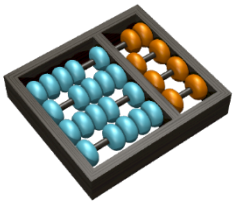
Universidade Estadual de Campinas
Instituto de Computação



Disciplina do 2º Semestre de 2020

IC - UNICAMP

Curso: Bacharelado em Ciência da Computação

MC920 - Introdução ao Processamento de Imagens Digitais

Projeto 0 - Processamento de Imagens Monocromáticas

Alunos: Gabriel Volpato Giliotti **RA**: 197569

Professores: Hélio Pedrini

Campinas – SP
2020

Trabalho 0 - Introdução ao Processamento de Imagem Digital

Nome: Gabriel Volpato Giliotti - **RA:** 197569

1 - Introdução

O objetivo deste trabalho é implementar algoritmos para diferentes tipos de manipulações no processamento de imagens, como aplicar filtros de clareamento, brilho, realizar recortes de imagens, rotação de bits, entre outras aplicações que serão descritas posteriormente nesse documento.

Atualmente, o processamento de imagens é um artifício muito utilizados em mídias sociais e aparelhos eletrônicos, para aplicação de filtros e outros recursos em imagens, nos oferecendo vasto repertório de técnicas que podem ser aplicadas. Com isso, o projeto zero visa introduzir os conceitos mais básicos de processamento de imagens, oferecendo base para o entendimento de como algumas técnicas podem ser aplicadas em imagens monocromáticas.

Junto desse relatório será entregue o arquivo .zip ou .tgz, que possui todos os arquivos citados nesse relatório e que foram utilizados para processar as imagens dadas como entrada.

2 - Programas

Os scripts para processamento das imagens foram implementados em Python 3.7.6 junto das bibliotecas Numpy 1.19.2 e OpenCV 4.4.0 que são voltadas para o processamento de imagens. Outras bibliotecas como Math e Sys também foram utilizadas mas não oferecem recursos específicos para processamento de imagens.

2.1 - Como Executar

Inicialmente, antes de executar qualquer um dos scripts, devemos criar nove pastas correspondentes para cada arquivo .py definido onde serão renderizadas as imagens processadas. Para se obter as imagens de saída você deve criar as pastas no mesmo diretório onde executará os arquivos .py. Os nomes dos diretórios são **:**

1 - NegativeOutputs
2 - MirrorOutputs
3 - IntensityOutputs
4 - InverseLinesOutputs
5 - ReflectionLinesOutputs
6 - BrightAdjustOutputs
7 - BitPlaneOutputs
8 - MosaicOutputs
9 - MergeImagesOutputs

Para o projeto foram criados nove arquivos com extensão do tipo .py onde cada um é responsável por uma tarefa específica exigida no enunciado. A seguir estão os formatos de construção para chamada de cada arquivo via linha de comando:

1 - C:\> python negativo.py pathImagem
2 - C:\> python espelhamento.py pathImagem
3 - C:\> python intervalo100200.py pathImagem
4 - C:\> python inversaoLinhasPares.py pathImagem
5 - C:\> python reflexaoLinhasSuperiores.py pathImagem
6 - C:\> python ajustaBrilho.py pathImagem valorDeGamaDiferenteDeZero
7 - C:\> python extrairPlanoBits.py pathImagem valorPlanoDe0a7
8 - C:\> python mosaico.py pathImagem
9 - C:\> python combinaImagens.py pathImagem1 pathImagem2 valorPond1 valorPond2

Substitua "pathImagem" pelo(s) caminho(s) correto(s) da(s) imagem(ns) e os valores necessários nas chamadas que os requisitam. As chamadas enumeradas de 1 a 9 correspondem as pastas numeradas de 1 a 9 que deverão ser criadas para receber as saídas

2.2 - Entradas

Como entrada para qualquer um dos arquivos é necessário o path da imagem ou das imagens a serem processadas e se necessário, o valor ou valores requisitados de acordo com o técnica a ser aplicada sobre a ou as imagens. Todas as imagens devem ser Monocromáticas , no formato PNG (Portable Network Graphics) e com resolução de 512x512 pixels. As imagens utilizadas para esse projeto vieram de um repositório indicado pelo professor e estão compactadas no .zip ou .tgz junto aos arquivos .py em uma pasta chamada ImagensMono.

