Multiclass Image Classification using KNN & Random-Forest Classifiers: Corn Kernels

Jon Tanner Nelson

Introduction:

Image classification is a foundational task in the overarching field of Computer Vision (CV). Computer vision itself is evidently of increasing importance as we see the rise of large AI systems such as GPT-4–a LLM which has shown improved performance in several testing categories when using vision–from OpenAI, and PaLM-E–an embodied Large Language Model (LLM) which is able to perform some basic tasks when given commands–from Google. Both of these systems are using advanced computer vision systems in order to translate linguistic input into some sort of dynamic output. Beyond CVs role in enabling LLMs to access an impressive range of novel applications, CV is essential to other application spaces such as self driving cars, medical X-Ray analysis (and other diagnostic processes in healthcare). The list of applications for CV is quite broad already and in time will likely grow to be as broad as the uses humans themselves have for vision as primary sensory modality. My interest in image classification and motivation for this project evolves out of a desire to pursue work in and understanding of the broad application space of CV.

This project is itself not about dynamic, integrated computer vision, rather it is the simpler task of image classification, a multiclass classification problem. The problem and accompanying data comes from Kaggle where participants were challenged to classify images of corn kernels into one of four different categories: 1. Pure, 2. Broken, 3. Discolored, 4. Silkcut. The labeled data set includes 14,322 images, accompanying this training data is 3,479 unlabeled images. The problem thus is to analyze these images using any tools available in order to predict classes of corn kernel images.

Kaggle competitions are frequently used to make progress towards real world problems which organizations or individuals are willing to reward users for working on, however, as stated by the individual who created this competition (Rob Mulla), the competition is meant to help "new data scientists grow in their knowledge and understanding of machine learning concepts". Seeing as I have no experience in the space of CV and only this class (Principals of Data Science) as experience in machine learning, I believed this was an ideal entry point to build up basic knowledge about image classification and CV thus.

While working on this problem I was unable to breach into the range of best predictions which can be seen on the leaderboard for the competition, which peak around .82 accuracy. My best predictions had ~.56 accuracy on the test data I subset from the original training data, which would be uncompetitive in the context of this competition. While my performance in this task is relatively poor, I believe the experience I have gained was still quite valuable, as most foundational concepts with respect to this problem were entirely new to me.

Approach:

The approach I used for this problem was entirely based in iterative learning. Given my lack of knowledge about the subject of image classification I decided that doing basic research regarding preprocessing of image data, methods of feature extraction used in object recognition, and classifiers which may be suited to this sort of task would be the best starting point.

Initially I set out to use as many classifiers, methods of feature extraction and preprocessing techniques as possible. However due to time constraints, and challenges in searching for optimal hyperparameters for each combination of techniques and tools I was only able to test two classifiers paired with two methods of feature extraction.

Overall my approach consisted of learning the foundations of the data and tools I would be using, then making naive assumptions about how I might use those tools for the classification problem. While this approach is obviously not sound in terms of producing quality results, I believe it is inevitable for one who is building basic experience. Instead of walking through each detail of my workflow, I will breakdown the basics of the classifiers (models) and feature extraction tools I used in this project, accompanied by my reasoning for choosing each.

Classifiers:

KNN and Random-Forest classifiers from the Scikit-learn library were selected as first choices for different reasons respectively. The KNN classifier was selected because I believed it would perform relatively poorly on this task, as my assumption of image data was that there is not likely to be the sort of orderly features which would naturally work well with a KNN model. Observations are compared by their distances, such as Manhattan distance, Euclidean distance, and Minkowski distance. Such distance comparisons thus make somewhat coarse assumptions about the data, such as the same differentiating elements occurring in the same location within a vector. Within an image however, location of some common interesting element (such as discoloration) may in fact be different across many instances of that element, and thus a raw vector representation of the pixels would not accurately represent these sorts elements which vary in their location. This observation I made while learning the fundamentals of image data lead me to believe that KNN would be a nice benchmark to include, as it would likely perform poorly (I predicted), and would thus highlight the strengths of other models tested.

The Random-Forest classifier appeared an appealing option because its purported strengths of being able to handle a large number of features, being resistant to overfitting and having the ability to capture complex patterns within the data. Using the principle of the *wisdom of crowds*, the Random Forest classifier uses an ensemble of decision trees which each process a subset of the total feature set representing the data, thus allowing many different feature combinations to be tested simultaneously. This sort of divide and conquer technique allows for the more important feature combinations to rise above the noise, as the mean predictions of the decision trees will tend towards some consensus. So long as the data provided to the Random-Forest classifier is itself meaningful, (i.e., the data is of a quality which allows clear differentiation of classes), then the model should be able to pickup on the patterns and produce reasonably good predictions.

