Rice Seed Classification

Jean-Sebastien Roger

October 18, 2020

Abstract

In this experiment we examined the classification of rice seed images into one of two categories: 'Proper' or 'Broken'. Visually, those classified as 'Proper' were full seeds that did not appear to be damaged, those classified as 'Broken' seemed to be a fraction the size of a full rice seed. The datasets we used in this experiment contain a test set for validation and training set with hundreds of images of rice seeds, each image with its own classification. Our goal was to build a function which could correctly classify a new rice seed image as a 'Broken' or 'Proper'. Our experiment was successful in that we were able to correctly classify a new dataset of rice seed images.

1 Introduction

The dataset utilized for this experiment was a list of images, each image was of a single seed of rice surrounded by a black background. As mentioned before, each seed of rice was classified as either 'Broken' or 'Proper', Figure 1 shows a sample of the data images we were dealing with.

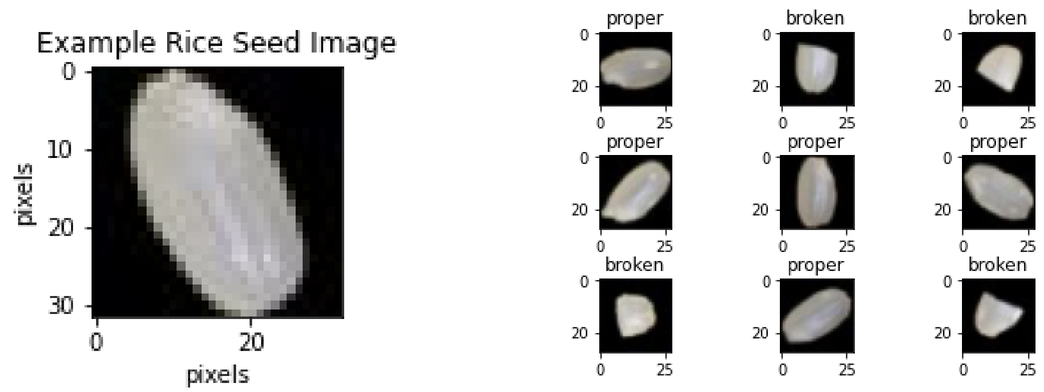


Figure 1: On the left is an example of the rice seed images in our dataset. On the right, we see multiple rice seeds and above each image is the assigned classification that we will be using to train our thresholds.

2 Theory

Our image classification task is to conclude whether or not a rice seed image is broken. We are interested in first reading in multiple '.png' images, each image is read in as an array of rgb elements. Using conversions from rgb to grayscale to binary data elements, we will methodically shape the image into a shape that we can plot and thus we can measure. Once the images can be mathematically represented we can then proceed to using our measurements to classify the images. Finally, we can functionalize our process and use it to quickly identify the optimal thresholds, thereby improving our classifications.

3 Procedures

```
#Let's try looking at a single image
image = X_train[0]

# Plot the original image
plt.figure(figsize=(3,3)) # Set figure size
plt.imshow(image) # Plot the image
plt.title("original") # Set a title for the image we just plotted
```

We have read in our list of images and used plotting libraries to set our figure size and show the original rice seed image.

