Progress Report 1: Image Classification Corn Kernels

Progress Thus Far:

Due to the task of image classification being completely new to me, my time has been spent familiarizing myself with novel concepts regarding image data. Image data is fundamentally different from the typical data tables we have seen in Principles of Data Science up until now. With a single data frame, all preprocessing can be done with respect to a single object, while an image set consists of many objects. Thus I will use this first report to talk about key points I have learned and how I have applied them to my data set at this point.

Within the image set there are ~14000 corn kernel images, each of which has some properties that must be normalized across the set in order to properly train a model on the data. The first important quality about image data is that there are three dimensions to an image after it is loaded into a python environment using either Matplotlib or OpenCV.

The dimensions represent the height, width and channels of an image (H x W x C). Height and width are fairly intuitive to understand, they represent how many pixels are used to represent the image with respect to both directions of a 2D image. More pixels means a higher resolution, less pixels means less resolution and in a purely visual sense, less information. The channels are the less intuitive 3rd dimension, and there are typically 3 channels in an image, each of which represents either red, green or blue. As can be seen in figure 1 below, these channels can be viewed separately. The purpose of paying attention to these fundamental qualities of image data is that skillful manipulation of these qualities is often necessary in training a model for image classification.

Models often expect images to be of the same aspect ratio, height and width. This fact usually implies a need for image resizing, and where images need to be increased in size there are different methods of interpolating missing data into the images. This page from OpenCV has some details on different interpolation algorithms, and how they measure up against one another.

For the corn kernel image set, I wanted to understand the best way to preserve vital image data, while also standardizing the size of all images. In order to do so I created a vector of all shapes within my image set, and created an aspect ratio boxplot to view understand the distribution a bit. I also gathered basic summary statistics on the height and width of the data in order to similarly understand what a reasonable resize for all images would be. I settled at essentially the mean of all respective values.

In reading I learned that center cropping images was a technique that could be used in cases where vital image data is contained primarily in the center of an image set. Given this image set is largely centered corn kernels, I decided this would be a useful technique for preprocessing. Thus I setup the code necessary to center crop images which are above the mean size (140 x 140), and to resize images which are below the mean size. This will introduce some distortions, as there is a truly wide range of aspect ratios as can be seen in the below boxplot, however I believe the compromise makes logical sense from my inexperienced point of view.

Beyond image size I learned that normalization of pixel values is in fact important, and can help lead to faster convergence of models for image classification tasks, thus reducing compute time. In figure 3 below an example of normalized and standardized images can be seen. I wanted to understand the visual difference each would make to build some intuition about how I was manipulating data. In researching and observation I learned that despite pixel values only ranging of 0-1 in normalized data visual details are still quite apparent and easily resolved by the human eye.

Finally I did some reading on the Histogram of Oriented Gradients (HOG) feature extraction technique for data. I found that there is a function from sklearn for HOG, and further learned the basic outline of what operations are done in order to perform HOG on an image set.

Next Steps:

Next Week:

First I will be taking my resized images and running them through HOG in order to extract meaningful feature data from them. Then I will begin testing a KNN model with different preprocessing techniques to see what degree of performance I can achieve with the methods I have learned so far. I don't expect to see very good performance, but rather am aiming to setup the framework to ease the effort of testing different models, as well as testing different combinations of preprocessing techniques.

In order to use the KNN model I will be taking the usual steps of loading in the image classifications, (as they are not included in the images themselves), splitting the data into training, testing and validation sets, and attempting to tweak relevant hyperparameters and preprocessing steps to achieve a first iteration image classification model. After I feel the KNN model has been pushed as far as possible I will turn to other models, particularly Support Vector Machine and Random Forest models. In some early research I learned that they can handle multiclass classification tasks, and thus are viable options to test. For each I intend to spend some time reading about the fundamentals and math behind each model to build some basic understanding, and finally I will funnel in the different forms of preprocessed data to each model before evaluating and comparing against the KNN model.

