

Universidade Federal do Agreste de Pernambuco**Bacharelado em Ciências da Computação****Prof. Tiago Buarque A. de Carvalho**

Reconhecimento de Padrões**Exercícios da Semana 01 – Introdução****Processamento de Imagem Digital****Aluno: Vinícius Santos de Almeida**

-
- 1.

- Nessa questão carreguei a imagem usando a classe fornecida `ImagenDigital` e foi bem simples criar o laço duplo com `for`, usei uma `ArrayList` para armazenar os valores e usei o método `contains()` para verificar se cada valor encontrado já existe na lista, caso não exista, então é um valor distinto, então adiciona à lista. A saída, mostrada abaixo, comprova que os retângulos da imagem de fato possuem intensidade constante:

```
32
64
96
128
160
192
```

- A implementação ficou como mostrado a seguir:

```
import java.util.ArrayList;

public class Questao1 {

    public static void main(String args[]) {
        int[][] imagem =
ImagenDigital.carregarImagem("./static/figura1.png");

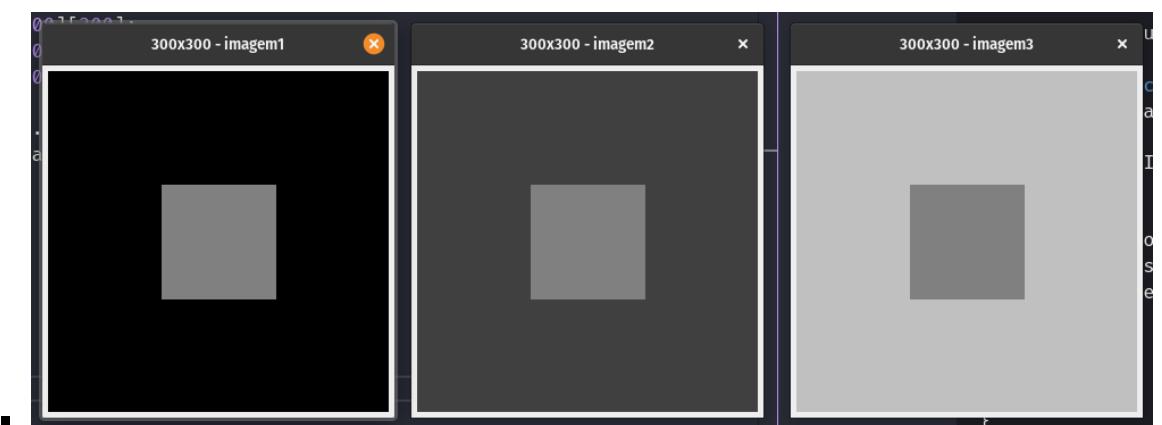
        ArrayList<Integer> novosPixelis = new ArrayList<Integer>();
        for (int i = 0; i < imagem.length; i++) {
            for (int j = 0; j < imagem[i].length; j++) {
                if (!novosPixelis.contains(imagem[i][j])) {
                    novosPixelis.add(imagem[i][j]);
                    System.out.println(imagem[i][j]);
                }
            }
        }
    }
}
```

```
        }
    }
}
```

- 2.

- Foi necessário gerar três vetores de duas dimensões de 300 x 300 de tamanho, cada um representando uma figura, em seguida, percorri um dos vetores fazendo a verificação se `j < 100 ou j > 200 ou i < 100 ou i > 200` então deveria colorir os três vetores na posição `[i][j]`, cada vetor diferente com uma das 3 cores 0, 64, e 192, no `else` eu colori o quadrado do meio, sempre 128, dessa forma, todos os quadrados do meio de fato têm a mesma cor.

- Saída:



- Implementação:

```
public class Questao2 {

    public static void main(String args[]) {
        int[][][] imagemGerada1 = new int[300][300];
        int[][][] imagemGerada2 = new int[300][300];
        int[][][] imagemGerada3 = new int[300][300];

        for (int i = 0; i < imagemGerada1.length; i++) {
            for (int j = 0; j < imagemGerada1[i].length; j++) {
                if (j < 100 || j > 200 || i < 100 || i > 200) {
                    imagemGerada1[i][j] = 0;
                    imagemGerada2[i][j] = 64;
                    imagemGerada3[i][j] = 192;
                } else {
                    imagemGerada1[i][j] = 128;
                    imagemGerada2[i][j] = 128;
                    imagemGerada3[i][j] = 128;
                }
            }
        }

        ImagemDigital.plotarImagem(imagemGerada1, "imagem1");
        ImagemDigital.plotarImagem(imagemGerada2, "imagem2");
        ImagemDigital.plotarImagem(imagemGerada3, "imagem3");
    }
}
```