2.3 - Saídas

As saídas são obtidas nas pastas criadas previamente antes da execução dos arquivos. Todas as imagens serão Monocromáticas , no formato PNG (Portable Network Graphics) e com resolução de 512x512 pixels.

3 - Parâmetros Utilizados

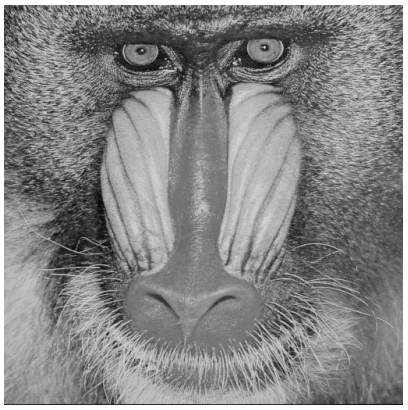
Os parâmetros utilizados para esse projeto são o path da imagem ou imagens a serem processadas, valor gamma diferente de zero responsável por definir o ajuste de brilho das imagens (quanto maior gamma, maior o brilho), valor do plano de bits o qual queremos obter a imagem (pode variar de 0 a 7) e valores de ponderação para combinar as imagens (que definem quantos % de cada imagem deve aparecer na imagem de saída).

4 - Soluções

4.1 - Transformações de Intensidade

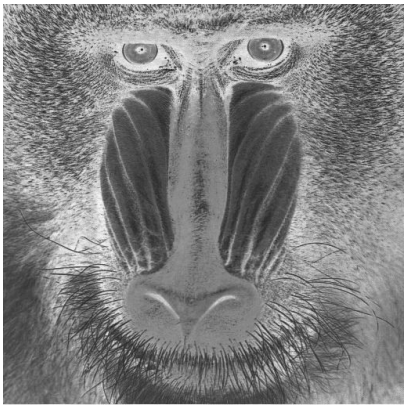
Como uma primeira transformação de intensidade apresentamos o negativo de imagens, onde realizamos a leitura da imagem de entrada com a função imread da biblioteca OpenCV e, sendo um array o retorno da função, aplicamos uma subtração de valor 255 sobre toda a matriz de bits da imagem, fazendo uso do artifício de vetorização. Assim obtemos a inversão dos valores dos bits e, consequentemente, o negativo da imagem. Observe que:

Se valor do pixel = 0, então 255 - 0 = 255
Se valor do pixel = 1, então 255 - 1 = 254
Se valor do pixel = 2, então 255 - 2 = 253 …



baboon.png Original

city.png Original



baboon.png Negativo

city.png Negativo

Nesse caso, não foi necessário realizar truncamentos ou arredondamento dos valores para cada pixel das imagens lidas, pois assumimos que os níveis de cinza da imagem original variam no intervalo [0, 255]. Assim, garantimos a inversão apenas com a subtração indicada anteriormente.

A segunda transformação no tópico de intensidade foi o espelhamento vertical. Nesse caso, não realizamos nenhuma operação direta nos bits. Na verdade, por fins de simplicidade, o que ocorre é a rotação da imagem de entrada, fazendo uso da função flip da biblioteca OpenCV, de forma que o resultado obtido é o espelhamento. Tal função aceita como parâmetro o array da imagem de entrada e um valor inteiro entre [-1,1] indicando o tipo de rotação.

