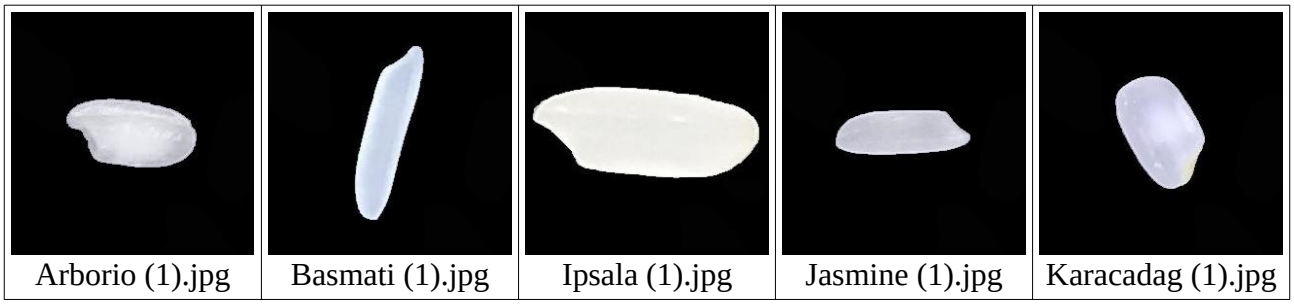
PSI-3471: Fundamentos de Sistemas Eletrônicos Inteligentes
Primeiro semestre de 2023 Exercício-programa Prof. Hae
Data de entrega: 09/07/2023 até 23:59 horas
Data máximo de limite de entrega com atraso e desconto: 11/07/2023 até 23:59 horas

Obs. 1: Não posso aceitar entrega após data-limite, pois preciso fechar as notas.
Obs. 2: Este EP deve ser resolvido individualmente.

O objetivo deste exercício é classificar os 5 tipos de grãos de arroz (figura 1), do conjunto de dados:
https://www.muratkoklu.com/datasets/
e verificar se alinhar os grãos horizontalmente ajuda (ou não) nessa tarefa.



Arborio (1).jpg	Basmati (1).jpg	Ipsala (1).jpg	Jasmine (1).jpg	Karacadag (1).jpg

Figura 1: Os 5 tipos de grãos de arroz do conjunto de dados.

Peguei esse conjunto (que é muito grande na versão original) e armazenei somente as primeiras 50 imagens de cada tipo de arroz, gerando o subconjunto com 250 imagens abaixo:
www.lps.usp.br/hae/psi3471/ep1-2023/rice_1to50.zip

Parte 1 (4 pontos): Alinhar os grãos horizontalmente - "alinha.cpp"

Esta parte pode ser feita em C++ (de preferência) ou em Python.

1) Ache o centro de massa e a orientação da imagem original convertida numa imagem binária. Para facilitar debugar o seu programa, pode imprimir a imagem intermediária como a figura 2b onde pode se visualizar o centro e a orientação do grão de arroz. Estou passando a função *findCenterAndOrientation* que acha o centro de massa e a orientação de uma imagem binária (uma imagem uint8 que só tem pixels com valores 0 ou 255). Para maiores detalhes, veja a discussão em:
https://stackoverflow.com/questions/14720722/binary-image-orientation

```
#include <cekeikon.h>
(...)
Point3d findCenterAndOrientation(Mat_<GRY> src) {
  Moments m = moments(src, true);
  double cen_x = m.m10/m.m00;
  double cen_y = m.m01/m.m00;
  double theta = 0.5 * atan2( 2*m.mu11, m.mu20-m.mu02);
  return Point3d(cen_x, cen_y, theta);
}
```

Um exemplo de chamada desta função, para uma imagem binária *a*, é:

```
Point3d p=findCenterAndOrientation(a);
printf("%f %f %f\n",p.x,p.y,p.z);
```

Após a chamada, *p.x* e *p.y* conterão o centro de massa da imagem *a* e *p.z* representará o ângulo em radianos da inclinação da figura.

