Creating a Training Data Set

```
import cv2
import numpy as np
import os
```

```
#importing black image as background to display the contour of each grain
bg = cv2.imread("black.jpg")
bg = cv2.cvtColor(bg, cv2.COLOR_BGR2RGB)
```

```
#address to the directories of broken rice grain training set and full rice grain training set
DIR_broken = r'/content/broken'
DIR_full = r'/content/full'
```

```
#storing the shape of background image that is to be used to resize the training images to fit the background
IMG_SIZE = (bg.shape[1], bg.shape[0])
```

```
#creating lists to store the training images of respective classes
broken_data = []
full_data = []

#importing all the images of broken grains, resizing them to the size of background image
for img in os.listdir(DIR_broken):
    img_path = os.path.join(DIR_broken, img)
    img_arr = cv2.imread(img_path)
    img_arr = cv2.resize(img_arr, IMG_SIZE)
    broken_data.append(img_arr)

#importing all the images of broken grains, resizing them to the size of background image
for img in os.listdir(DIR_full):
    img_path = os.path.join(DIR_full, img)
    img_arr = cv2.imread(img_path)
    img_arr = cv2.resize(img_arr, IMG_SIZE)
    full_data.append(img_arr)
```

```
#method to preprocess the images
def img_preprocess(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #converting the image to grayscale
    blur = cv2.GaussianBlur(gray, (11,11), 0) #blurring the image to avoid noise
    canny = cv2.Canny(blur, 30, 150, 3) #getting the edges
    dilate = cv2.dilate(canny, (1,1), iterations = 2) #sharpening the edges
    return dilate

#lists to append the processed images
broken = []
full = []

for i in broken_data:
    broken.append(img_preprocess(i))

for i in full_data:
    full.append(img_preprocess(i))

#finding the contours of the broken rice grains and storing them in a list
broken_cnt = []
for i in broken:
    (cnt, h_broken) = cv2.findContours(i.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    broken_cnt.append(cnt)

#finding the contours of the full rice grains and storing them in a list
full_cnt = []
for i in full:
    (cnt, f_broken) = cv2.findContours(i.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    full_cnt.append(cnt)

#method to increase the size of contours to print on the background image
def scale_contour(contour, scale):
    moments = cv2.moments(contour) #finding the moments of the contour
    midX = 0
    midY = 0
    if (moments['m00']!=0):
        midX = int(round(moments["m10"] / moments["m00"]))
        midY = int(round(moments["m01"] / moments["m00"]))
    mid = np.array([midX, midY])
    contour = contour - mid
    contour = (contour * scale).astype(np.int32) #scaling the contour coordinates to the desired size
    contour = contour + mid
    return contour

epoch = 0
#putting the contours of the broken rice grain filled in white on the black background and saving it in the given address
for j in broken_cnt:
    for i in range(0, len(j)):
        bg_copy = bg.copy()
        cnt_scaled = scale_contour(j[i], 10)
        cv2.fillPoly(bg_copy, pts =[cnt_scaled], color=(255,255,255)) #filling in the contours with white
        cv2.imwrite(r'/content/train/broken_train/broken_%04d.png'%(epoch+1), bg_copy)
        epoch+=1

#putting the contours of the full rice grain filled in white on the black background and saving it in the given address
epoch = 0
for j in full_cnt:
    for i in range(0, len(j)):
        bg_copy = bg.copy()
        cnt_scaled = scale_contour(j[i], 10)
        cv2.fillPoly(bg_copy, pts =[cnt_scaled], color=(255,255,255))
        cv2.imwrite(r'/content/train/full_train/full_%04d.png'%(epoch+1), bg_copy) #filling in the contours with white
        epoch+=1
```