Beyond Next Week:

After testing some different preprocessing combinations and models for this task I would aim to fine tune my process further by introducing other methods of feature extraction. Methods I came across as potential options in research are Scale-Invariant Feature Transform, and Speeded Up Robust Features, (SIFT, SURF). Given that none of the models or feature extraction techniques in combination perform reasonably well (say above 65% accuracy) I would move into further research to learn of other models or techniques which could improve performance. Ideally I would like to get near the performance of individuals within the Kaggle competition (~80%), though I understand that the tool set and experience I have are quite limited, and will thus be satisfied to learn some basics and make improvements and iterations within the set of models and techniques I have mentioned here. Obviously working towards my final report will happen in this time period as well.

Figures:

Figure 1: Channels

```
fig, axs = plt.subplots(3,3, figsize=(15,15))
axs[0,0].imshow(plt.imread(corn[1])[:,:,0], cmap='Reds')
axs[0,1].imshow(plt.imread(corn[1])[:,:,1], cmap='Greens')
axs[0,2].imshow(plt.imread(corn[1])[:,:,2], cmap='Blues')
axs[1,0].imshow(plt.imread(corn[2])[:,:,0], cmap='Reds')
axs[1,1].imshow(plt.imread(corn[2])[:,:,1], cmap='Greens')
axs[1,2].imshow(plt.imread(corn[2])[:,:,2], cmap='Blues')
axs[2,0].imshow(plt.imread(corn[3])[:,:,0], cmap='Reds')
axs[2,1].imshow(plt.imread(corn[3])[:,:,1], cmap='Greens')
axs[2,2].imshow(plt.imread(corn[3])[:,:,2], cmap='Blues')
plt.show()
```