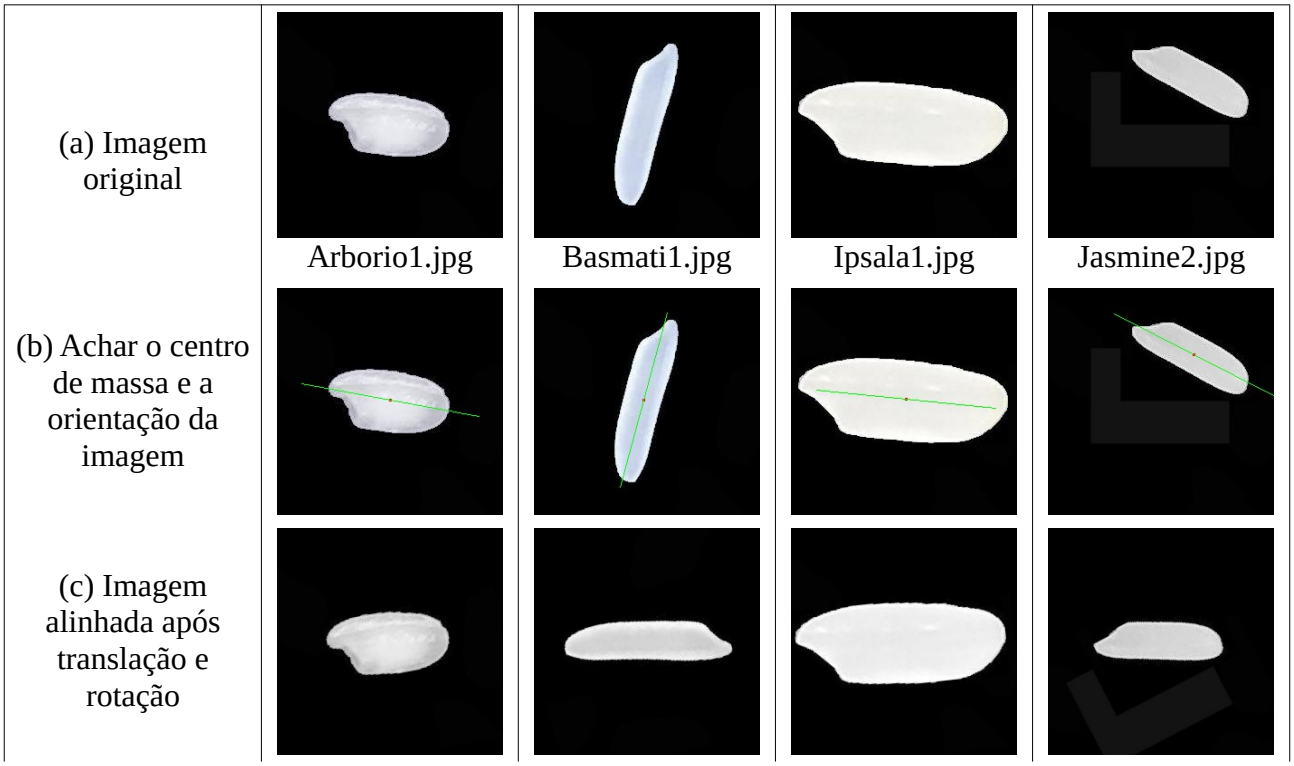


Figura 2: Achar o centro de massa e a orientação da imagem e transformá-la para que fique centrada e horizontal. Estas imagens estão no subdiretório *orientacao*.

2) Rotacione e centralize a imagem, para que a imagem esteja centrada e a orientação dos grãos seja horizontal, como mostra a figura 2c. Teste obrigatoriamente o seu programa para a imagem *Jasmine2.jpg* (do diretório *orientacao*), para se certificar de que o seu programa está colocando corretamente o centro de massa no centro da imagem. Coloque o resultado de *Jasmine2.jpg* tanto no relatório como no vídeo. Você deve calcular a orientação usando imagem binária, mas a imagem alinhada deve ser colorida (do mesmo tipo que a imagem original).

Para cada imagem, você deve fazer reamostragem (*warpAffine* ou *warpPerspective*) uma única vez, para não introduzir distorções desnecessariamente. Para isso, calcule a matriz afim ou perspectiva que efetua a transformação desejada e efetue a reamostragem uma única vez.