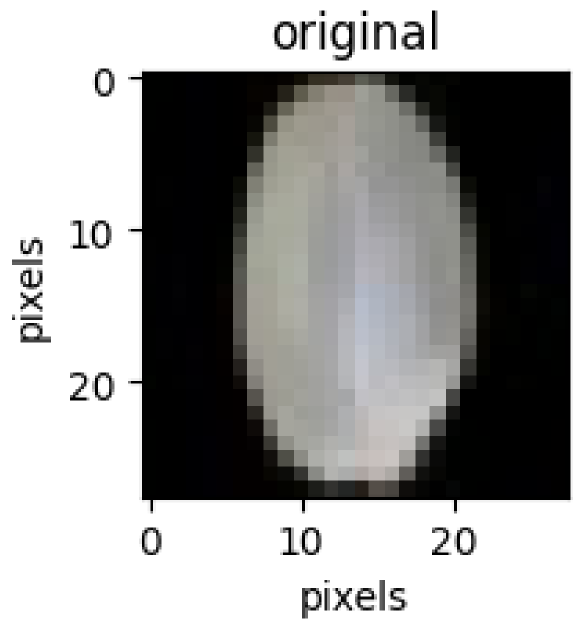


Figure 2: A plot showing the original image of a single grain of rice.

```
# Gray Conversion
# Each rgb array element is converted to a single number
```

```
gray = rgb2gray(image)

plt.figure(figsize=(2,2))

# Plot the grayscale image
plt.imshow(gray, cmap=plt.cm.gray) # The cmap is required here
plt.title("gray converted")
plt.xlabel("pixels")
plt.ylabel("pixels")
```

We have now taken our list of images and converted each one from an rgb array into a list of grayscale intensity values.

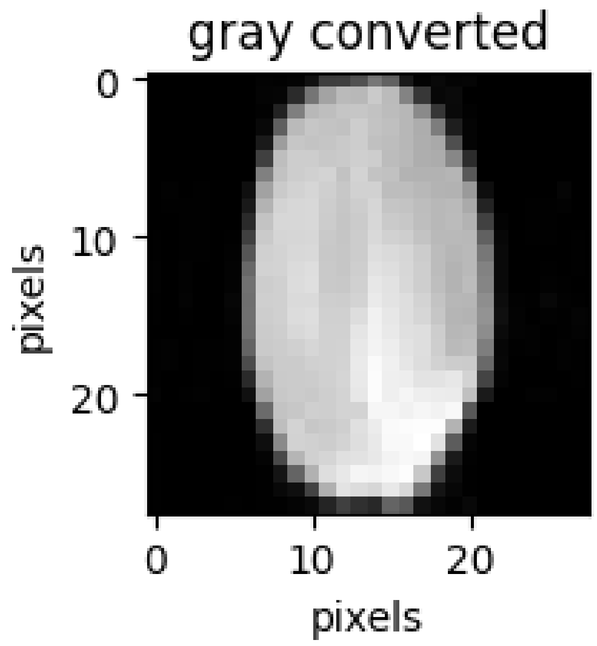


Figure 3: A plot showing the image of a single grain of rice converted into a grayscale in order to make identifying the outline of the shape possible by using a threshold on the intensity.

```
# Binary Conversion
# This line computes a threshold from a grayscale image
threshold = threshold_otsu(gray)

# Binary representation whether a pixel is above or below the threshold
binary = gray > threshold

# Plot a pure white and black image
plt.imshow(binary, cmap=plt.cm.gray)
plt.title("Binary converted")
plt.xlabel("pixels")
plt.ylabel("pixels")
```

We have now made our image an array of 1s and 0s indicating if it is background or the rice seed image.

