ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
Departamento de Engenharia de Sistemas Eletrônicos
PSI3471 - Fundamentos de Sistemas Eletrônicos Inteligentes
Resolução do Exercício-Programa 2023

No. USP	Nome	Data
11806980	Cezar Gabriel Moreno Almeida Lima	09/07/2023

1. Objetivo

Este exercício-programa [1] tem como objetivo utilizar uma amostra de um banco de dados com fotografias de grãos de arroz para treinar redes neurais artificiais. No conjunto de dados, há 5 tipos de grãos, e as redes devem aprender a classificar imagens dos grãos em um desses tipos. Para realizar este estudo, a **Etapa 1** do projeto consiste em alinhar os grãos horizontalmente e centralizadamente em todas as imagens, para que o posicionamento deles nas imagens seja sempre uniforme. A **Etapa 2,** por sua vez, fará o método K-fold cross validation para gerar subconjuntos de teste e treino dentro das imagens alinhadas e as imagens não alinhadas. Por fim, como resultado, haverão 5 modelos de rede neural para classificar imagens não-alinhadas de grão de arroz e 5 modelos de rede neural para classificar imagens alinhadas de grão de arroz.

2. Técnicas utilizadas no estudo de classificação

Para solucionar o alinhamento de grãos de arroz, utilizou-se a função *Point3d findCenterAndOrientation(Mat_src)* [2] fornecida no enunciado, que, à partir dos cálculos de momentos do objeto **binarizado** na imagem, retorna o ponto central do objeto e o ângulo de inclinação.

a. Etapa 1 - alinhaUmaImagem.cpp

Primeiro, lê a imagem “Jasmine (2).cpp” em duas variáveis: *a* (que será a versão binária) e *q* (que será a versão colorida). Para transformar *a*, utilizou-se a função *threshold(*)* do OpenCV [3] e inclusa no Cekeikon. O centro do objeto é então encontrado baseado em *a*, e plota-se um círculo neste centro, bem como uma linha que indica o ângulo do objeto.

Gera-se uma matriz de rotação através da função *getRotationMatrix2D(*)*, que contém elementos para rotacionar um pixel (x_B, y_B) para um pixel (x_B, y_B). Edita-se a matriz para incluir elementos que transladam o ponto (x_A, y_A) em um deslocamento de (t_x, t_y). Na aplicação do projeto, *tx* e *ty* correspondem aos deslocamentos necessarios para centralizar o centro do grão no centro da imagem. Mais informações sobre essas matrizes podem ser encontradas na apostila de transformações geométricas da disciplina [4]. A função *warpAffine(*)* do openCV, por fim, aplica essas matrizes. Como ela realiza reamostragem, utilizamos uma única matriz já com a rotação e a translação.

$$\begin{bmatrix} x_B \\ y_B \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -sen(\theta) \\ sen(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_A \\ y_A \end{bmatrix}$$

Matriz de rotação do ponto (x_a, y_a) em um ângulo θ

$$\begin{bmatrix} x_B \\ y_B \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x_A \\ y_A \\ 1 \end{bmatrix}$$

Matriz de translação do ponto (x_a, y_a) por um vetor (t_x, t_y)

b. Etapa 1 - alinha.cpp

O algoritmo aplica as transformações para alinhamento de “alinhaUmaImagem.cpp”, mas no(s) arquivo(s) passado(s) pelos parâmetros ao rodar o código. Esta versão do algoritmo foi essencialmente útil para alinhar em lotes as 250 imagens totais do dataset deste exercício-programa. Para isto, utilizou-se a técnica de wildcard expansion [1]. Para mais informações, consulte a seção **3. Instruções dos ambientes de execução**.