Tipos de rotação:
cv2.flip(imgArray, 0) Rotação da imagem original no sentido vertical
cv2.flip(imgArray, 1) Rotação da imagem original no sentido horizontal
cv2.flip(imgArray, -1) Rotação da imagem original no sentido vertical e horizontal

Só é possível obter um dos tipos de rotação por vez. Para isso, acesse o código e modifique as linhas com a mensagem “Descomente ou comente aqui”.

Imagem city.png Original

Imagem Espelhada Verticalmente

Imagem Espelhada horizontalmente

Imagem Espelhada em ambos os sentidos

A terceira transformação no tópico de intensidade foi a conversão de intensidade do intervalo de pixels de [0,255] para [100,200]. Nesse caso o algoritmo foi implementado de duas formas. No primeiro formato, fazemos o uso de vetorização e aplicamos a expressão:

```
newImg = (imgInput/255) * 100) + 100
```

para todos os bits lidos no array da imagem de entrada. Já no segundo formato, o que fazemos é uma busca do menor e maior valores de pixels existentes na imagem e em seguida a aplicação da expressão:

```
newImg = (((imgInput - menorValor)/(maiorValor - menorValor)) * 100) + 100
```

gerando imagens levemente diferentes quanto a intensidade dos pixels. No primeiro caso aplicamos uma conversão dos valores dos pixels para inteiro, pois dada a fórmula utilizada, os valores dos pixels foram convertidos para ponto flutuante. Para segundo caso aplicamos um truncamento dos valores dos pixels, mostrando que é possível aplicar ambos os métodos para se tratar os pixels convertidos para ponto flutuante.


Img city.png Original


Img city.png algoritmo 2


Img city.png algoritmo 1

Repare que a imagem de saída obtida a partir do algoritmo 2 é uma pouco mais clara que a imagem obtida a partir do algoritmo 1 (Para melhor visualizar compare os pontos mais escuros entre as imagens). Além disso, um outro processo realizado foi a inversão dos tratamentos dos pixels para cada algoritmo. Para ambos os algoritmos aplicamos o tratamento através da conversão dos valores dos pixels para inteiro, quanto o truncamento. Nenhuma imagem foi apresentada aqui pois não foram obtidos resultados visuais significativos para as conversões realizadas em um mesmo algoritmo. Para criar a imagem de saída utilizamos a função imwrite da biblioteca OpenCV.

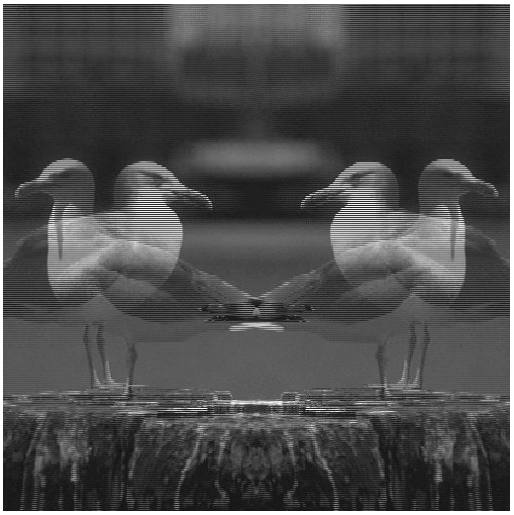
Para executar as comparações, acesso o código e comente um dos modos enquanto executa o outro. Cada método apresenta um comentário indicando seu trecho de código correspondente.

A quarta transformação no tópico de intensidade foi a inversão de linhas pares das imagens. Para esse processamento, fazemos a leitura da imagem com a função imread da biblioteca OpenCV e através do recurso de slice disponível pela linguagem python, criamos duas cópias reduzidas do array original que representa a imagem de entrada. No primeiro array chamado de list1, fazemos uma cópia das linhas pares da imagem, já realizando a inversão dos valores dos pixels para cada linha. No segundo array, chamado de list2, fazemos uma cópia das linhas ímpares da imagem sem qualquer alteração. Finalmente criamos uma lista result com os tamanhos somados de list1 e list2, fazemos a leitura das linhas pares e ímpares em seus locais correspondentes através do uso do slice e transformamos a lista result de saída em um array com a função np.array() disponível na biblioteca Numpy.