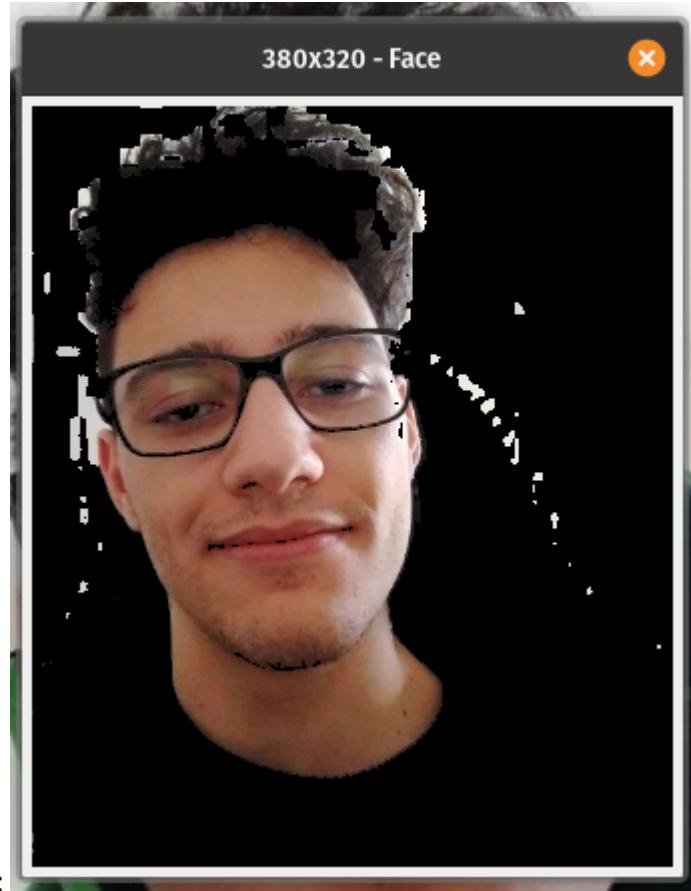
```
    }  
}
```

- 3. a)

- Nessa questão, após carregar a minha imagem, criei um novo vetor 3×3 para colocar a imagem gerada da face. Para isso percorro cada linha e coluna da imagem, e onde o índice $[i][j][0]$ for maior que $[i][j][1]$ e $[i][j][0]$ for maior que $[i][j][2]$ é salvo o valor de $[i][j]$ em face, dividindo os valores de i e j por 2 em cada para escalar e imagem menor, além disso, subtraio alguns valores ainda de i e j para tentar posicionar a face o mais perto da origem possível sem perder informações, também adiciono mais algumas condições no `if` para tentar remover o cabelo da imagem da face, esses dois últimos fiz na tentativa e erro. Segue abaixo a entrada e saída:

- Entrada:





