Trabalho 0

MC920 Introdução ao Processamento Digital de Imagens

Rafael Figueiredo Prudencio
r186145@g.unicamp.br

14 de Março de 2018

1 Introdução

Este trabalho visa realizar alguns processamentos básicos em imagens.

2 Tarefas

Tendo um *NumPy* array representando a imagem, um booleano indicando se deve-se salvar ou exibir as saídas do programa e uma string com o nome da imagem, definiu-se quatro funções com funcionalidades distintas que constroem o histograma da imagem, transformam o negativo da imagem, transformam o intervalo de intensidades da imagem para [120, 180] e imprimem alguns dados básicos da imagem de entrada.

2.1 Histograma

Para a construção do histograma usou-se o módulo *pyplot* da biblioteca *Matplotlib*. Usando a função *flatten()* da biblioteca *NumPy*, foi fácil iterar sobre cada pixel da imagem e incrementar 1 ao número de ocorrências da intensidade de pixel correspondente. Obtendo uma lista representativa do eixo x com todas as intensidades de pixel possíveis e uma lista representando o eixo y com a frequência de cada pixel, usou-se o método *plot()* do módulo *pyplot* para obter o gráfico desejado. Na Figura 1 tem-se um exemplo do resultado ao executar a função *histogram()* no código anexado. Caso o usuário tenha escolhido salvar a imagem através do argumento booleano *save*, ela será salva por default no diretório "./out". Caso o diretório não exista, ele será criado. A imagem será salva com seu nome original concatenado à string "_hist"no formato png. Para as demais fuções, o mesmo booleano será usado para indicar se deve-se salvar a imagem no diretório "./out", apenas alterando-se o nome usado para salvá-la. Algumas limitações do programa são que o usuário não pode escolher o formato com que deseja salvar a imagem nem mesmo o nome e local onde deseja salvá-la.

2.2 Estatísticas

Para esta funcionalidade, o único argumento necessário é a matriz de pixels da imagem no formato de um *NumPy* array. Usando apenas as funções disponíveis na biblioteca *NumPy* como *max()*, *min()* e *mean()* e o atributo *shape* do *NumPy* array, foi possível obter todos os dados pedidos. Por default os dados são impressos na saída padrão do terminal. Uma das limitações do programa é que não há a opção do usuário salvar os dados da imagem em um arquivo texto.

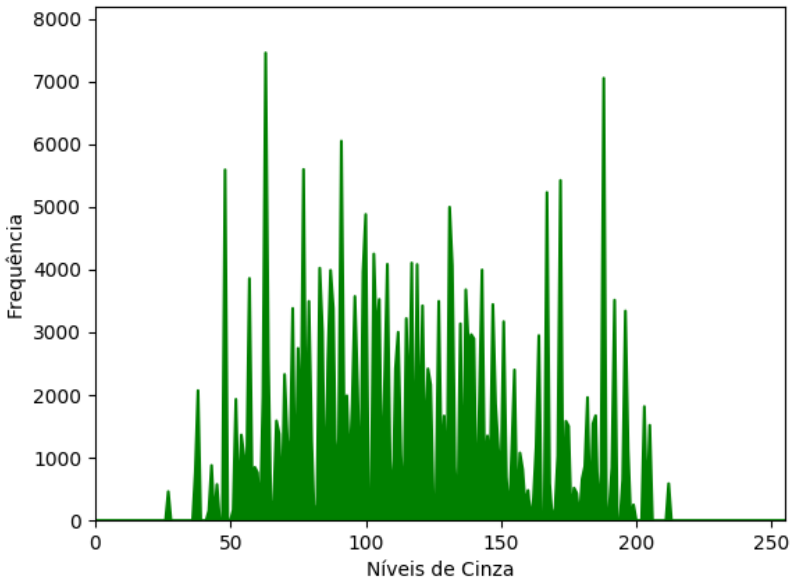

Figura 1: Saída da função *histogram()* do módulo *basic.py* quando a flag "save"está setada.

Na Figura 2 temos o exemplo da saída padrão após rodar o programa para um diretório de imagens png.

2.3 Transformações de Intensidade

Para as transformações de intensidade, definiu-se as funções *invert* e *normalize* que respecitvamente, invertem a intensidade de píxeis da imagem e normalizam o intervalo de intensidades entre um valor mínimo e máximo especificado pelo usuário.