Img city.png Original

Img seagull.png Original

Img city.png linhas pares invertidas

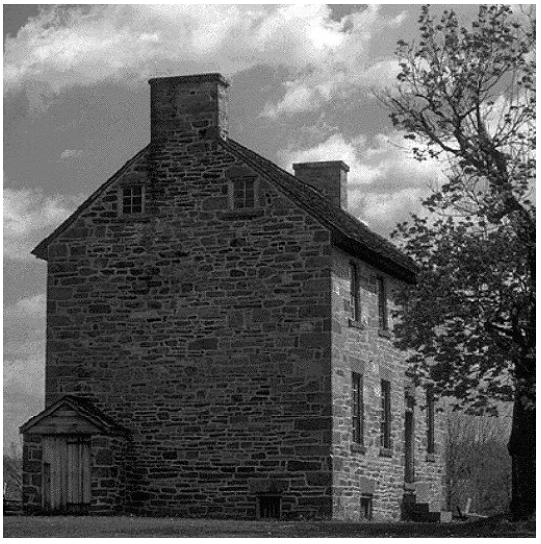


Img seagull.png linhas pares invertidas

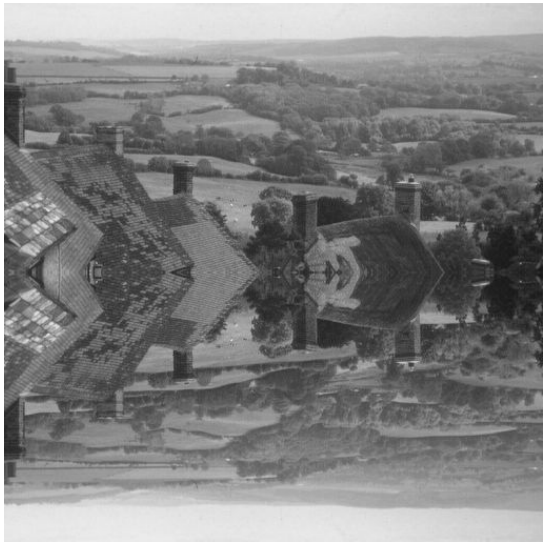
A impressão que temos de imagem sobreposta é devido a inversão dos pixels. Fazer uma aproximação ou afastamento da imagem, assim como aumenta-la ou diminuí-la pode trazer uma interpretação diferente para os olhos.

A quinta e última transformação no tópico de intensidade foi a reflexão das linhas superiores na metade inferior da imagem. Para essa transformação, utilizamos dos mesmos artifícios aplicados na transformação de linhas pares, fazendo uso de duas listas (list1 e list2) e também de slice. Desse modo, na primeira lista (list1) armazenamos a metade superior da imagem de entrada, contendo metade dos valores dos pixels sem qualquer modificação. Na segunda lista (list2), fazemos a leitura também da primeira metade da imagem de entrada, porém de forma reversa para conseguir o efeito de reflexão desejado. Finalmente, copiamos na primeira metade de result a list1 contendo os pixels da parte superior e, em seguida, copiamos list2 que contém os pixels da primeira metade da imagem na ordem reversa, obtendo o efeito desejado. Por último, aplicamos np.array() da biblioteca Numpy e imwrite da biblioteca OpenCV para renderizar a imagem de saída.