c. Etapa 2 - class_orig.ipynb

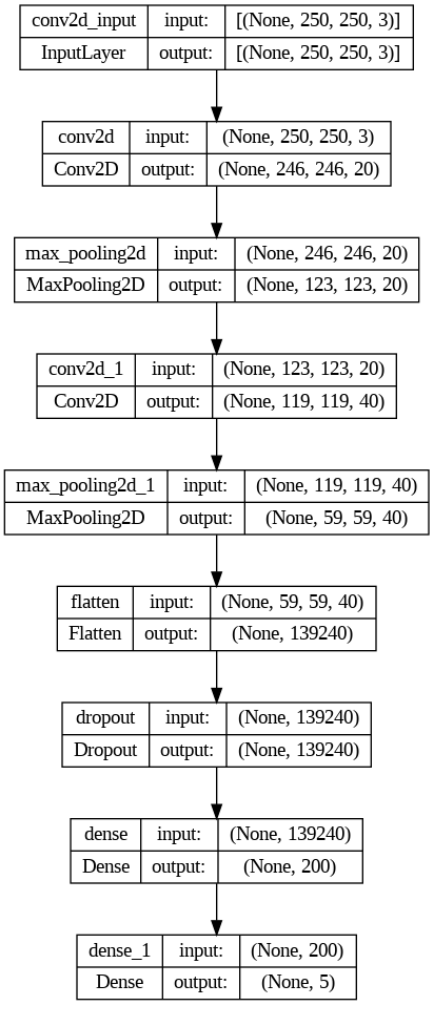
Esta etapa utiliza Keras para criar modelos de CNN [7], treiná-los e obter resultados. Como a quantidade de imagens disponíveis para a resolução do estudo é consideravelmente pequena, será realizado o "5-fold cross validation" [8]. Em cada fold, um dos subconjuntos Sx (x=[1,2,3,4,5]) é escolhido como conjunto de teste para validação do modelo, enquanto os 4 outros subconjuntos são utilizados para treinar o modelo em questão.

O arquivo "rice_1to50.zip" é baixado, extraído [5] e categorizado nas pastas S1, S2, S3, S4 e S5. As imagens e seus respectivos rótulos (presente nos títulos) são carregados em matrizes numpy. Os rótulos passam pelo *one-hot encoding* [6], processo pelo qual a categorização passa a ser descrita por um vetor.

A segunda etapa foi realizada inteiramente em Jupyter Notebooks, no ambiente Google Colab. Há dois arquivos: class_orig e class_alin que estão hospedados nos seus respectivos hyperlinks.

Todas as redes serão então treinadas e validadas para cada um dos *folds*. Todos os modelos de rede, apesar de serem treinadas e validadas de forma diferentes (pois diferem nos subconjuntos de treino e de validação), são constituídos pela arquitetura presente na imagem à direita.

O fit do modelo foi feito com um batch de tamanho 100, em 15 épocas, com uma função de perda "sparse categorical crossentropy".



d. Etapa 2 - class_alin.ipynb

Os arquivos de imagens de grãos alinhados obtidos na Etapa 1 foram enviados para a nuvem. O arquivo class_alin carrega estas imagens, e gera mais 5 modelos de redes neurais. Desta vez, para grãos de arroz alinhados.

3. Instruções dos ambientes de execução

3.1. Primeira Etapa - Alinhamento de grãos

A primeira etapa (alinhamento horizontal de grãos) foi realizada em C, utilizando a biblioteca *cekeikon* (para mais informações e instruções de instalação, acesse o site da biblioteca).
Dois arquivos foram feitos para essa etapa: alinhaUmaImagem.cpp e alinha.cpp. O primeiro lê o arquivo **Jasmine(2).jpg** e gera o arquivo **aComCentroELinha.jpg** e **Jasmine (2)_alin.png**. A primeira imagem gerada indica o centro e o ângulo de inclinação do grão de arroz da imagem, e a segunda a imagem alinhada ainda com os plots.
Para compilar e executar alinhaUmaImagem.cpp, basta abrir o CMD do Windows (com o Cekeikon instalado), navegar para a pasta de instalação do diretório do EP1 e digitar:

```
...\EP1> compila alinhaUmaImagem.cpp -cek && alinhaUmaImagem.exe
```

No Windows, a navegação para o diretório do EP1 pode ser realizada da seguinte forma: abra o diretório no explorador, e depois digite “cmd” na barra do caminho de diretório.

alinha.cpp é um programa que lê imagens *.jpg* passadas pelos parâmetros e gera versões *.png* dessas imagens, nas quais os grãos de arroz estão alinhados horizontalmente. Para executá-la, pode-se passar cada uma das imagens para ser alinhada como parâmetro na linha de código, ou realizar *wildcard expansion*, que alinhará todas as imagens presentes na pasta onde o executável está. Para gerar o executável:

```
...\EP1> compila alinha.cpp -cek
```