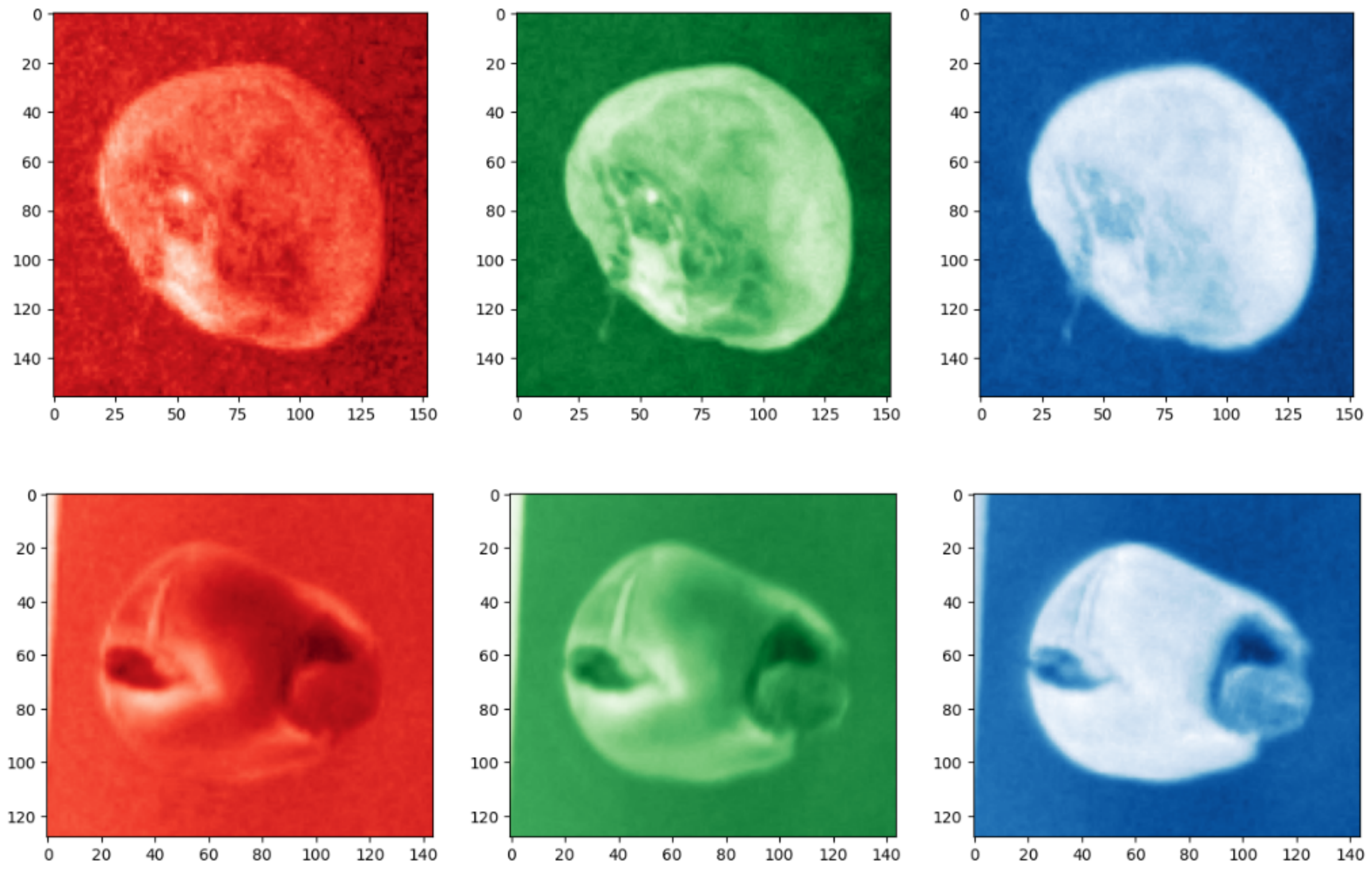


Figure 2: Boxplot of Aspect Ratios

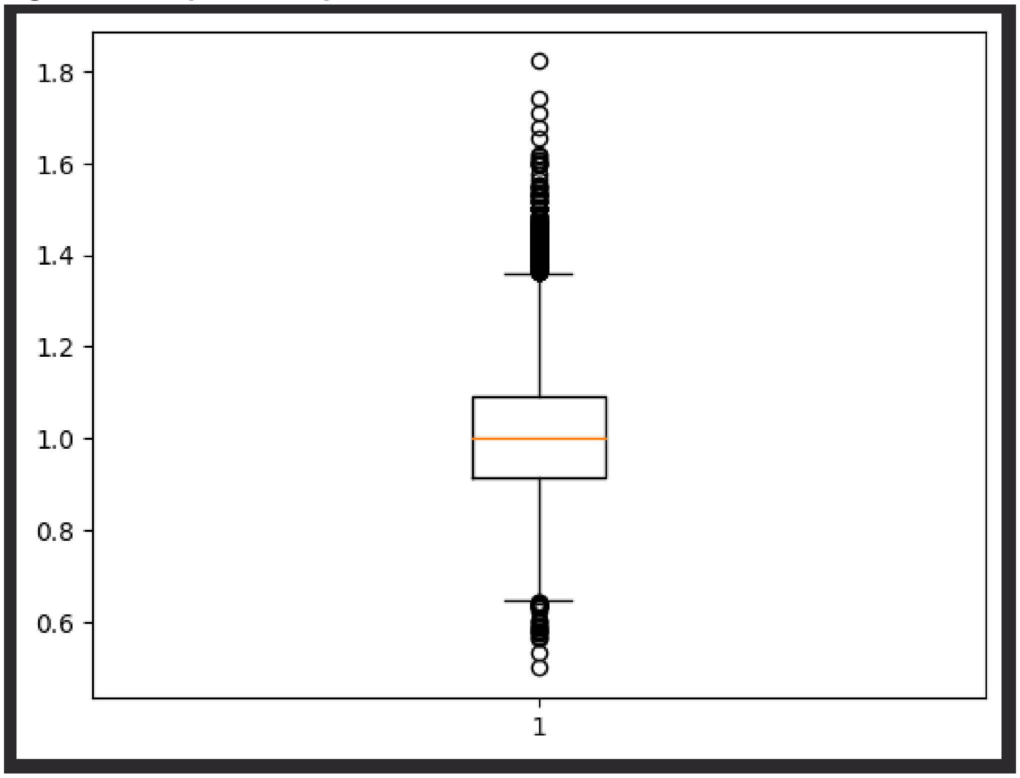
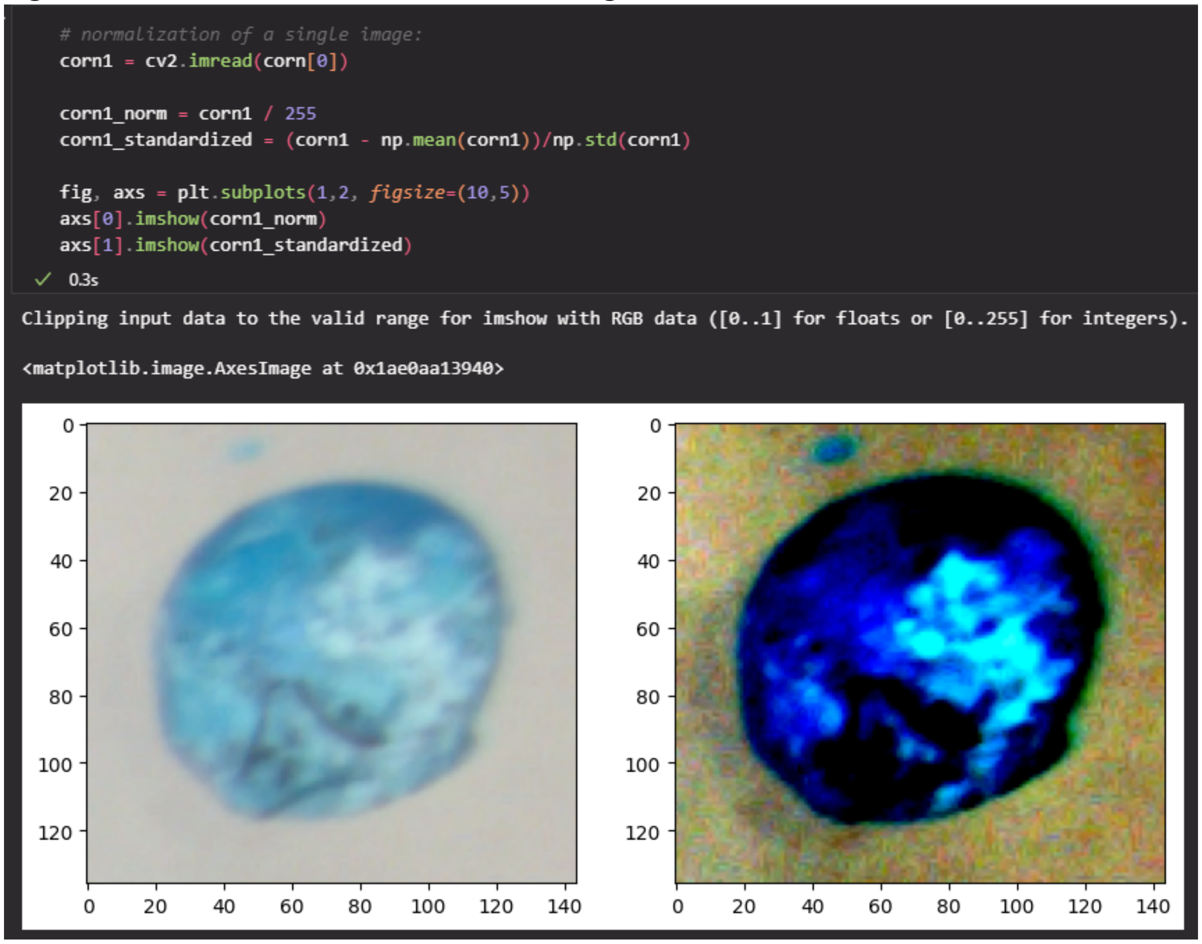


Figure 3: Normalized and Standardized image:



```
# normalization of a single image:
corn1 = cv2.imread(corn[0])

corn1_norm = corn1 / 255
corn1_standardized = (corn1 - np.mean(corn1))/np.std(corn1)

fig, axs = plt.subplots(1,2, figsize=(10,5))
axs[0].imshow(corn1_norm)
axs[1].imshow(corn1_standardized)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

<matplotlib.image.AxesImage at 0x1ae0aa13940>
```