3) Depois de verificar que o seu programa funciona para algumas imagens, escreva o programa alinha.cpp (ou alinha.py) que permite alinhar várias imagens de uma vez. Vamos gravar as imagens alinhadas no mesmo diretório das imagens originais, só mudando a extensão ".jpg" para extensão ".png". A extensão ".png" irá denotar as imagens alinhadas, distinguindo-as das imagens originais com o mesmo nome e extensão ".jpg". O programa deve funcionar da seguinte maneira:

O seguinte comando alinha *imagem.jpg* criando *imagem.png*:

```
$ alinha imagem.jpg
```

O seguinte comando alinha *imagem1.jpg* e *imagem2.jpg*, gerando as imagens alinhadas *imagem1.png* e *imagem2.png*:

```
$ alinha imagem1.jpg imagem2.jpg
```

O seguinte comando alinha todas as imagens *.jpg* do diretório atual:

```
$ alinha *.jpg
```

Acredito que a forma mais fácil de fazer isso é usar “wildcard expansion”. Isto funciona em C++/Linux e Python/Linux. Também funciona em C++/Windows usando compilador GCC que vem dentro do Cekeikon. Porém, não funciona se você usar Python/Windows ou um compilador de C++ diferente de GCC em Windows. Neste caso, você precisa achar alguma forma alternativa de ler todos os arquivos *.jpg* de um diretório. Veja Anexo A para mais informações.

Parte 2 (6 pontos): Classificar as imagens não-alinhadas e alinhadas

Pode ser feita em Python (de preferência) ou C++.

O objetivo da parte 2 do exercício é classificar os 5 tipos de arroz usando algum método de aprendizado de máquina, verificando se alinhar os grãos horizontalmente ajuda (ou não) nessa tarefa.

Para obter resultado estatisticamente significativo, vamos executar “5-fold cross validation”: vamos treinar o modelo 5 vezes e classificar 5 diferentes conjuntos de teste, primeiro usando as imagens não-alinhadas e depois as alinhadas. K-fold cross validation é muito usada quando a quantidade de dados é pequena.

Vamos dividir as imagens em 5 subconjuntos: S1={imagens de 1 a 10}, S2={imagens de 11 a 20}, S3={imagens de 21 a 30}, S4={imagens de 31 a 40} e S5={imagens de 41 a 50}. Em cada fold, vamos escolher um dos subconjuntos como conjunto de teste, treinar o modelo com os 4 subconjuntos restantes e fazer predição no conjunto de teste escolhido. Com isso, obteremos 5 taxas de acerto para imagens não-alinhadas e 5 taxas de acerto para imagens alinhadas. Esta quantidade de dados permitirá afirmar (ou não) se alinhar melhora taxa de acerto.

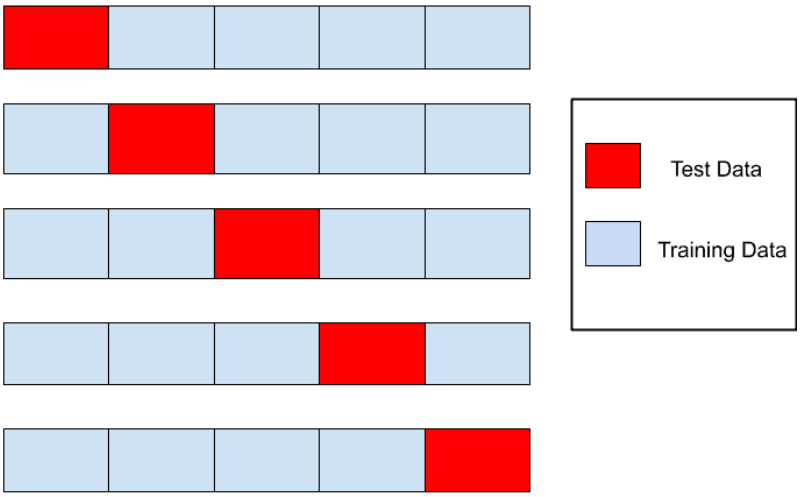


Figura 3: 5-fold cross validation.

2.1) class_orig.py – classificar imagens JPG originais não-alinhadas
a) Treine e classifique usando as imagens JPG originais não-alinhadas usando algum método de aprendizado de máquina (que pode ser rede neural convolucional) e 5-fold cross validation para obter as 5 taxas de acertos. Se achar necessário, pode redimensionar as imagens na entrada.
b) Procure obter a maior taxa de acerto possível. A nota do exercício dependerá da taxa de acerto.

2.2) class_alin.py – classificar imagens PNG alinhadas
a) Repita o procedimento acima usando as imagens PNG alinhadas e obtenha as 5 taxas de acerto de 5-fold cross validation.
b) Novamente, procure obter a maior taxa de acerto possível. A nota do exercício dependerá da taxa de acerto.

2.3) Análise dos resultados
a) Anote as 5 taxas de acertos usando imagens JPG não-alinhadas e outras 5 taxas de acertos usando as imagens PNG alinhadas no relatório e no vídeo.
b) Calcule as 2 médias e os 2 desvios-padrões dos algoritmos para imagens não-alinhadas e alinhadas.
c) Alinhar as imagens melhorou a acuracidade? Registre a sua conclusão no relatório e no vídeo.

