UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
http://www.icmc.usp.br/

Professor: Rodrigo Fernandes de Mello (mello@icmc.usp.br)
Alunos PAE: Lucas Pagliosa (lucas.pagliosa@usp.br)
Felipe Duarte (fgduarte@icmc.usp.br)
Monitores: Victor Forbes (victor.forbes@usp.br)

Trabalho 04: Convolução de Imagens e Classificação Usando K Nearest Neighbors

1 Prazos e Especificações

O trabalho descrito a seguir é individual e não será tolerado qualquer tipo de plágio ou cópia em partes ou totalidade do código. Caso seja detectada alguma irregularidade, os envolvidos serão chamados para conversar com o professor responsável pela disciplina.

A entrega deverá ser feita única e exclusivamente por meio do sistema run.codes no endereço eletrônico `https://run.codes` até o dia **28 de outubro de 2016 às 23 horas e 59 minutos**. Sejam responsáveis com o prazo final para entrega, o run.codes está programado para não aceitar submissões após este horário e não serão aceitas entregas fora do sistema.

Leia a descrição do trabalho com atenção e várias vezes, anotando os pontos principais e as possíveis formas de resolver o problema. Comece a trabalhar o quanto antes para que você não fique com dúvidas e que consiga entregar o trabalho a tempo.

2 Descrição do Problema

Considere um conjunto T de imagens, para as quais conhecemos suas respectivas classes. Por exemplo, para um problema de identificação de faces de pessoas, considere que tenhamos o conjunto (no contexto deste trabalho definido como conjunto de treinamento) $T = \{i_1, i_2, \ldots, i_n\}$ e que os respectivos indivíduos são $C = \{c_1, c_2, \ldots, c_n\}$ (no contexto deste trabalho definido como o conjunto de classes associadas à cada elemento do conjunto de treinamento).

Considere, ainda, que duas características foram extraídas de cada uma das imagens em T, sendo elas: i) o número de pixels amarelos e ii) o número de pixels pretos. Suponha (com muito exagero) que essas duas características são suficientes para identificar indivíduos (apenas para compreensão do problema). Sejam essas características plotadas na figura a seguir, cuja cor identifica ou o personagem Homer, em amarelo, ou o Gru, em preto (Meu Malvado Favorito, qualquer coincidência é mero acaso) – cada ponto na figura abaixo identifica uma das imagens em função de suas características:

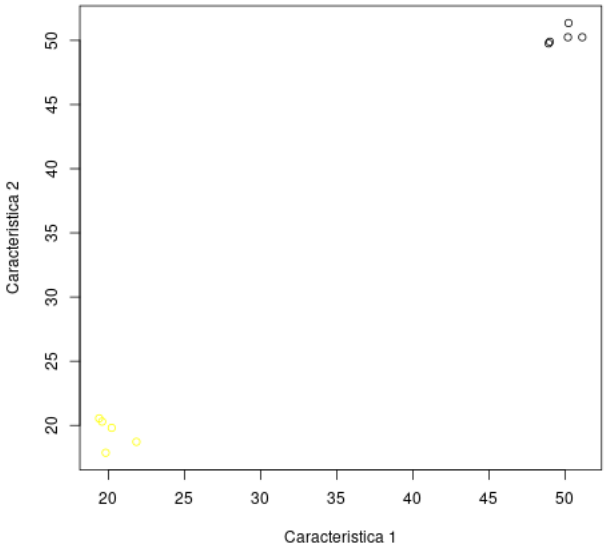

Figura 1: Características das imagens do personagem Homer (pontos amarelos) e do personagem Gru (pontos pretos).