Agora basta mover o alinha.exe para a pasta wildcardExpansion, abrir o caminho “/EP1/wildcardExpansion” no CMD, e rodar:

```
...\EP1\wildcardExpansion> alinha.exe *.jpg
```

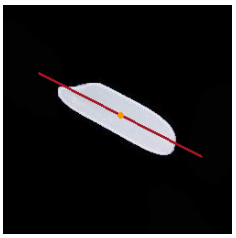
3.2. Segunda etapa - Aprendizado de máquina

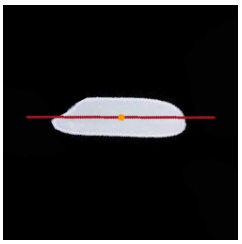
Para executar um dos notebooks, primeiro vá em “Ambiente de Execução” > “Alterar o tipo de ambiente de execução” e mude o Acelerador de Hardware para “GPU”. Salve e utilize “Ctrl + F9” para executar todo o notebook sequencialmente.
As imagens não precisam ser inseridas no site, ambos os notebooks baixam automáticamente e extraem os arquivos necessários.
Cada célula de treino de rede (para cada um dos casos *Sx*) pode ser executada individualmente para re-treinar e obter uma acurácia de teste nova.

4. Resultados

O resultado do processo alinhamento dos grãos pode ser visto na sequência de imagens abaixo, utilizando como exemplo a segunda imagem dos grãos Jasmine. Este mesmo procedimento foi aplicado para todas as 250 imagens por wildcard expansion e salvas na pasta wildcardExpansion.


Jasmine(2).jpg


aComCentroELinha.jpg


Jasmine(2)_ centroELinha.png


Jasmine(2).png

O resultado dos diferentes modelos de CNN treinados para cada um dos folds pode ser visto na tabela abaixo.

	Conjunto de teste no Fold	Acurácia (%)	Média da Acurácia	Desvio-padrão (σ) da Acurácia	Tempo de treino (s) / Tempo de predição (s)
Imagens de grãos de arroz **não-alinhados**	S1	96.00%	93.60%	2.332	5.88 / 0.19
	S2	94.00%			6.39 / 0.20
	S3	96.00%			5.99 / 0.17
	S4	92.00%			6.38 / 0.18
	S5	90.00%			5.91 / 0.16
Imagens de grãos de arroz **alinhados**	S1	96.00%	96.00%	1.2649	21.57 / 0.42
	S2	98.00%			5.14 / 0.20
	S3	96.00%			5.20 / 0.18
	S4	94.00%			5.86 / 0.17
	S5	96.00%			6.12 / 0.17

Poucos erros foram encontrados, como na S3-alin.cnn, que contém apenas 3 erros de classificação.

5. Conclusões

Realizar o alinhamento das imagens aumentou significativamente a acurácia final registrada. A diferença obtida só não foi maior porque o conjunto de teste é de apenas 50 imagens e, portanto, 1 único erro de classificação pode reduzir o *score* em 2%. Porém, ao realizar vários treinamentos para testar os diferentes modelos, foi possível constatar que os modelos de grãos alinhados são mais robustos. Praticamente sempre alcançam uma acurácia maior que 90%, enquanto os modelos para grãos alinhados chegaram a alcançar até ~70%.

O tempo de treino e predição permaneceram bem semelhantes entre os 2 tipos de alinhamentos de grãos, então caso se tratasse de uma aplicação em tempo real, o tempo gasto para alinhar os grãos talvez não fosse interessante.

6. Referências

[1] PSI-3471: Fundamentos de Sistemas Eletrônicos Inteligentes Primeiro semestre de 2023 Exercício-programa Prof. Hae Data de entr

[2] Binary Image Orientation - Stack Overflow

[3] Image Thresholding

[4] Transformações geométricas

[5] how to download zip file from google Drive to colab - Stack Overflow

[6] One-hot-encoding, o que é? | Medium

[7] Rede neural convolucional (CNN) em Tensorflow/Keras

[8] Cross Validation: Avaliando seu modelo de Machine Learning | by Eduardo Braz Rabello | Medium