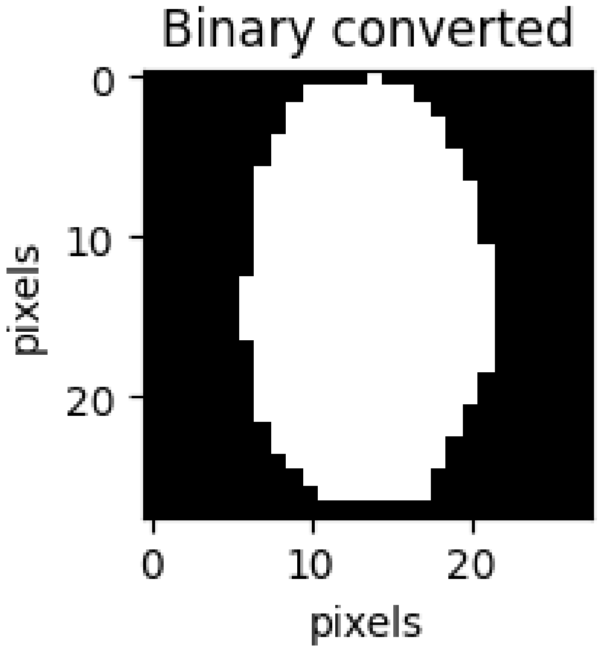


Figure 4: A plot showing the image of a single grain of rice converted into a binary image of black and white. This conversion allows us to classify each pixel as background or foreground.

```
# Calculating area

# This line of code will label the binary array with 1s and 0s
label_im, nb_labels = ndimage.label(binary)

# Using the grayscale image we convert the labels into a region
regionprops = measure.regionprops(label_im, intensity_image=gray)
regionprop = regionprops[0]

print("area is",regionprop.area)
print("major axis length is", regionprop.major_axis_length)
print("minor axis length is", regionprop.minor_axis_length)

area is 332
major axis length is 27.110738879206334
minor axis length is 15.628814201948895
```

4 Analysis

In this section we will review the results of the classification procedure and formulate a function which simplifies the amount of code needed to perform future classifications. Code will be demonstrated here to present the final function, however, in the interest of not overcrowding the Lab report, we have moved the package import section and various other code chunks to the Appendix.

```
def quantify_area(image):
# Converts to grayscale
    gray = rgb2gray(image)

    # Find the threshold to outline
    threshold = threshold_otsu(gray)

    # Set binary representation of the image
    binary = gray > threshold
    label_im, nb_labels = ndimage.label(binary) # add labels for conversion

    # Measure the region and return a region object
    regionprops = measure.regionprops(label_im, intensity_image=gray)
    regionprop = regionprops[0]

    # Calculate the area of the region
    area = regionprop.area
    return area
```

The function above returns a single area calculation of our image. Now we can proceed to using this function to quickly find the area of each rice seed in our dataset; then use these area measurements to select a threshold with which to classify out of sample datasets like our testing set.

```
X_train_area = []
for image in X_train:
    area = quantify_area(image)
    X_train_area.append(area)

area_threshold = 261

#classify whether the image is a proper seed or a broken seed
train_y_pred = []
for area in X_train_area:
    if area > area_threshold:
        train_y_pred.append(0)
    else:
        train_y_pred.append(1)

#plot scatter with threshold line
plt.figure(figsize=(5,3))

# Plot the scatter plot
plt.scatter(range(len(X_train_area)),
        X_train_area, c=y_train, cmap=plt.cm.coolwarm)
plt.axhline(y=area_threshold)
plt.title("blue:proper seed, red: broken seed")
plt.xlabel("rice seed images")
plt.ylabel("area (px)")

plt.show()
```

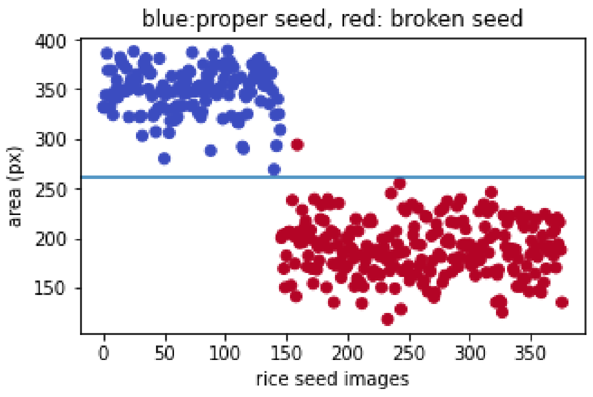


Figure 5: In this plot, each individual rice seed image makes up each respective circle. Along the x-axis, we have the rice seed number, along the y-axis, we have the area calculated from our area function. We can see how the original classifications (depicted by the color) are very much in line with our area threshold.