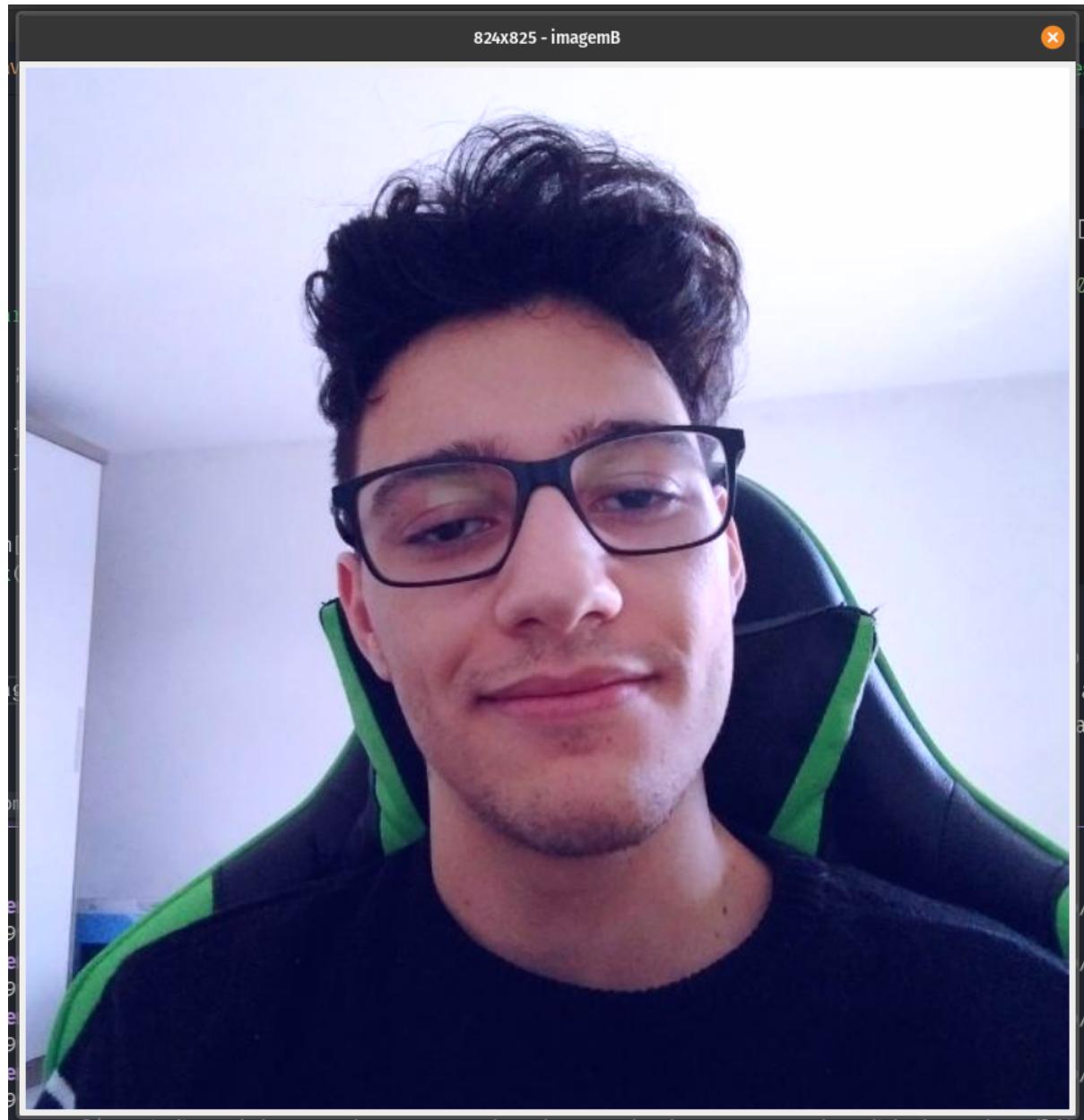
- Saída:
- Implementação:

```
import java.math.*;  
  
public class Questao3 {  
  
    public static void main(String args[]) {  
        int[][][] imagem =  
ImagemDigital.carregarImagenCor("./static/eu.png");  
        int[][][] face = new int[320][380][3];  
  
        for (int i = 0; i < imagem.length; i++) {  
            for (int j = 0; j < imagem[i].length; j++) {  
                if (  
                    (imagem[i][j][0] > imagem[i][j][1] && imagem[i][j]  
[0] > imagem[i][j][2])  
                    && (imagem[i][j][0] > 20 && imagem[i][j][1] > 20  
&& imagem[i][j][2] > 20)) {  
                    face[Math.max(0, i/2-90)][Math.max(0, j/2-30)] =  
imagem[i][j];  
                }  
            }  
        }  
  
        ImagemDigital.plotarImagenCor(imagem, "Eu");  
        ImagemDigital.plotarImagenCor(face, "Face");  
    }  
}
```

- 3. b)

- Usei o código da questão anterior, criando um novo vetor para representar a imagem azulada e fazendo ajustes no loop, atribuo ao novo vetor todos os valores encontrados na imagem original em suas mesmas posições, porém somando 40 ao valor do índice `[i][j][2]` para ficar notável que foi adicionado mais azul à imagem.

- Saída:



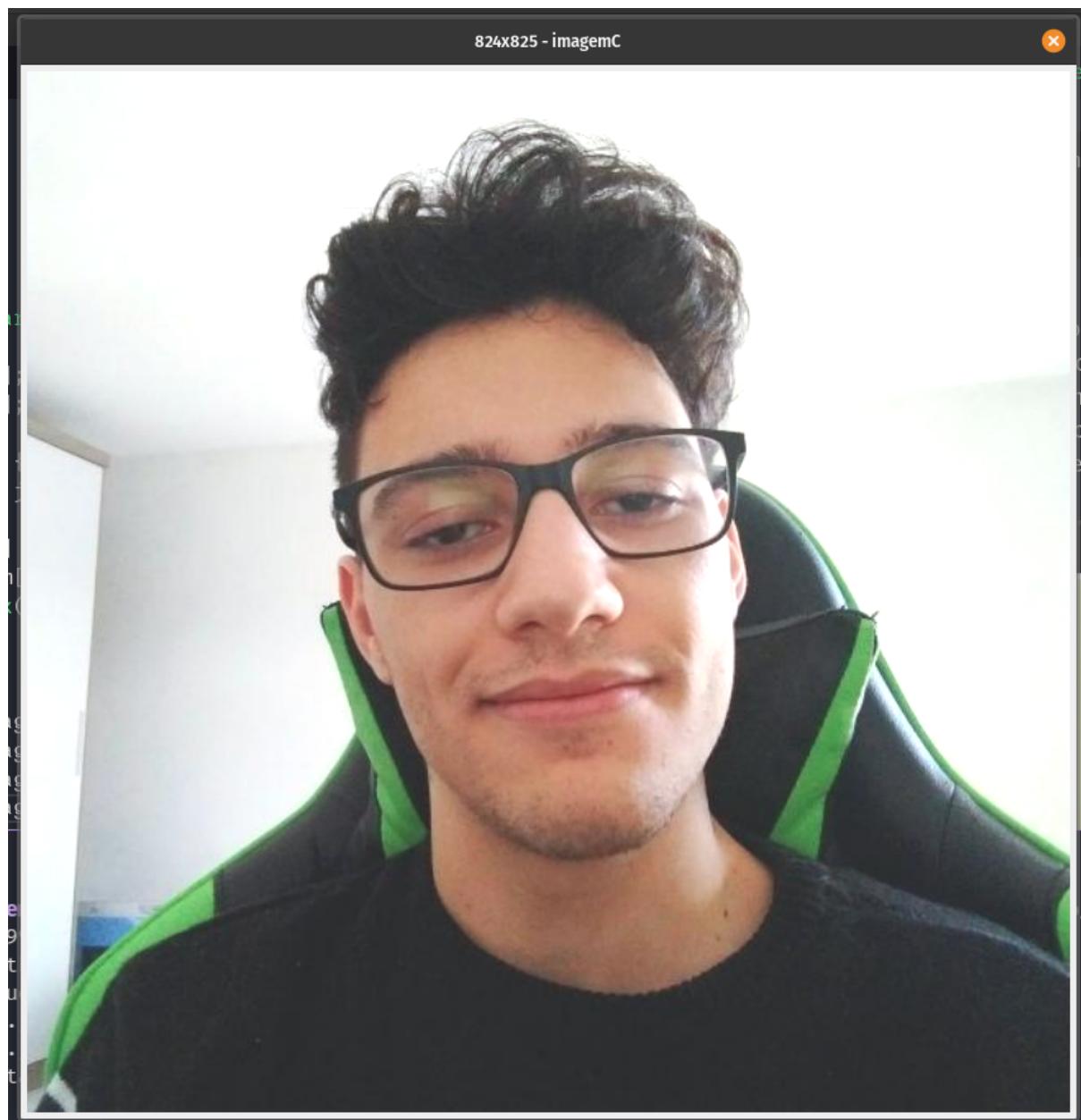
- Implementação (partes adicionadas ao `main()` e aos `for`):

```
public static void main(String args[]) {  
    ...  
    int[][][] imagemB = new int[825][824][3];  
  
    for (int i = 0; i < imagem.length; i++) {  
        for (int j = 0; j < imagem[i].length; j++) {  
            ...  
            imagemB[i][j][0] = imagem[i][j][0];  
            imagemB[i][j][1] = imagem[i][j][1];  
            imagemB[i][j][2] = Math.min(255, imagem[i][j][2] + 40);  
        }  
    }  
}
```

```
    }
}
...
ImagemDigital.plotarImagenCor(imagemB, "imagemB");
}
```

- 3. c)

- Usei o código da questão anterior, criando um novo vetor para representar a nova imagem e fazendo ajustes no loop, atribuo ao novo vetor todos os valores encontrados na imagem original em suas mesmas posições, porém somando 40 ao valor do índice de todos os canais. Eu esperava que a imagem ficasse mais esbranquiçada, o que de fato aconteceu no final, pois como adicionou mais cor em todos os canais, significa que estou adicionando branco à imagem, ou mais saturação. É possível notar isso bem no fundo, que já era em tons de cinza claro, e agora ficou branco total em muitas regiões.
- Saída:



- Implementação (partes adicionadas ao `main()` e aos `for`):

```
public static void main(String args[]) {  
    ...  
    int[][][] imagemC = new int[825][824][3];  
  
    for (int i = 0; i < imagem.length; i++) {  
        for (int j = 0; j < imagem[i].length; j++) {  
            ...  
            imagemC[i][j][0] = Math.min(255, imagem[i][j][0] + 40);  
            imagemC[i][j][1] = Math.min(255, imagem[i][j][1] + 40);  
            imagemC[i][j][2] = Math.min(255, imagem[i][j][2] + 40);  
        }  
    }  
    ...  
    ImagemDigital.plotarImagenCor(imagemC, "imagemC");  
}
```

- 3. d)

- Usei o código da questão anterior, criando um novo vetor para representar a nova imagem e fazendo ajustes no loop, atribuo ao novo vetor todos os valores encontrados na imagem original em suas mesmas posições, porém somando -40 ao valor do índice de todos os canais. Eu esperava que a imagem ficasse mais escura, o que de fato aconteceu no final, pois como estou removendo cor em todos os canais, significa que estou removendo branco da imagem, ou seja, adiciona mais ruído, menos saturação. É possível notar isso muito bem no cabelo e na roupa que já eram escuros ficarem totalmente pretos.