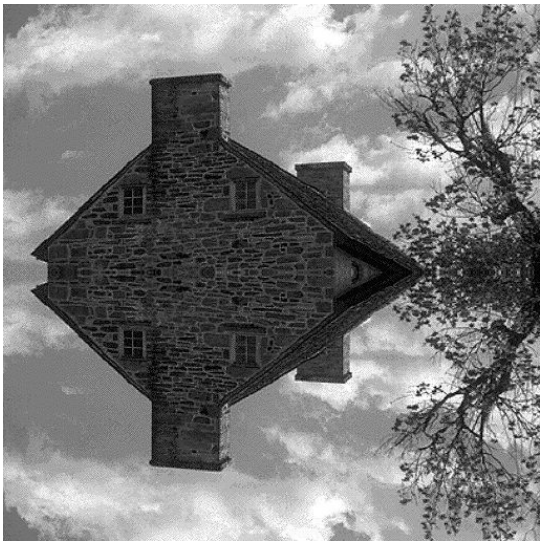
Img city.png Original



Img house.png Original



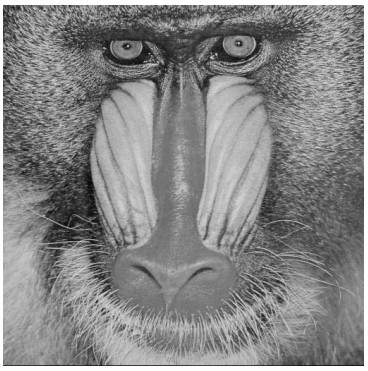
Img city.png Reflexão de linhas superiores



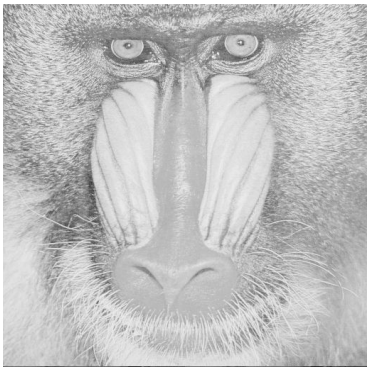
Img house.png Reflexão de linhas superiores

4.2 - Ajuste de Brilho

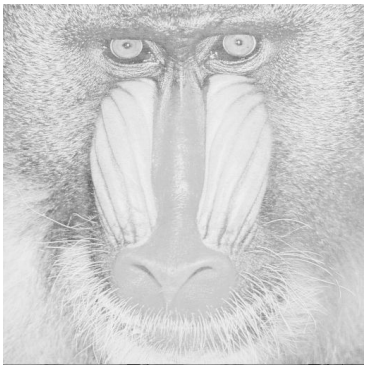
Para o ajuste de brilho, o algoritmo executado segue a descrição dada pelo enunciado. Com isso, aplicamos diferentes valores de gamma para ajustar o brilho de uma imagem de entrada. Como saída, foi testado um limite do valor gamma equivalente a 40, onde ainda é possível visualizar algum conteúdo ainda presente da imagem de entrada original. Além disso, não existe valor máximo definido para gamma, mas a partir de certos valores, a imagem passa a ser não reconhecível aos olhos, sendo um grande quadrado branco. (O valor de gamma onde se vê um limite para aplicar o ajuste de brilho pode variar de pessoa para pessoa, de acordo com a capacidade visual do usuário. No meu caso, a partir de gamma = 40 já não consigo verificar qualquer traço da imagem original.)