Agora considere que você recebeu um conjunto $E = \{e1, e2\}$ (denominado conjunto de teste) que contém imagens de teste. Considere que as mesmas características foram extraídas para as imagens do conjunto E, as quais também foram plotadas:

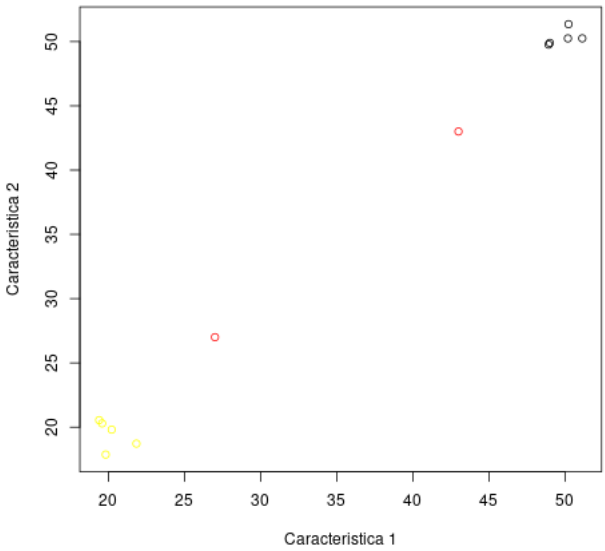

Figura 2: Os pontos em vermelho representam duas novas imagens do conjunto de teste. Para $K = 1$, a imagem representada pelo ponto vermelho da direita deve ser classificada como a do personagem Gru, enquanto a outra imagem como Homer.

Há uma técnica (muito simples) de Aprendizado de Máquina que classifica um determinado indivíduo do conjunto de teste, com base em seus K vizinhos mais próximos. Para $K = 1$, observe que e_1 será classificado como Homer (ponto em vermelho mais próximo dos pontos amarelos), e e_2 como Gru (ponto vermelho mais próximo dos pontos pretos). No entanto, diferentes valores de K podem ser definidos para seu algoritmo.

Observe que a medida de distância Euclidiana deve ser empregada para verificar os K vizi-

nhos mais próximos de um exemplo de consulta (esse é o nome que damos tanto para e_1 quanto para e_2, quando desejamos identificar seus respectivos indivíduos):

$$\text{Euclidiana}(\mathbf{e_i}, \mathbf{i_j}) = \sqrt{\sum_{p=1}^{\text{qtde características}} (e_i[p] - i_j[p])^2} \quad (1)$$

Vamos agora generalizar. Considere que o conjunto de treinamento T contém imagens de objetos (procurem mais na Internet sobre a base de imagens COIL e a CIFAR para que compreendam) tais como carros, bolas, prédios, guitarras flying V, guitarras old school, guitarras bb king-style (Lucile), guitarras Les Paul, guitarras telecaster (perai, vou parar com as guitarras), bem mais uma, guitarras SG (estas são demais!!! lembro-me do Tony Iommi tocando...). Considere que o conjunto C contém as respectivas classes desses objetos, por exemplo, $C = \{\text{carro}, \text{bola}, \text{prédio}, \text{guitarra}, \text{guitarra}, \text{guitarra}, \text{guitarra}, \text{guitarra}, \text{guitarra}\}$ (hehe).

Agora, considere a técnica de convolução de imagens por correlação descrita na Seção 4, a qual recebe uma imagem x (tanto do conjunto T quanto do conjunto E) qualquer de entrada e uma matriz denominada máscara de convolução para produzir uma de imagem de resultado. Esta técnica de convolução será, em nosso caso, utilizada para a extração de características das imagens tanto do conjunto de treinamento quanto do de teste. Logo, o exemplo acima com o Homer e o Gru foi apenas para explicar o algoritmo de K vizinhos mais próximos em um cenário super simples.

Considere que seu programa deve receber M matrizes quadradas de convolução com tamanho $m \times m$ (em que m sempre será um número ímpar) e que ele aplica cada uma desses matrizes sobre cada imagem x, obtendo um conjunto de imagens resultantes. Em seguida, e na ordem dessas imagens resultantes, você deve organizar um vetor de características que será utilizado na técnica de K vizinhos mais próximos, conforme descrito na Seção 5.

