**PSI3471 - Fundamentos de Sistemas Eletrônicos Inteligentes**

**Exercício Programa 1**

**André Lucas Pierote Rogrigues Vasconcelos (11356540)**

**Leonardo Isao Komura (11261656)**

**alinha**

Esse programa é utilizado para teste das funções responsáveis por centralizar e rotacionar a imagem, a deixando centrada e alinhada horizontalmente.

O programa possui três funções:

* **findCenterAndOrientation**: Encontra o centro de massa e o ângulo de inclinação do grão de arroz. (Fornecida previamente)

def findCenterAndOrientation(img):

    m *=* cv2.moments(img)

    cen\_x *=* int(m["m10"]*/*m["m00"])

    cen\_y *=* int(m["m01"]*/*m["m00"])

    theta *=* 0.5 *\** math.atan2(2*\**m["mu11"], m["mu20"]*-*m["mu02"])

*return* cen\_x, cen\_y, theta

* **center**: Centraliza o grão de arroz, baseando-se no centro de massa dela. Utiliza a função warpAffine, que realiza a translação baseada na distância entre o centro do grão de arroz com o centro da imagem.

def center(img):

    print(img.shape)

    center *=* int(img.shape[1]*/*2)

    deslocate *=* np.float32(np.empty((img.shape)))

*for* i *in* range(img.shape[0]):

        x, y, theta *=* findCenterAndOrientation(img[i])

        M *=* np.float32([[1,0,center*-*x],[0,1,center*-*y]])

        deslocate[i] *=* cv2.warpAffine(img[i], M, (img.shape[1], img.shape[2]))

*return* deslocate

* **rotate**: Rotaciona a imagem, deixando-a alinhada horizontalmente

def rotate(img):

   rotation *=* np.float32(np.empty((img.shape)))

   height, width *=* img.shape[1:3]

*for* i *in* range(img.shape[0]):

      x, y, theta *=* findCenterAndOrientation(img[i])

      rotM *=* cv2.getRotationMatrix2D(center*=*(x,y), angle*=*theta*\**180*/*math.pi, scale*=*1)

      rotation[i] *=* cv2.warpAffine(src*=*img[i], M*=*rotM, dsize*=*(width, height))

*return* rotation

Após a declaração das funções, foi feita a leitura da imagem “Jasmine2.jpg” e atribuída a primeira posição de um tensor (apenas para ficar igual ao formato utilizado nos programas posteriores).

st *=* "Jasmine" *+* "2" *+* ".jpg"; st *=* "orientacao"*+*"/"*+*st

a*=*cv2.imread(st,0)

a*=*cv2.resize(a,(125,125),interpolation*=*cv2.INTER\_AREA)

img *=* np.float32(np.empty((1,125,125)))

img[0] *=* a

imgplot *=* plt.imshow(img[0], cmap*=*'gray')

plt.show()

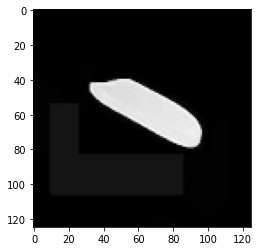


Então, foi feita a centralização da imagem:

deslocated *=* center(img)

imgplot *=* plt.imshow(deslocated[0], cmap*=*'gray')

plt.show()



E a rotação dela:

rotated *=* rotate(deslocated)

imgplot *=* plt.imshow(rotated[0], cmap*=*'gray')

plt.show()



É possível notar que a imagem não ficou perfeita devido ao polígono cinza abaixo do grão de arroz. Entretanto, quando testado em uma imagem sem ele (como nas que serão utilizadas para treinamento e classificação de uma rede neural) isso não ocorre.

**classifica\_jpg**

Esse programa utiliza as imagens disponibilizadas, mas já centralizadas, e as classifica pelo uso de uma CNN implementada por meio do Keras sem alinhá-las horizontalmente.

Há uma função nova nesse programa:

* **le**: faz a leitura dos arquivos e gera um tensor e um vetor para armazenar as imagens e suas classificações. (Disponibilizada previamente).

def le(diretorio,nl,nc,inic,fim):

    nclasses*=*len(diretorio)

    n*=*nclasses*\**(fim*-*inic*+*1)

    AX*=*np.empty((n,nl,nc),np.uint8)*;*

    AY*=*np.empty((n,),np.uint8)*;*

    j*=*0; k*=*0

*for* nome *in* diretorio:

*for* i *in* range(inic,fim*+*1):

            st *=* nome *+* " (" *+* str(i) *+* ")."*+*"jpg"; st *=* nome*+*"/"*+*st; *#print(st)*

            a*=*cv2.imread(st,cv2.IMREAD\_GRAYSCALE)

*if* a *is* None: print("Erro leitura",st); sys.exit()

            a*=*cv2.resize(a,(nl,nc),interpolation*=*cv2.INTER\_AREA)

            AX[k,:,:]*=*a; AY[k]*=*j

            k*+=*1

        j*+=*1

*return* AX,AY

Primeiramente, foi feita a leitura dos arquivos:

diretorio*=*["Arborio","Basmati","Ipsala","Jasmine","Karacadag"]

nl *=* 125

nc *=* 125

AX,AY*=*le(diretorio,nl,nc,1,100)

qx,qy*=*le(diretorio,nl,nc,101,1000)