Obs. 1: Este exercício deve ser resolvido individualmente.

Obs. 2: Entregue os programas-fontes (alinha.cpp, class_orig.py, class_alin.py). Você pode enviar links para Google Colab em vez de programas .py. Não envie arquivo .ipynb.

Obs. 3: Entregue um documento PDF de no máximo 4 páginas (relatorio.pdf) descrevendo o funcionamento dos seus programas, os resultados obtidos e as suas conclusões. O envio do relatório é obrigatório (veja o anexo B). O relatório é um documento Word/Latex/Writer convertido para PDF. Um notebook Python anotado não será aceito como relatório.

Obs. 4: Entregue um vídeo de no máximo dois minutos explicando o funcionamento dos seus programas, os resultados obtidos e as suas conclusões. No vídeo deve aparecer em algum momento o seu rosto e um documento seu. É necessário que o vídeo contenha áudio, pois é muito difícil entender um vídeo sem a explicação falada. O vídeo não pode ter sido acelerado para diminuir a duração, pois dificulta entender a fala. Vocês podem enviar o vídeo em si (.mkv, .mp4, .avi, etc.) ou um link para o vídeo (youtube, google drive, etc). No segundo caso, deve assegurar que todos os professores (Hae, Renato e Wesley - hae.kim@usp.br, renatocan@gmail.com, wesleybeccaro@usp.br) tenham acesso ao seu vídeo. Não se esqueçam de escrever o seu nome em 2 lugares diferentes: no início do vídeo e no início dos programas-fontes.

Obs. 5: Não envie o banco de dados, isto é, não envie o arquivo “rice_1to50.zip” ou as imagens descompactadas. Se você quiser explicar sobre alguma imagem, coloque-a no relatório e mostre-a no vídeo.

Obs. 6: Envie os arquivos e/ou links em edisciplinas.

Anexo A: Argumentos e expansão de wildcard

A expansão de wildcard pode tornar mais simples executar um programa para vários arquivos de entrada, por exemplo, executar o programa *alinha* para os 50 arquivos ".jpg" do diretório, sem ter que chamar o programa *alinha* 50 vezes.

Um programa escrito em C++ ou Python, tanto em Linux como em Windows aceita argumentos. Os programas abaixo, em C++ e Python, imprimem o número de argumentos e cada um dos argumentos recebidos.

```cpp
//arg.cpp
#include <cstdio>
using namespace std;
int main(int argc, char** argv) {
  printf("Argc=%d\n",argc);
  for (int i=0; i<argc; i++)
    printf("%s\n",argv[i]);
}
```

```python
#arg.py
import sys
print("N arg=",len(sys.argv))
for i in range(len(sys.argv)):
  print(sys.argv[i])
```

Se executar esses programas, eles imprimem o número de argumentos seguido por cada um dos argumentos. Por exemplo (o argumento índice zero é o nome do programa):

```
Linux$ ./arg x y
Argc=3
./arg
x
y

Linux$ python3 arg.py x y
N arg= 3
arg.py
x
y
```

Agora, Linux possui um mecanismo chamado "wildcard expansion" que substitui argumentos com símbolo * ou ? pelos nomes dos arquivos que "casam" com o padrão. O símbolo * casa com um número arbitrário de caracteres e o símbolo ? casa com um único caractere. Digamos que seu diretório contenha os seguintes arquivos:

```
diretorio$ ls
Arborio1.jpg  Arborio2.jpg  arg  arg.cpp  arg.py
```

Se você executar nesse diretório:

```
Linux$ ./arg *.jpg
```

Linux irá expandir “*.jpg” para “Arborio1.jpg Arborio2.jpg”. Seria o mesmo que tivesse escrito: `“$ ./arg Arborio1.jpg Arborio2.jpg”`. O programa irá imprimir:

```
Argc=3
./arg
Arborio1.jpg
Arborio2.jpg
```

Este mecanismo pode ajudar executar o programa *alinha* para as 50 imagens de arroz de cada diretório, sem ter que chamar *alinha* 50 vezes. O mesmo vale para um programa Python em Linux:

```
Linux$ python3 arg.py *.jpg
N arg= 3
arg.py
Arborio1.jpg
Arborio2.jpg
```

Windows não faz expansão de wildcard. Porém, o compilador GCC que está incluso em Cekeikon faz expansão de wildcard, para emular esta funcionalidade de Linux:

```
Windows>arg.exe *.jpg
Argc=3
arg.exe
Arborio1.jpg
Arborio2.jpg
```

Onde o programa *arg.exe* foi compilado com compilador GCC incluso em Cekeikon. Assim, a expansão de wildcard só não funciona para Python em Windows ou se você usar um compilador de C++ diferente de GCC em Windows.