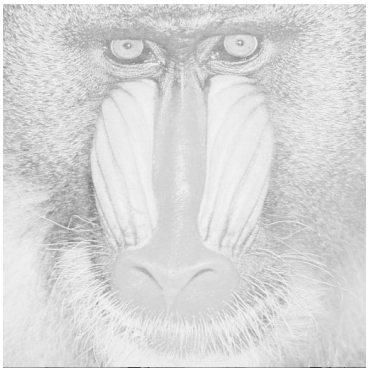
Img baboon.png Original



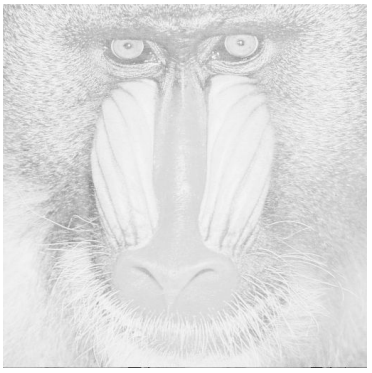
Img baboon.png gamma 2.5



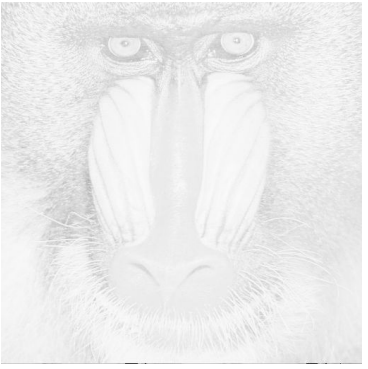
Img baboon.png gamma 3.5



Img baboon.png gamma 3.5



Img baboon.png gamma 4.5



Img baboon.png gamma 10

Img baboon.png gamma 25

Img baboon.png gamma 40

4.3 - Plano de Bits

Para o plano de bits, temos que primeiro entender que além da imagem de entrada ser uma matriz de bits 512x512, ela também apresenta diferentes planos para um mesmo bit. Tais planos nada mais são que as definições dos valores de bits para cada pixel da imagem. Logo para um único pixel temos o seguinte formato:

Bits	0	1	0	1	1	0	1	0
Planos	7	6	5	4	3	2	1	0

Exemplo de planos de bits para um pixel

No caso de exemplo, temos que o valor do pixel é 90. Então, aplicando operações bitwise disponíveis em python, conseguimos fazer um shift dos valores binários que representam o valor do pixel dado. Logo, aplicando a operação de shift right (90 >> 2) obtemos uma novo valor para o pixel, que é 22:

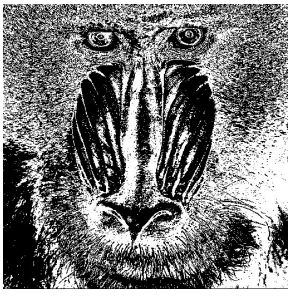
Bits	0	0	0	1	0	1	1	0
Planos	7	6	5	4	3	2	1	0

Exemplo de planos de bits para um pixel após aplicar shift right 2 vezes (>> 2)

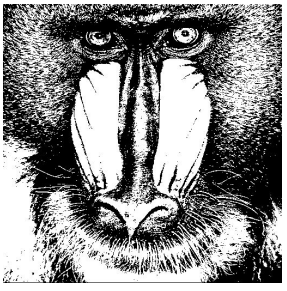
Além do shift right, precisamos descobrir se no plano de uma dada ordem de entrada, o pixel tem valor 0 ou 1. Assim, buscamos o resto da divisão por 2 (por serem binários) e multiplicamos por 255 para definir se o bit naquele plano tem representação de cor preta ou branca. Observe alguns dos planos de bits de uma imagem dada como entrada e, que entre os planos 6 e 7 ocorre quase que uma inversão de coloração entre as imagens. Logo, quando todos os planos são sobrepostos, temos o tom acinzentado apresentado na imagem original.