- Saída:

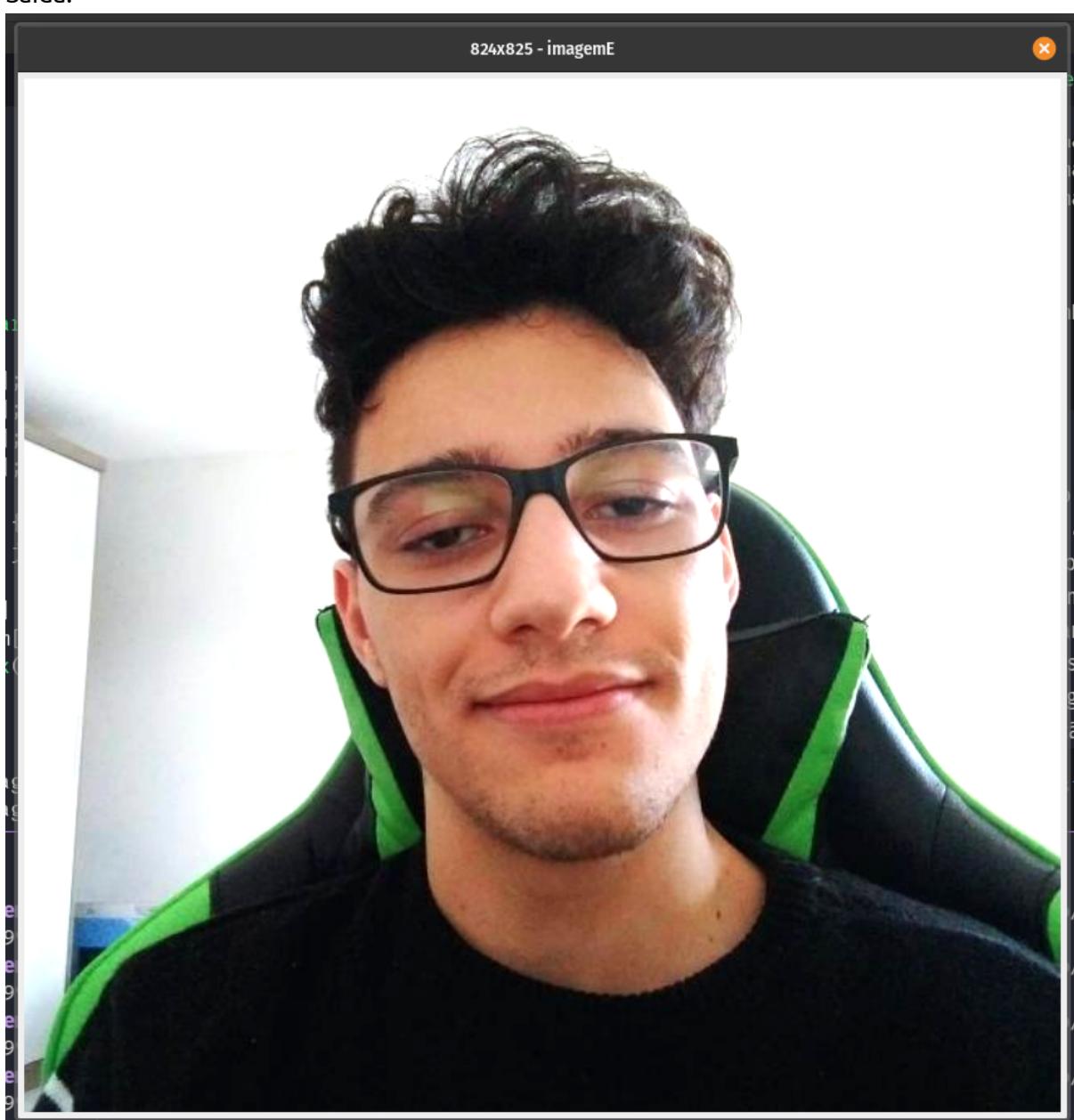


- Implementação (partes adicionadas ao `main()` e aos `for`):

```
public static void main(String args[]) {  
    ...  
    int[][][] imagemD = new int[825][824][3];  
  
    for (int i = 0; i < imagem.length; i++) {  
        for (int j = 0; j < imagem[i].length; j++) {  
            ...  
            imagemD[i][j][0] = Math.max(0, imagem[i][j][0] + (-40));  
            imagemD[i][j][1] = Math.max(0, imagem[i][j][1] + (-40));  
            imagemD[i][j][2] = Math.max(0, imagem[i][j][2] + (-40));  
        }  
    }  
    ...  
    ImagemDigital.plotarImagemCor(imagemD, "imagemD");  
}
```

