todos os temos o efeito da imagem em escalas de cinza.

Figura 55 – Página do Filtro de conversão em escala de cinza.

Fonte: O autor.

5.2.3 Botão F3 - Média dos pixels

O Botão **F3** no menu lateral abre uma página com um botão que aplica o filtro de suavização com o operador média apresentado em 4.4.1, logo ao lado da

imagem temos o texto a seguir: Cada pixel é conectado por outros 8 pixels ao seu redor, com exceção dos pixels da borda imagem, esse filtro considera uma vizinhança de raio 2, ou seja, considera 24 pixels ao redor do pixel analisado e calcula a média aritmética simples desses 25 pixels e substitui o valor do pixel analisado por essa média, fazendo assim um efeito de suavização da imagem que chega a borrar seus traços mais marcantes.

Figura 56 – Página do Filtro de Suavização com o operador média.

Fonte: O autor.

5.2.4 Botão F4 - Passa - Alta Básico

O Botão **F4** no menu lateral abre uma página com um botão que aplica o filtro de Passa-Alta Básico 4.5.1, do lado da imagem contém o texto: Regiões da imagem com cores próximas são chamadas de homogêneas, pixels que compõem uma linha mais marcante por exemplo tem valores que se destacam de sua vizinhança, esse filtro identifica esses valores que se destacam e deixam passar já os outros valores são ignorados. Fazendo isso com toda a imagem temos o efeito de mostrar somente as altas frequências de cores da imagem.

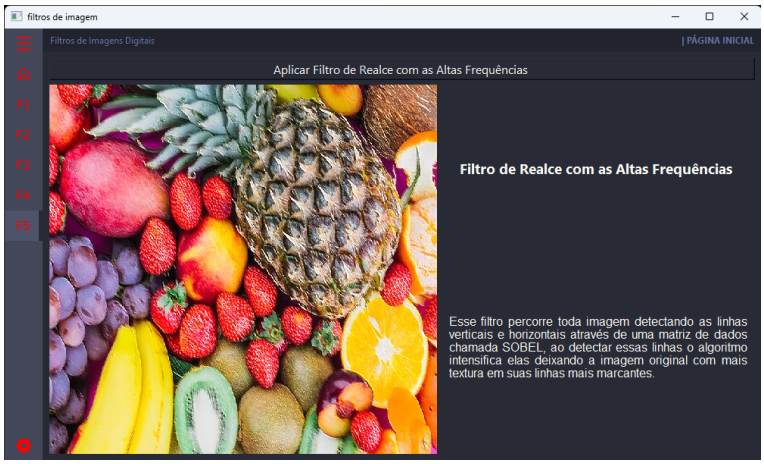
5.2.5 Botão F5 - Sobel

O Botão **F5** no menu lateral abre uma página com um botão que aplica o filtro de Sobel apresentado na seção 4.5.1, do lado da imagem contém o texto: Esse Filtro percorre toda imagem detectando as linhas verticais e horizontais através de uma matriz de dados chamada Sobel, ao detectar essas linhas o algoritmo as intensifica deixando a imagem original com mais textura em suas linhas mais marcantes.

Figura 57 – Página do Filtro de Passa-Alta Básico.

Fonte: O autor.

Figura 58 – Página do Filtro de Realce Sobel com o operador soma com a imagem original.



Fonte: O autor.

6 CONCLUSÕES

O ensino de algoritmos no ensino da matemática é de extrema importância de acordo com a Base Nacional Comum Curricular (BNCC). Esta introduziu mudanças significativas na área de matemática, visando fornecer uma educação matemática mais abrangente e significativa, preparando os alunos para aplicações do mundo real e estimulando habilidades de pensamento crítico (BRASIL, 2018).

Nesse trabalho buscamos apresentar o Processamento de Imagens Digitais (PDI) como justificativa para o ensino de algumas dessas ferramentas matemáticas através de algoritmos computacionais, portanto o trabalho se apresentou como um compilado de algoritmos que podem ser trabalhados com estudantes de todos os níveis de ensino, desde média aritmética a transformada de Fourier.

Além disso foi mostrado uma outra maneira de olhar as matrizes no Ensino Médio, adição, subtração, multiplicação e divisão foram alguns dos algoritmos abordados que trazem um efeito visual para as imagens matriciais.

Por fim, como adicional ao trabalho apresentamos um aplicativo que pode auxiliar o professor e o aluno que queria ter os primeiros contatos com o PDI.

Para os trabalhos futuros temos buscaremos melhorar interface do aplicativo colocando mais filtros e mais páginas explicativas, criação e aplicação de roteiros de aula para guiar professores quanto a utilização desse trabalho.

REFERÊNCIAS

BRASIL. Base nacional comum curricular. Ministério da Educação, Brasília, DF, 2018.

CORMEN, T. H. et al. *Algoritmos: Teoria e Prática*. 6°. ed. Rio de Janeiro: Elsevier, 2012.

CUNHA, D. S. O uso da tecnologia no ensino da matemática: contribuições do software geogebra no ensino da função do 1º grau. *Revista Educação Pública*, v. 21, n. 36, p. https://educacaopublica.cecierj.edu.br/artigos/21/36/o–uso–da–tecnologia–no–ensino–da–matematica–contribuicoes–do–isoftwarei–geogebra–no–ensino–da–funcao–do–1–grau, 2021.

FIGUEIREDO, D. G. *Análise de Fourier e Equações Diferenciais Parciais*. 5°. ed. Rio de Janeiro: IMPA: IMPA, 2018.

FILHO, O. M.; NETO, H. V. *Processamento Digital de Imagens*. 1º. ed. Brasília, DF: Brasport, 1999.

GOMES, J.; VELHO, L. *Fundamentos da Computação Gráfica*. 1º. ed. Rio de Janeiro - RJ: IMPA, 2015.

GONZALES, R. C.; WOODS, R. E. *Processamento de Imagens Digitais*. 3°. ed. Brasília, DF: Pearson Prentice Hall, 2010.

IGNáCIO, J. Processamento e análise digital de imagens em estudos da cinéticade recristalização de ligas al-mg-x. Dissertação (Mestrado em Ciências)- Instituto de Pesquisas Energéticas e Nucleares, Universidade de São Paulo, São Paulo, SP, p. https://www.teses.usp.br/teses/disponiveis/85/85134/tde–14042014–135224/publico/2013IgnacioProcessamento.pdf, 2013.

LIMA, E. L. *Espaços Métricos*. 5°. ed. Rio de Janeiro: IMPA: Projeto Euclides, 2013.

MENESES, P. R. et al. *Introdução Ao Processamento de Imagens de Sensoriamento Remoto*. 1°. ed. Brasília: UnB - CNPq, 2012.

MORETTIN, P. A.; BUSSAB, W. de O. *Estatística Básica*. Tradução da 3° edição americana. São Paulo: Saraiva, 2010.