Após isso, foi criada dois grupos separados das imagens, um para treinamento e outro para validação, para, então, poder ser utilizado validação cruzada no treinamento:

*# Validação cruzada*

*from* sklearn.model\_selection *import* train\_test\_split

x\_train, x\_val, y\_train1, y\_val1 *=* train\_test\_split(AX, AY, test\_size *=* 0.1, random\_state*=*1)

print('Qtde de treino: {}'.format(len(x\_train)))

print('Qtde de validação: {}'.format(len(x\_val)))

Então, transformou-se os vetores de classificação em matrizes (dados inteiros para binários):

y\_train *=* keras.utils.to\_categorical(y\_train1, 5)

y\_val *=* keras.utils.to\_categorical(y\_val1, 5)

qy2 *=* keras.utils.to\_categorical(qy, 5)

Após o tratamento dos dados, criou-se a uma Rede Neural Convolucional:

model *=* Sequential()

model.add(Conv2D(50, kernel\_size*=*(3,3), activation*=*'elu', input\_shape*=*(125, 125, 1) ))

model.add(MaxPooling2D(pool\_size*=*(2,2)))

model.add(Conv2D(100, kernel\_size*=*(3,3), activation*=*'elu'))

model.add(MaxPooling2D(pool\_size*=*(2,2)))

model.add(Dropout(0.25))

model.add(Conv2D(50, kernel\_size*=*(3,3), activation*=*'elu'))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(50, activation*=*'elu'))

model.add(Dense(5, activation*=*'softmax'))

Como é possível observar no código acima, utilizou-se filtros 3x3 com 50 filtros, com exceção da segunda camada. Além disso, foi escolhido a função de ativação “elu” e foi utilizado um “dropout” de 25%, para evitar overfitting.

Ademais, foi escolhido o otimizador Adam e função de custo de Entropia Cruzada:

opt*=*optimizers.Adam()

model.compile(optimizer*=*opt, loss*=*'categorical\_crossentropy', metrics*=*['accuracy'])

Após isso, foi realizado o treinamento da rede com um tamanho de batch de igual a 25, 50 e 100 e com 10 épocas:

t0*=*time()

history *=* model.fit(x\_train, y\_train,

                    batch\_size*=*25,

                    epochs*=*10,

                    validation\_data*=*(x\_val, y\_val))

t1*=*time(); print("Tempo de treino: %.2f s"*%*(t1*-*t0))

score *=* model.evaluate(qx, qy2, verbose*=*False)

print('\nTest loss: %.4f'*%*(score[0]))

print('Test accuracy: %.2f %%'*%*(100*\**score[1]))

print('Test error: %.2f %%'*%*(100*\**(1*-*score[1])))

print("\n")

t2*=*time()

QP2*=*model.predict(qx); QP*=*np.argmax(QP2,1)

t3*=*time(); print("Tempo de predicao: %.2f s"*%*(t3*-*t2))

nerro*=*np.count\_nonzero(QP*-*qy); print("nerro=%d"*%*(nerro))

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tamanho do batch | Precisão | Taxa de erros | Nº de erros | Tempo de treino |
| 25 | 95,71% | 4,29% | 193 | 112,85s |
| 50 | 94,64% | 5,36% | 241 | 104,78s |
| 100 | 93,38% | 6,62% | 298 | 105,74s |

Como é possível observar, a rede piorou conforme foi aumentado o número de dados de treino. Isso pode ocorreu, pois os dados não foram ajustados, ou seja, os grãos de arroz estão em posições inconsistentes, o que prejudicam o treinamento da rede. Mesmo assim, pode-se dizer que os resultados são bem decentes.

**classifica\_png**

Esse programa é praticamente idêntico ao “classifica\_jpg”, com a diferença de centralizar (redundantemente) e rotacionar as imagens dos grãos de arroz. Para isso, foi utilizada as funções “center” e “rotate”.



Exemplo de grão de arroz processado

Além disso, o código também grava as imagens processadas no formato png, pelo uso da função criada:

def escreve(img, diretorio, inic, fim):

    dir *=* 0

    bias *=* int(img.shape[0]*/*5)

*for* nome *in* diretorio:

*for* i *in* range(inic,fim*+*1):

            st *=* nome *+* " (" *+* str(i) *+* ")."*+*"png"; st *=* nome*+*"/"*+*st

            cv2.imwrite(st, img[dir*+*i*-*inic])

        dir *=* dir*+*bias

Finalmente, foi realizado o treinamento da rede com as imagens formatadas e seu teste, obtendo-se o seguinte resultado:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tamanho do batch | Precisão | Taxa de erros | Nº de erros | Tempo de treino |
| 25 | 96,51% | 3,49% | 157 | 110,99s |
| 50 | 97,93% | 2,07% | 93 | 107,71s |
| 100 | 98,44% | 1,56% | 70 | 104,81s |

Pode-se concluir que essa rede apresentou o comportamento esperado, tornando-se mais eficiente com um maior número de dados de treino. Além disso, comparando-a com a rede treinada com os dados não formatados, ela mostrou um resultado consideravelmente melhor.