Above in Figure 5, we see a visual representation of our classification threshold on our training data. We see below in Figure 6, a confusion matrix to empirically represent how well our threshold performed on our training data. The code for the confusion matrix is provided in the Appendix.

```
#calculate confusion matrix
cnf = confusion_matrix(y_train, train_y_pred)

#confusion matrix in figure
plt.figure(figsize=(3,3))
plot_confusion_matrix(cnf, classes=["proper","broken"])
plt.show()
```

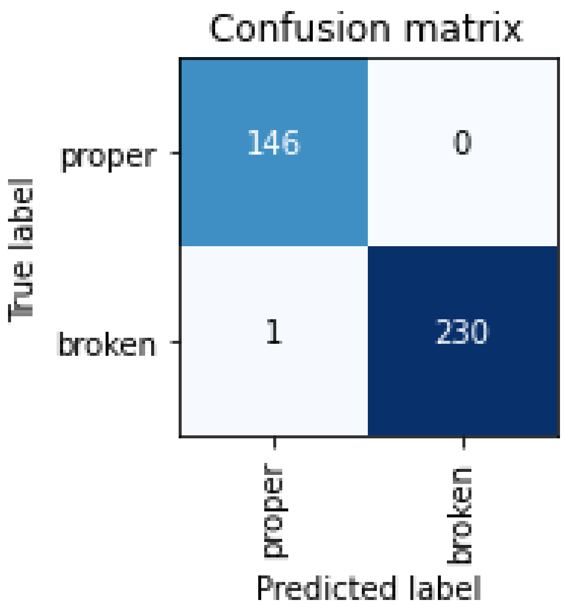


Figure 6: In the confusion matrix shown here we can see that we have only mis-classified 1 rice seed as 'proper' when it is in fact 'broken'.

Let us take a look at the seed we mis-classified in order to see how we can improve our classification.

```
# Set the space for the subplots
plt.subplots_adjust(wspace=0.2, hspace=0.8)

# Create an array of 9 random numbers
index = np.random.randint(0,X_train.shape[0],size=8)
for i in range(len(X_train)):
  if y_train[i] != train_y_pred[i]:
    index = np.append(index, i)

for i, idx  in enumerate(index):
    plt.subplot(3,3,i+1) # add to the subplot space we have created
    if y_train[idx] == 0:
        label_1 = "proper"
    else:
        label_1 = "broken"
    if train_y_pred[idx] == 0:
        label_2 = "proper"
    else:
        label_2 = "broken"
    label = label_1 + " : " + label_2
    plt.title(label) # Add the titles we just created
   plt.imshow(X_train[idx])
```

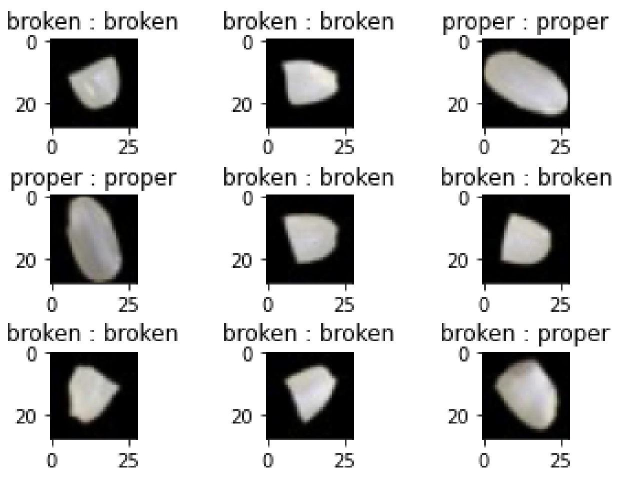
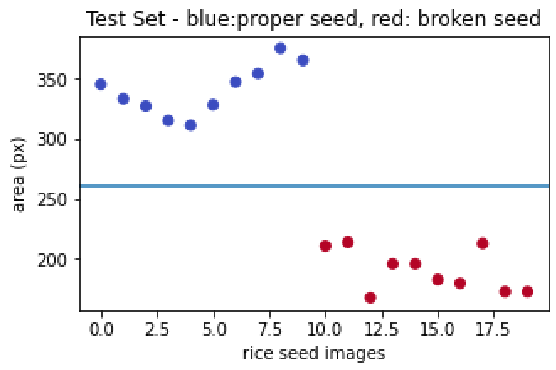


Figure 7: Our mis-classified seed (the seed in the bottom right corner) does have an area comparable to some of the 'proper' seeds.