- 3. e)

- Usei o código da questão anterior, criando um novo vetor para representar a nova imagem e fazendo ajustes no loop, atribuo ao novo vetor todos os valores encontrados na imagem original em suas mesmas posições, porém multiplicando por 1.3, que foi o valor que me fez notar bem a diferença da imagem. O resultado esperado era que a saturação aumentasse muito onde já era alta, e pouco nos demais pontos, e foi o que realmente aconteceu, é possível notar, por exemplo, que na bochecha direita onde a luz reflete a saturação ficou no máximo, porém em outros pontos do rosto onde a luz não reflete tanto que a saturação aumentou pouco.
- Saída:



- Implementação (partes adicionadas ao `main()` e aos `for`):

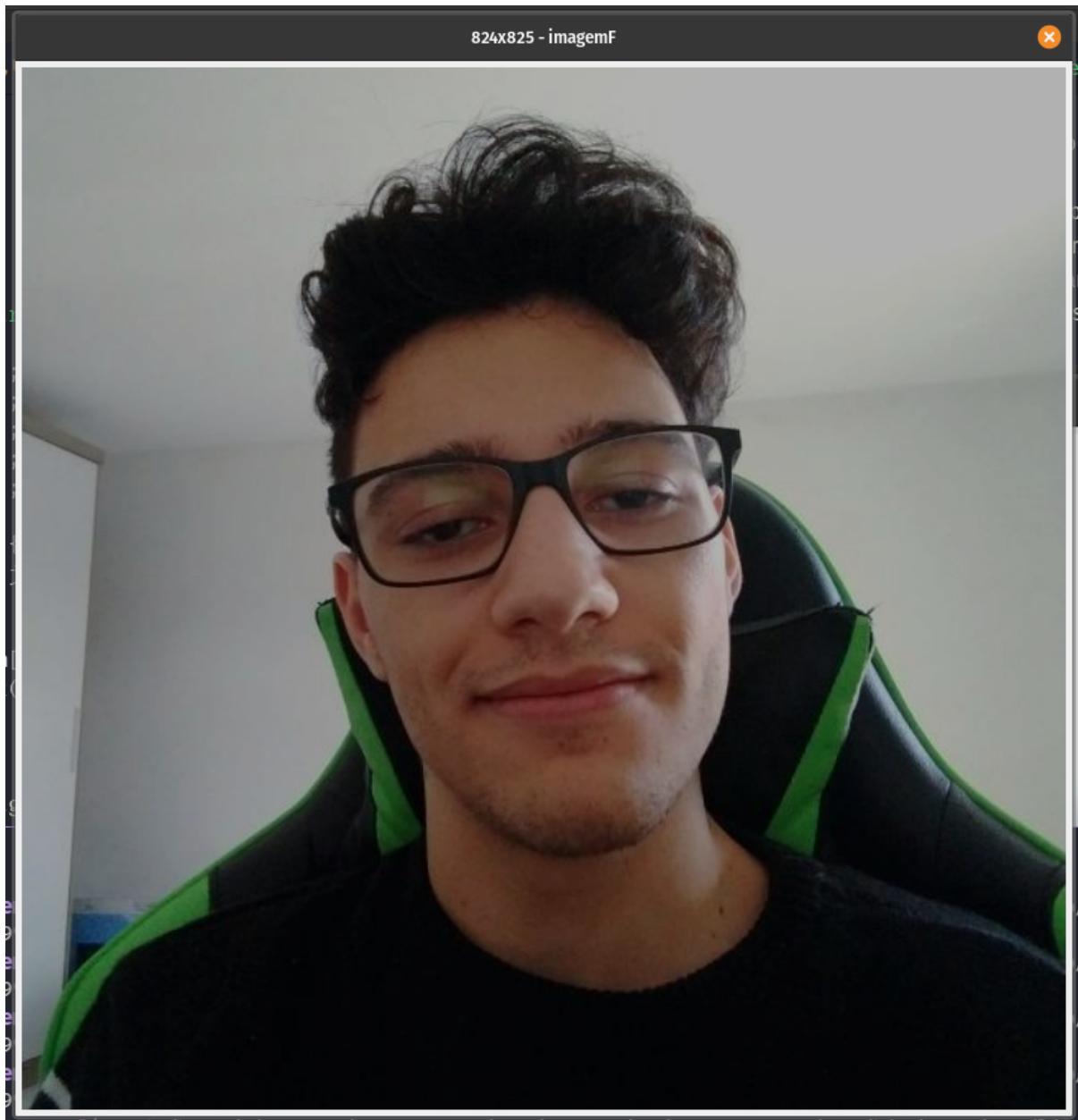
```
public static void main(String args[]) {  
    ...  
    int[][][] imagemE = new int[825][824][3];
```

```
for (int i = 0; i < imagem.length; i++) {  
    for (int j = 0; j < imagem[i].length; j++) {  
        ...  
        imagemE[i][j][0] = Math.min(255, (int) (imagem[i][j][0] *  
1.3));  
        imagemE[i][j][1] = Math.min(255, (int) (imagem[i][j][1] *  
1.3));  
        imagemE[i][j][2] = Math.min(255, (int) (imagem[i][j][2] *  
1.3));  
    }  
    ...  
    ImagemDigital.plotarImagenCor(imagemE, "imagemE");  
}  
}
```