Anexo B: Relatórios dos exercícios programas

O mais importante numa comunicação escrita é que o leitor entenda, sem esforço e inequivocamente, o que o escritor quis dizer. O texto ficar “bonito” é um aspecto secundário. Se uma (pseudo) regra de escrita dificultar o entendimento do leitor, essa regra está indo contra a finalidade primária da comunicação. No site do governo americano [1], há regras denominadas de “plain language” para que comunicações governamentais sejam escritas de forma clara. As ideias por trás dessas regras podem ser usadas em outros domínios, como na escrita científica. Resumo abaixo algumas dessas ideias.

(1) Escreva para a sua audiência. No caso do relatório, a sua audiência será o professor ou o monitor que irá corrigir o seu exercício. Você deve enfocar na informação que o seu leitor quer conhecer. Não precisa escrever informações que são inúteis ou óbvias para o seu leitor.

(2) Organize a informação. Você é livre para organizar o relatório como achar melhor, porém sempre procurando facilitar o entendimento do leitor. Seja breve. Quebre o texto em seções com títulos claros. Use sentenças curtas. Elimine as frases e palavras que podem ser retiradas sem prejudicar o entendimento. Use sentenças em ordem direta (sujeito-verbo-predicado).

(3) Use palavras simples. Use o tempo verbal o mais simples possível. Evite cadeia longa de nomes, substituindo-os por verbos (em vez de “desenvolvimento de procedimento de proteção de segurança de trabalhadores de minas subterrâneas” escreva “desenvolvendo procedimentos para proteger a segurança dos trabalhadores em minas subterrâneas”). Minimize o uso de abreviações (para que o leitor não tenha que decorá-las). Use sempre o mesmo termo para se referir à mesma realidade (pode confundir o leitor se usar termos diferentes para se referir a uma mesma coisa). O relatório não é obra literária, não tem problema repetir várias vezes a mesma palavra.

(4) Use voz ativa. Deixe claro quem fez o quê. Se você utilizar oração com sujeito indeterminado ou na voz passiva, o leitor pode não entender quem foi o responsável (Ex: “Criou-se um novo algoritmo” - Quem criou? Você? Ou algum autor da literatura científica?). O site diz: “Passive voice obscures who is responsible for what and is one of the biggest problems with government writing.”

(5) Use exemplos, diagramas, tabelas, figuras e listas. Ajudam bastante o entendimento.

O relatório deve conter pelo menos as seguintes informações:

Identificação

Nomes dos integrantes do grupo, números USP, nome da disciplina, etc.

Breve enunciado do problema

Apesar do enunciado do problema ser conhecido ao professor/monitor, descreva brevemente o problema que está resolvendo. Isto tornará o documento compreensível para alguma pessoa que não tem o enunciado do EP à mão.

Técnica(s) utilizada(s) para resolver o problema

Descreva quais técnicas você usou para resolver o problema. Se você mesmo inventou a técnica, descreva a sua ideia, deixando claro que a ideia foi sua. Se você utilizou alguma técnica já conhecida, utilize o nome próprio da técnica (por exemplo, filtragem Gaussiana, algoritmo SIFT, etc.) juntamente com alguma referência bibliográfica onde a técnica está descrita. Use elementos gráficos como imagens intermediárias e diagramas, pois ajudam muito a compreensão. Não “copie-e-cole” código-fonte, a não ser que seja relevante. Use preferencialmente o pseudo-código.

1 https://plainlanguage.gov/guidelines/

Ambiente de desenvolvimento utilizado

Em qual plataforma você desenvolveu o programa? Como o professor/monitor pode compilar o programa? Você utilizou que bibliotecas?

Operação

Como o professor/monitor pode executar o programa? Que argumentos são necessários para a execução do programa? Há parâmetros que devem ser configurados? Quais arquivos de entrada são necessários? Quais arquivos de saída são gerados?

Resultados Obtidos

Descreva os resultados obtidos. Qual é o tempo de processamento típico? O problema foi resolvido de forma satisfatória?

Referências

Descreva o material externo utilizado, como livros/artigos consultados, websites visitados, etc.