2.3.1 Negativo

A função *invert* admite três argumentos: um NumPy array representando a intensidade dos pixels da imagem, um booleano indicando se deve-se salvar ou exibir a imagem e o nome da imagem. Aplicando a Equação 1, onde I é a matriz de dimensões N $\times$ N representando a imagem original e β é a profundidade de bits da imagem (β = 8 no nosso caso), estamos na prática invertendo as regiões mais intensas da imagem com as menos intensas. A Figura 3b é a imagem invertida da Figura 3a. A imagem é salva no diretório "./out"com o seu nome original concatenado à string "_inv"no formato png. Após salvar ou exibir a imagem, a função retorna um NumPy array representando o negativo da imagem de entrada. Além das mesmas limitações citadas em 2.1, essa função assume que as imagens de entrada têm profundidade de 8 bits e não permite que o usuário especifique outra.

$$|I(x,y) - 2^{\beta} + 1| \qquad x,y \in N \tag{1}$$

```
rafael:~/Documents/unicamp/mcxxx/920/t1
$ python basic.py ./data png True
Para a imagem './data/city.png' temos:
Propriedades:
  largura: 512
  altura: 512
  intensidade mínima: 16
  intensidade máxima: 235
  intensidade média: 112.20

Para a imagem './data/butterfly.png' temos:
Propriedades:
  largura: 512
  altura: 512
  intensidade mínima: 27
  intensidade máxima: 212
  intensidade média: 116.77
```

Figura 2: Screenshot do terminal após execução do programa em um diretório com imagens png.

(a) Original *(b)* Negativo *(c)* Intensidade

Figura 3: Resultado da execução do programa com a flag "save"setada. (esquerda) Imagem original obtida pelo site da disciplina. (centro) Transformação da imagem em seu negativo. (direita) Transformação da intensidade da imagem para o intervalo [120, 180]

2.3.2 Intervalo de Intensidade

A função *normalize* é responsável por realizar a transformação do intervalo de intensidades e admite cinco parâmetros: um NumPy array, um valor mínimo de intensidade, um valor máximo de intensdidade, um booleano para salvar ou exibir a imagem e o nome da mesma. Aplicando a Equação 2, onde I é a matriz de dimensões N×N representando a imagem original, α é o valor máximo de intensidade, γ é o valor mínimo de intensidade e β é a profundidade de bits da imagem, é possível transformar o intervalo de intensidades da imagem. Para este trabalho, usamos α = 180, γ = 120 e β = 8. A Figura3c é a tranformação da imagem na Figura 3a para o intervalo de intensidades entre 120 e 180. Uma das limitações do programa é que esses parâmetros não podem ser customizados pelo usuário na execução. O programa salva a imagem no formato png no diretório "./out"com o nome concatenado à string "_norm".

$$I(x,y)\frac{\alpha-\gamma}{2^{\beta}-1}+\gamma \qquad x,y \in N \tag{2}$$

3 Execução

O programa denominado *basic.py* pode ser executado tanto pelo terminal como script, como pode ser importado e suas funções podem ser chamadas individualmente. A execução pelo terminal consiste na chamada da função *main* e suporta várias paramterizações. O usuário pode especificar até três parâmetros, onde os omitidos assumem os valores default indicados na documentação do código. O primeiro parâmetro é o *image_dir* que pode referir-se tanto a um diretório como um único arquivo de imagem. Por default, o programa procurará imagens no diretório "./data". O segundo parâmetro é o *file_type* que deve ser algum formato válido de imagem suportado pela função *scipy.imread()*. Caso *image_dir* seja um arquivo, este argumento será ignorado e o tipo de imagem do próprio arquivo será considerado. Por default, o código procura no diretório imagens no formato png. O último parâmetro é um booleano *save* que indica se o programa deve salvar ou exibir as imagens. Por default, *save* é falso e o programa apenas exibe as imagens usando funções como *matplotlib.pyplot.show()* e *scipy.imshow()*. Note que se o programa for executado com *save* valendo *False*, o usuário deve fechar as janelas que são abertas para continuar a execução do programa.

O programa foi rodado com os pacotes NumPy(v1.14.1), Matplotlib(v2.1.1), Tensorflow(v1.5.0) e Scipy(v1.0.0) usando Python 3.5.2. Não é garantido o funcionamento do programa em qualquer outro ambiente com diferentes versões além das citadas acima.

4 Conclusão

Através deste trabalho, foi possível familirizar-se com alguns pacotes de python como NumPy, Matplotlib e o Scipy, que devem ser úteis para o resto da disciplina. Também exercitou-se conceitos básicos de processamento de imagens como transformações, obtenção de estatísticas e construção do histogerama da imagem.

Referências

NumPy v1.14 Manual (2018). URL: `https://docs.scipy.org/doc/numpy/` (acedido em 03/11/2018).

Python 3.5.2 (2016). URL: `https://www.python.org/downloads/release/python-352/` (acedido em 03/11/2018).

SciPy 1.0.0 Release Notes (2017). URL: `https://docs.scipy.org/doc/scipy/reference/release.1.0.0.html` (acedido em 03/11/2018).

Tensorflow API r1.5 (2018). URL: `https://www.tensorflow.org/versions/r1.5/api_docs/` (acedido em 03/11/2018).

The Pyplot API (2017). URL: `https://matplotlib.org/2.1.1/api/pyplot_summary.html` (acedido em 03/10/2018).