- 3. f)

- Usei o código da questão anterior, criando um novo vetor para representar a nova imagem e fazendo ajustes no loop, atribuo ao novo vetor todos os valores encontrados na imagem original em suas mesmas posições, porém multiplicando por 0.7, que foi o valor que me fez notar bem a diferença da imagem. O resultado esperado era que a saturação diminuisse mais significativamente nos mesmo pontos que na letra anterior aumentou e foi o que realmente aconteceu, é possível notar, por exemplo, que na bochecha direita onde a luz reflete a saturação ficou diminuída e que a deixou cinza, porém em outros pontos do rosto onde a luz não reflete tanto que a saturação diminuiu pouco.

- Saída:



- Implementação (partes adicionadas ao `main()` e aos `for`):

```
public static void main(String args[]) {  
    ...  
    int[][][] imagemF = new int[825][824][3];  
  
    for (int i = 0; i < imagem.length; i++) {  
        for (int j = 0; j < imagem[i].length; j++) {  
            ...  
            imagemF[i][j][0] = Math.min(255, (int) (imagem[i][j][0] *  
0.7));  
            imagemF[i][j][1] = Math.min(255, (int) (imagem[i][j][1] *  
0.7));  
            imagemF[i][j][2] = Math.min(255, (int) (imagem[i][j][2] *  
0.7));  
        }  
    }  
    ...
```

```
    ImagemDigital.plotarImagemCor(imagemF, "imagemF");
```

```
}
```

- 4.

- Nesta questão voltei a usar Python pela facilidade da biblioteca Sklearn, por isso, foi preciso utilizar a biblioteca Pillow para carregar as imagens da base e numpy para converter as imagens em arrays de RGB, ainda foi necessário empilhar cada valor da array de RGB da imagem em uma lista, um dimensão. Cada imagem foi adiciona a X, e sua classe correspondente foi adicionada a y. A partir desse momento, usei as classes StratifiedShuffleSplit e KNeighborsClassifier, a primeira para fazer os 10 splits estratificados, e o segunda para classificar com 1-NN, assim como já usei em atividade passadas.
- Segue as taxas de acertos de cada classe, e a média de acerto de todos os splits:

```
Taxa de acerto (classe 1): 0.93
Taxa de acerto (classe 2): 0.96
Taxa de acerto (classe 3): 0.95
Taxa de acerto (classe 4): 0.935
Taxa de acerto (classe 5): 0.95
Taxa de acerto (classe 6): 0.935
Taxa de acerto (classe 7): 0.915
Taxa de acerto (classe 8): 0.945
Taxa de acerto (classe 9): 0.93
Taxa de acerto (classe 10): 0.895
Média de acertos total: 0.935
```

- Implementação:

```
import numpy as np

from PIL import Image
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.neighbors import KNeighborsClassifier

def load_images(n_class: int, n_sample: int) -> list[list]:
    X = []
    y = []
    for i in range(1, n_class + 1):
        for j in range(1, n_sample + 1):
            image = Image.open(f'./static/orl/class_{str(i).rjust(2, "0")}/sample_{str(j).rjust(2, "0")}.png')
            image_matrix = np.asarray(image)
            image_list = []
            for row in image_matrix:
                for value in row:
                    image_list.append(value)
            X.append(image_list)
            y.append(i)
```

```

        X.append(image_list)
        y.append(i)

    return np.array(X), np.array(y)

n_class = 40
n_sample = 10
X, y = load_images(n_class, n_sample)

sss = StratifiedShuffleSplit(test_size=.5, n_splits=10)
acertos = []
for train_index, test_index in sss.split(X,y):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    knn = KNeighborsClassifier(n_neighbors=1, weights="distance",
metric="euclidean")
    knn.fit(X_train, y_train)

    acertos.append(knn.score(X_test, y_test))

[print(f'Taxa de acerto (classe {i+1}): {acerto}') for i, acerto
in enumerate(acertos)]
print(f'Média de acertos total:
{round(np.average(np.array(acertos)), 3)}')

```

- 5.