Em resumo, as principais tarefas de seu programa são:

1. Receber um arquivo de entrada com uma lista dos nomes das imagens no formato PGM (descrição no formato na Seção 3) correspondente ao conjunto de treinamento;
2. Receber um arquivo de entrada com cada uma das classes das imagens do conjunto de treinamento;
3. Receber um arquivo de entrada com uma lista dos nomes das imagens PGM correspondente ao conjunto de teste;
4. Receber o número de máscaras de convolução;
5. Receber o parâmetro m que define o número de linhas e colunas de cada máscara de convolução (lembre-se que as máscaras serão definidas somente por matrizes quadradas);
6. Receber os valores (células) de cada uma das matrizes de convolução;
7. Receber o valor de K para o algoritmo de K vizinhos mais próximos;
8. Aplicar as convoluções sobre todas as imagens do conjunto de treinamento, gerando os vetores de características (um vetor por imagem);
9. Aplicar as convoluções sobre todas as imagens do conjunto de teste, gerando os vetores de características (um vetor por imagem);
10. Rodar K vizinhos mais próximos e classificar as imagens de teste (em caso de empate, você pode escolher qualquer classe como resultado. Tentaremos garantir que isso não ocorra nos casos de teste).

3 Formato de Imagens PGM

Um arquivo de imagem PGM é composto pela seguinte estrutura:

- formato do arquivo: P2 informa que a matriz está escrita em ASCII. P5 a matriz está na forma binária (nossas matrizes serão sempre em formato P5);
- dimensões da imagem: número de colunas e número de linhas, nesta ordem;
- máximo valor da matriz. Esta informação pode ser útil a priori para normalização dos valores da matriz, por exemplo;
- matriz binária ou ASCII.

Abaixo, um exemplo de imagem ASCII no formato PGM (impossível de exemplificar o formato P5 no texto, devido aos binários):

```
P2
5 5
1
1 1 1 1 1
1 0 0 0 1
1 0 1 0 1
1 0 0 0 1
1 1 1 1 1
```

4 Convolução de Imagens

Uma máscara é uma matriz geralmente quadrada e de tamanho ímpar que quando "aplicada" à imagem original, a modifica de forma a realçar certas características importantes para a aplicação. Uma das formas mais comuns de uma máscara ser "aplicada" sobre a imagem é através de uma operação de convolução por correlação (ou simplesmente correlação de agora em diante) conforme ilustrado na Figura 3. Dada uma máscara $\mathbf{M}_{m\times m}$ (em verde), uma convolução sobre um pixel $p_{i,j}$ é a soma das multiplicações de cada elemento da máscara sobre os pixels vizinhos de $p_{i,j}$, dado pelo conjunto $\mathcal{V}_{i,j} = p_{x,y}, \forall x = [i - m/2, i + m/2], y = [j - m/2, j + m/2]$ (em azul). Para os pixels vizinhos que extrapolam os limites da imagem, assume-se valores iguais a zero. Logo, consideramos que outra imagem resultante (em vermelho) é produzida após realizarmos uma convolução sobre todos os pixels da imagem de entrada.

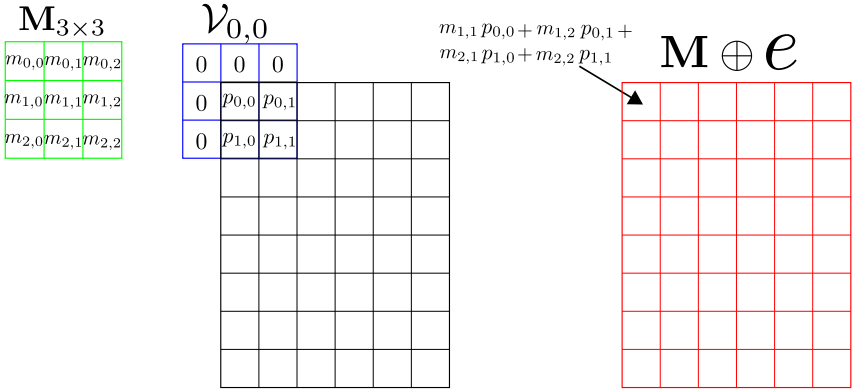

Figura 3: Processo de convolução por correlação de uma imagem.

Para exemplificar, tome as matrizes de convolução empregadas por Canny para identificar bordas de uma imagem. Canny aplica duas máscaras de convolução definidas por Sobel, $\mathbf{S}_x$ e $\mathbf{S}_y$,

as quais permitem computar diferenças finitas de primeira ordem em relação aos eixos vertical (x) e horizontal (y) das imagens, respectivamente:

$$\mathbf{S}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \ \mathbf{S}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}.$$

Estas máscaras, quando aplicadas individualmente, servem para realçar segmentos de retas verticais e horizontais na imagem, respectivamente. Quando usadas em conjunto em um pós-processamento do algoritmo de Canny, estas matrizes auxiliam no realce de bordas da imagem, conforme ilustrado na Figura 4.