4.1 Validation

We can now move on to testing our classification threshold on an out of sample set of images, here is where our testing set comes into play. In the Appendix we have included the code for the classifying and visualizing the testing set. In Figure 8 we can see that our threshold performs very well and we can see from the plot how there is a very clear distinction between the areas of the 'proper' seed and the areas of the 'broken' seeds. The performance in this out of sample validation allows us to move forward with our classifier using our current threshold of 261.



Confusion matrix, without normalization

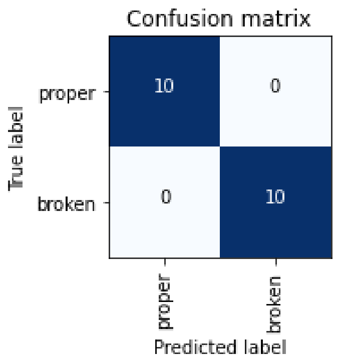


Figure 8: The image on the left is a plot showing the area of each seed in our test set and their true classification. We can clearly see that all of these seeds pass our threshold and would be classified correctly by our classifier. The image on the right is a confusion matrix similar to the one showed before, here we can see that all 10 'proper' seeds have been correctly classified and all 10 'broken' seeds have also been correctly classified

4.2 Classifier

Finally, we can create a classifier which will take in an image parameter and a threshold parameter for the cutoff area. This function will calculate the area of the image and will output a prediction of 'proper' or 'broken' based on the threshold provided.

```
def manual_classifier(image, area_threshold):
    gray = rgb2gray(image)
    threshold = threshold_otsu(gray)
    binary = gray > threshold
    label_im, nb_labels = ndimage.label(binary)
    regionprops = measure.regionprops(label_im, intensity_image=gray)
    regionprop = regionprops[0]
    area = regionprop.area
    if area > area_threshold:
        return 'proper'
    else:
        return 'broken'
```

5 Conclusions

This lab provided some very valuable lessons on how to approach basic image classification. There is huge value in being able to take an image file and convert the image into empirical values which a computer can then use to make decisions. This allows for large scale reproducible classification which can make work much more efficient and less prone to human error. The difficulty arises in establishing a proper classifier which does not require human intervention.

In this lab we witnessed how our classifier performed very well, however, it was prone to slight error because it was only taking into account the area of the image. Had we received more broken rice seeds with large areas we may have reached a different conclusion. It is very interesting to walk through a simple classification like this, because it makes it much easier to see how adding another dimension of measurement can improve classification dramatically. For example, if we also had a threshold for the sharpness of the corners within the image, or how jagged some of the edges are, then we can continue to improve our classifier making it much more accurate.

6 Appendix

Package import

```
import numpy as np # Import numpy (np)
import math, os, sys # Import math, os, and sys
import itertools # Import itertools

import matplotlib.pyplot as plt # Import matplotlib.pyplot for ploting (plt)
```

```
plt.style.use('default')
from scipy import ndimage

# Below we are importing specific functions
from skimage import measure, morphology
from skimage.io import imsave, imread
from skimage.color import rgb2gray
from skimage.filters import threshold_otsu
from skimage.transform import resize

# Importing functions
from sklearn import svm, datasets
from sklearn.metrics import confusion_matrix
import pandas as pd
```

Validation code

```
test_y_pred = []
for area in X_test_area:
    if area > area_threshold:
        test_y_pred.append(0)
    else:
        test_y_pred.append(1)

#plot scatter with threshold line
plt.figure(figsize=(5,3))
plt.scatter(range(len(X_test_area)),
        X_test_area,
    c = y_test,
    cmap = plt.cm.coolwarm)
plt.axhline(y=area_threshold)
#plt.plot([100,0],[100,350],'k-',lw=2)
plt.title("Test Set - blue:proper seed, red: broken seed")
plt.xlabel("rice seed images")
plt.ylabel("area (px)")
plt.show()

#calculate confusion matrix
cnf = confusion_matrix(y_test, test_y_pred)

#confusion matrix in figure
plt.figure(figsize=(3,3))
plot_confusion_matrix(cnf, classes=["proper","broken"])
plt.show()
```