- Para fazer essa questão, adaptei o código da anterior, alterando o seguinte: Peguei o array de classes preditas ao invés do score e percorri esse array comparando com a classe real, onde a classe foi diferente da real, usei a função `kneighbors()` do `KNeighborsClassifier` na imagem classificada erroneamente e obtive a imagem mais próxima dela no treino, em seguida, reempilhei as duas imagens e adicionei numa array de 2 indices, fiz isso em cada split e ao finalizar fiz a concatenação de cada par de imagens com todas as imagens do split e obtive 10 imagens diferentes (uma para cada split), onde em cada imagem, a coluna da esquerda representa a imagem do teste que deu errado e a da direita a imagem do treino mais próxima dela. Segue abaixo cada uma das imagens:





■ Split 1:



- Split 2:





- Split 3:









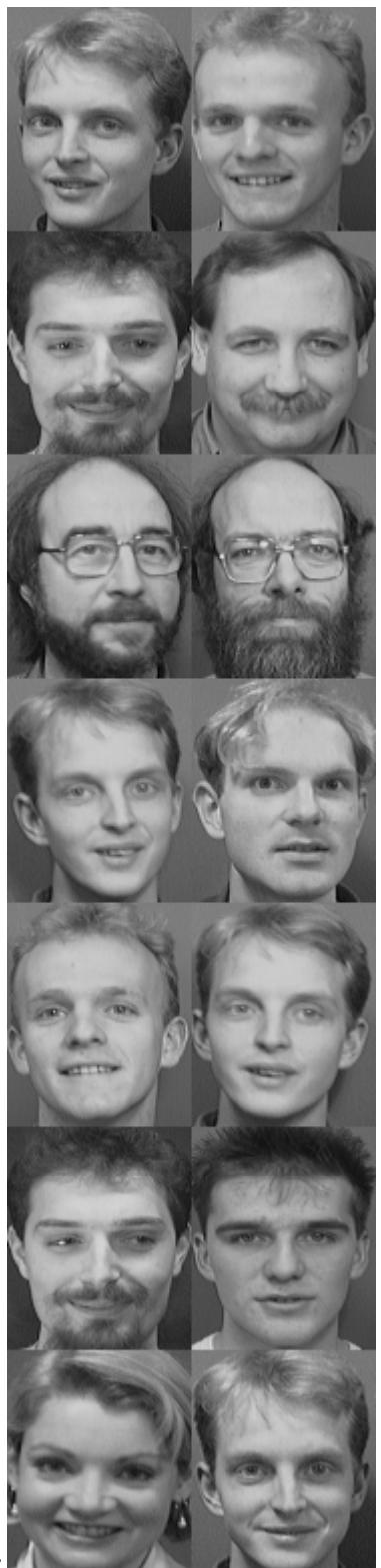
- Split 6:



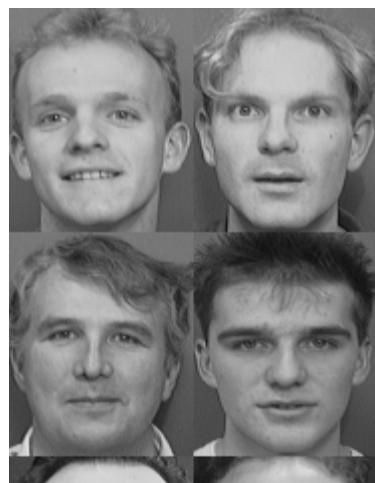
- Split 7:



■ Split 8:



■ Split 9:









■ Split 10:

- Implementação (a função `load_images()` é a mesma da questão 4):

```
n_class = 40
n_sample = 10
X, y = load_images(n_class, n_sample)

sss = StratifiedShuffleSplit(test_size=.5, n_splits=10)
images = []
images_neighbor = []
for train_index, test_index in sss.split(X,y):
    images_split = []
    images_neighbor_split = []
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    knn = KNeighborsClassifier(n_neighbors=1, weights="distance",
metric="euclidean")
    knn.fit(X_train, y_train)

    predicts = knn.predict(X_test)

    for i, predict in enumerate(predicts):
        couple = []
        if predict != y_test[i]:
            neighbor = knn.kneighbors([X_test[i]], n_neighbors=1,
return_distance=False)
```

```
image_matrix = np.array_split(X_test[i], 112)
neighbor_matrix = np.array_split(X_train[neighbor[0][0]], 112)
couple.append(image_matrix)
couple.append(neighbor_matrix)
if couple:
    images_split.append(couple)
images.append(images_split)

for i, images_split in enumerate(images):
    images1 = [np.concatenate(image, axis=1) for image in images_split]
    Image.fromarray(np.concatenate(np.array(images1))).save(f'./static/questao5_split_{i+1}.png')
    print(f'- Split {i+1}: ![Questão 5 split {i+1}] (static/questao5_split_{i+1}.png)')
```