Training Model

```
#importing all the dependancies
```

```
import cv2
import numpy as np
import os
import random
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation, Dropout

IMG_SIZE = (224,224) #setting the image size for the neural network

train = [] #list to store the training data

DIR = r'/content/train' #address to the directory of training data
category = ['broken_train', 'full_train'] #folder names for the respective classes

#iterating through both the folders and adding the images to the training data list
for c in category:
    folder =  os.path.join(DIR, c)
    label = category.index(c) #0 for broken, 1 for full
    for img in os.listdir(folder):
        img_path = os.path.join(folder, img)
        img_arr = cv2.imread(img_path) #reading the images
        img_arr = cv2.resize(img_arr, IMG_SIZE) #resizing the images to the size described above for CNN
        train.append([img_arr, label]) #returning the data along with labels

#shuffling the data so the model can learn better
random.shuffle(train)

#break the dataset and store the features in X_train and labels in y_train
X_train = []
y_train = []

for features, labels in train:
    X_train.append(features)
    y_train.append(labels)

#converting the data into numpy array
X_train = np.array(X_train)
```

```
y_train = np.array(y_train)

#normalising the pixel values
X_train = X_train/255



#creating the CNN model for classiying
model = Sequential()

#first CNN layer with 32 layers and feature extractor of size 3x3
model.add(Conv2D(32, (3, 3), input_shape = X_train.shape[1:]))
model.add(Activation('relu')) #Rectified Linear Unit as Activation function
model.add(MaxPooling2D(pool_size = (2, 2))) #max pooling layer of size 2x2

#second CNN layer with 32 layers and feature extractor of size 3x3
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu')) #Rectified Linear Unit as Activation function
model.add(MaxPooling2D(pool_size = (2, 2))) #max pooling layer of size 2x2

#third CNN layer with 64 layers and feature extractor of size 3x3
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu')) #Rectified Linear Unit as Activation function
model.add(MaxPooling2D(pool_size = (2, 2))) #max pooling layer of size 2x2

model.add(Flatten()) #Flattening to get 1D array of features
model.add(Dense(64)) #defining the hidden layer with 64 neurons
model.add(Activation('relu')) #Rectified Linear Unit as Activation function
model.add(Dropout(0.5))
model.add(Dense(2))
model.add(Activation('softmax')) #activation function that returns the probability of a data lying in either classes

model.compile(loss = 'sparse_categorical_crossentropy', optimizer = 'rmsprop', metrics = ['accuracy']) #compiling the model with the

model.fit(X_train, y_train, epochs = 10, validation_split = 0.1) #training the model over the training dataset

model.save("grain_classifier.h5") #saving the CNN classifier model
```

```
Epoch 1/10
125/125 [==============================] - 270s 2s/step - loss: 0.6726 - accuracy: 0.6021 - val_loss: 0.6476 - val_accuracy: 0.
Epoch 2/10
125/125 [==============================] - 267s 2s/step - loss: 0.6409 - accuracy: 0.6316 - val_loss: 0.6485 - val_accuracy: 0.
Epoch 3/10
125/125 [==============================] - 268s 2s/step - loss: 0.6066 - accuracy: 0.6832 - val_loss: 0.6569 - val_accuracy: 0.
Epoch 4/10
125/125 [==============================] - 275s 2s/step - loss: 0.5545 - accuracy: 0.7383 - val_loss: 0.6979 - val_accuracy: 0.
Epoch 5/10
125/125 [==============================] - 272s 2s/step - loss: 0.4837 - accuracy: 0.7784 - val_loss: 0.7500 - val_accuracy: 0.
Epoch 6/10
125/125 [==============================] - 270s 2s/step - loss: 0.4188 - accuracy: 0.8149 - val_loss: 0.7366 - val_accuracy: 0.
Epoch 7/10
125/125 [==============================] - 270s 2s/step - loss: 0.3469 - accuracy: 0.8485 - val_loss: 0.8737 - val_accuracy: 0.
Epoch 8/10
125/125 [==============================] - 271s 2s/step - loss: 0.2817 - accuracy: 0.8833 - val_loss: 1.0035 - val_accuracy: 0.
Epoch 9/10
125/125 [==============================] - 277s 2s/step - loss: 0.2248 - accuracy: 0.9138 - val_loss: 1.1359 - val_accuracy: 0.
Epoch 10/10
125/125 [==============================] - 274s 2s/step - loss: 0.1842 - accuracy: 0.9314 - val_loss: 1.2277 - val_accuracy: 0.
```

Testing Model

```
#importing all the dependencies
import cv2
import os
import numpy as np
import pandas as pd
from tensorflow.keras.models import load_model

#importing the output csv file, the background image, and the saved model
op = pd.read_csv('submission.csv')
bg = cv2.imread('black.jpg')
model = load_model('grain_classifier.h5')
```

```
DIR = r'/content/test' #the address to the tesing data
img = cv2.imread(r'/content/test/image_4.jpg') #reading the test images one by one

#converting the test image to grayscale
img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (11,11), 0) #blurring the images to avoid noise
canny = cv2.Canny(blur, 30, 150, 3) #reading the edges of the grains
dilate = cv2.dilate(canny, (1,1), iterations = 2) #dilating the image to sharpen and thicken the edges

#getting the contours and storring it in a list
(cnt, heirarchy) = cv2.findContours(dilate.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

#method to scale the contour coordinates to the desired size
def scale_contour(contour, scale):
    moments = cv2.moments(contour) #finding the moments
    midX = 0
    midY = 0
    if (moments['m00']!=0):
        midX = int(round(moments["m10"] / moments["m00"]))
        midY = int(round(moments["m01"] / moments["m00"]))
    mid = np.array([midX, midY])
    contour = contour - mid
    contour = (contour * scale).astype(np.int32) #scaling the contours to a desired size
    contour = contour + mid
    return contour

#painting the contours on to the black background and saving the images in the given directory's address
epoch = 0
for i in cnt:
    bg_copy = cv2.imread('black.jpg') #reading the black image
    cnt_scaled = scale_contour(i, 10) #resizing the contour
    cv2.fillPoly(bg_copy, pts =[cnt_scaled], color=(255,255,255)) #painting the contours on to the black image
    cv2.imwrite(r'/content/test/test_image5/image_%04d.jpg'%(epoch+1), bg_copy) #saving the image in the desired address
    epoch+=1

op.iloc[4, 1] = int(len(cnt))#the number of contours is the number of grains
```

```
folder = r'/content/test/test_image5' #folder for the contour images of the test dataset
X_test = []
for img in os.listdir(folder):
    img = os.path.join(folder, img)
    img_arr = cv2.imread(img) #reading the images
    img_arr = cv2.resize(img_arr, (224, 224)) #resizing the images to the desired size for the network to read
    X_test.append(img_arr) #Adding it to the list of test data

X_test = np.array(X_test) #converting the list to array as neural network expects numpy as the input array

pred = model.predict(X_test) #predict the probablities of the contours to be of a roken rice grain or a full rice grain
pred = pred.tolist() #converting the prediction array to list for our convinience

count = 0
for [broken, full] in pred:
    if(broken>full):
        count+=1 #counting the number of data that has probability of being broken grain greater than that of being a full grain

op.iloc[4,2] = count #the count is the number of broken grains in the image

op.to_csv('final_submission4.csv') #adding the values to the op dataframe and saving it to the final csv file
```

```
20/20 [==============================] - 11s 521ms/step
```

✓ 4m 10s completed at 5:37 PM ● ×