Plano de Bits 4

Plano de Bits 5



Plano de Bits 6

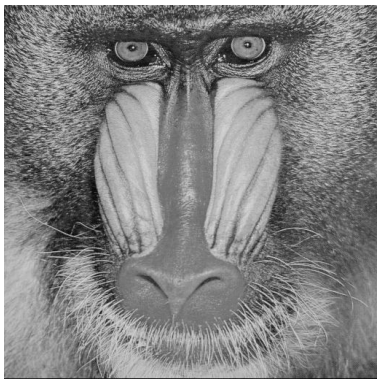


Plano de Bits 7

4.4 - Mosaico

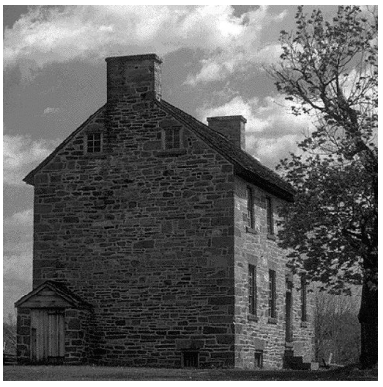
Para o mosaico, assim como qualquer outra seção do projeto, poderiam ser feitos diferentes algoritmos. Aqui, foi optado por um algoritmo menos automatizado, que apenas processa imagens de formato 512x512 pixels e Monocromáticas, em contrapartida à não utilização de laços de repetição. Em processamento de imagens, utilizar estruturas de laços podem interferir na eficiência do código que processa a imagem, fazendo com que os recortes e construções das imagem levem algum tempo. Logo não utilizamos esse tipo de estrutura o que torna o código menos versátil ao processamento da entrada dada, mas faz o tratamento rapidamente no caso de uma entrada que cumpre os requisitos necessários.

A estrutura do mosaico tem formato fixo pré-definido no enunciado, logo para qualquer imagem monocromática de 512x512 pixels, o resultado é um mosaico com os pedaços da imagem original construídos de forma a corresponder com a enumeração dada.

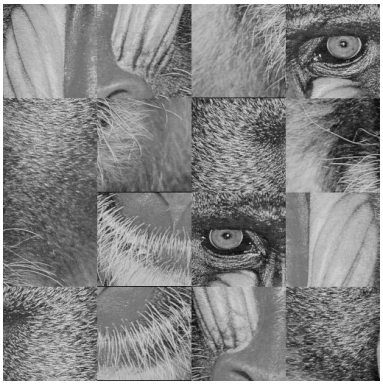


Img baboon.png Original

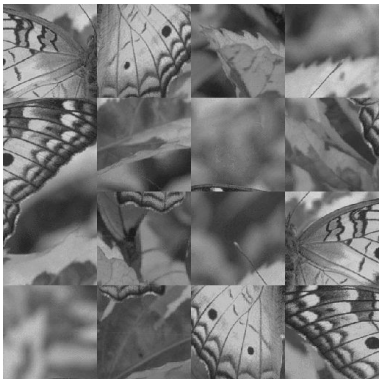
Img butterfly.png Original



Img house.png Original



Img baboon.png mosaico



Img butterfly.png mosaico

Img house.png mosaico

4.5 - Combinação de Imagens

Finalmente, para a combinação de imagens é necessário passar por parâmetro quatro informações: Path da imagem 1, Path da imagem 2, valor de ponderação no intervalo [0,1] para imagem 1 e valor de ponderação no intervalo [0,1] para imagem 2. Desse modo, fazemos a leitura das imagens separadamente e, após esse passo, multiplicamos os arrays das imagens pelos seus respectivos valores de ponderação. Finalmente, fazemos uma soma vetorizada das imagens, onde a imagem com maior valor de ponderação prevalece sobre a imagem de menor valor.

Merge de seagull.png e butterfly.png com valores 0.5 de ponderação (para ambas)

4.5 - Limitações no Projeto

Como limitações pertinentes ao projeto, temos inicialmente a falta de experiência com a utilização da linguagem python, além de não conhecer todo o repertório oferecido pelas bibliotecas utilizadas e disponíveis. Além disso, foi recomendado a não utilização de laços de repetição para processar as imagens, o que traz um pouco de limitação para se tentar resolver os problemas. Além disso, alguns dos algoritmos são somente aplicáveis em determinados formatos de imagens (monocromáticas e de tamanho 512x512 pixels), com valores de entrada específicos.

4.6 - Conclusão

Através de cada seção do projeto 0, pudemos obter um primeiro contato com o Processamento de Imagens e observar as primeiras dificuldades que existem no tratamento de imagens monocromáticas, através de diferentes tipos de processamento. Além disso, experienciamos o tempo estimado necessário para se construir o relatório que deve acompanhar a entrega, assim como os algoritmos implementados, mesmo com bibliotecas dedicadas para tais fins.