Figura 4: Imagem original (esquerda) e após Canny (direita), que envolve a convolução com as máscaras $\mathbf{S}_x$ e $\mathbf{S}_y$.

Neste trabalho, as máscaras são do tipo `double` tal que a soma dos seus elementos sempre é igual a 1.

5 Vetor de características

Após a convolução obtemos outra representação da imagem original, denominada imagem resultante no contexto deste trabalho. O próximo passo consiste em definirmos um vetor de características (do inglês *feature vector*) a partir da imagem resultante.

Neste trabalho utilizaremos um vetor de características simples: para cada linha da imagem resultante, crie um vetor com seis dimensões formado pela:

1. Quantidade de elementos maiores que zero;
2. Quantidade de elementos iguais a zero;
3. Quantidade de elementos menores que zero;
4. Média dos pixels da linha (como `double`);
5. Variância dos pixels da linha (como `double`);
6. Entropia dos pixels da linha na forma:

$$H(\text{linha}_i) = -\sum_j \text{pixel}_{i,j} * \log_2(|\text{pixel}_{i,j}| + 1) \tag{2}$$

,

nesta exata ordem. Concatene todos estes vetores, um em frente ao outro, na ordem das linhas processadas da imagem resultante para formar o *feature vector*, conforme ilustrado na Figura 5. Caso tenha interesse em aprender mais sobre a montagem de vetores de características mais completos, procure por *descritor de textura de Haralick*.

$$\left.\begin{array}{l}\text{media}_1 = \frac{0+1+0-1}{4} = 0 \\ \text{variância}_1 = \frac{(0-0)^2+(1-0)^2+(0-0)^2+(-1-0)^2}{4} = 0.5 \\ \text{entropia}_1 = <[0,-1,0,1],[0.0,\ 1.0,\ 0.0,\ 1.0]> = 0.0\end{array}\right\} l1$$

$$e = \left.\begin{array}{|cccc|l} \hline 0 & 1 & 0 & -1 & l1 = [1, 2, 0, 0.0, 0.5, 0.0] \\ 0 & 2 & 0 & 0 & l2 = [1, 3, 0, 0.5, 0.75, -3.16] \\ 3 & 4 & 5 & -2 & l3 = [3, 0, 1, 2.5, 7.25, -25.042] \\ \hline \end{array}\right\} \mathbf{v}_e = [l1, l2, l3]$$

Figura 5: Vetor de características de uma imagem.

6 Arquivos de Entrada e Saída

O arquivo de entrada conterá, em cada linha e nesta ordem:

- Arquivo de entrada com os nomes das imagens do conjunto de treinamento T;
- Arquivo de entrada com as classes C associadas a cada uma das imagens do conjunto de treinamento T. Classes serão definidas na forma de números double;
- Arquivo de entrada com os nomes das imagens do conjunto de teste E;
- O número M de máscaras de convolução;
- A dimensão de cada máscara ($m \times m$);
- Os valores double de cada uma das máscaras (valores da primeira linha da primeira matriz, depois os valores da segunda linha da primeira matriz, até a última linha da primeira matriz e então inicia-se a segunda matriz de convolução até a última);
- O número k de vizinhos mais próximos.

A Saída deve ser a classe de cada imagem de teste $E = \{e_1, e_2, \ldots, e_q\}$ impressa utilizando o seguinte formato (apenas para ilustrar):

```
printf("%.3lf\n", class[i]);
```

7 Observações importantes

- Todas as imagens que serão lidas pelo seu programa possuem o mesmo número de linhas e colunas.
- Programe as impressões na tela EXATAMENTE como exemplificado no decorrer deste documento. Tome cuidado com pulos de linha, tabs, espaços, etc.
- Coloque dentro do zip todos os arquivos de código (`*.h *.c`), o makefile e um arquivo texto com o nome e número USP.