Feature Extractors:

Given there are a large number of methods to extract features from image data Speeded-Up Robust Features (SURF), Local Binary Patterns (LBP), Basic Intensity Features, and more which need not be mentioned, I decided to settle on a few simply because they were a starting point, and I knew to little to truly judge what a better or worse starting point might be.

Histogram of Oriented Gradients (HOG) works by capturing the structure of an image through oriented gradients, where intensity and direction of gradients detected in the image created by shifts in color, texture, or stark lines are captured in relatively simple visualizations. Below is an image displaying a visualization of the extracted features found by HOG, which clearly depict the most significant edges and district gradients of value in the image. The actual steps involved in HOG feature extraction are somewhat involved but all handled by the `hog` function from the sklearn.feature module. HOG requires greyscale conversion to work and produces a large number of features. Due to the large number of features generated by HOG, Principal Component Analysis (PCA) was applied after initial tests in my work, as feature reduction served to effectively reduce noise and increase the signal of relevant features.

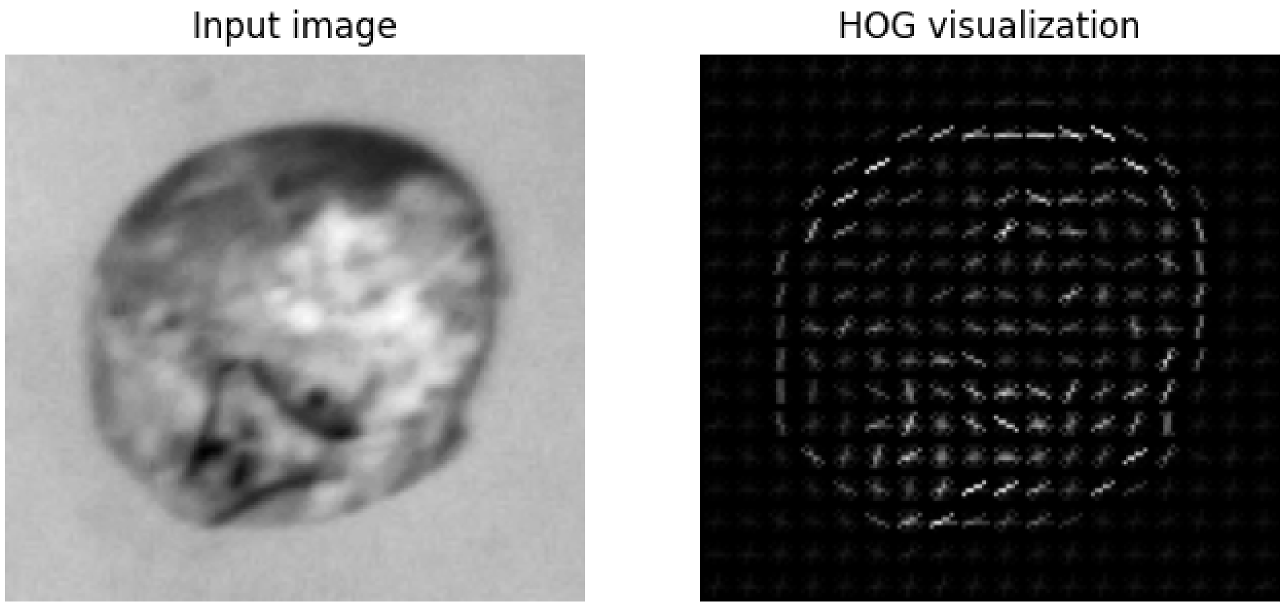


Fig-2: HOG visualization produced using `hog(visualize = True)`

The second method of feature extraction used was Scale Invariant Feature Transform (SIFT), which is rather involved in terms of implementation, as the `SIFT_create()` function from the OpenCV package "cv2" was not built to output feature extractions which that are of equal shape and thus ready to be used within a classifier. SIFT works by identifying *key points* which represent interesting elements of an image. Key points are accompanied by *key point descriptors* which captures the characteristics of a given key point numerically, and thus are the features which would be entered into a classification model. A quality of SIFT which made it appealing to me initially as I read into it was its ability to assign orientation to the keypoints it finds. This means that objects captured in images can be identified despite having substantially different orientations, as the algorithm captures orientation of the "blobs". Given the image set I am working with come in all orientations, I thought the quality of identifying similar blobs regardless of orientation would be desirable.

Results:

Each classifier and feature extractor were first tested naively, without searching for the best hyperparameters, as I wanted to take time to digest and understand the impact that different setting changes had on model performance. I will thus share the naive results for a combination before showing results following hyper parameter tuning.

First to note the distribution of samples overall:

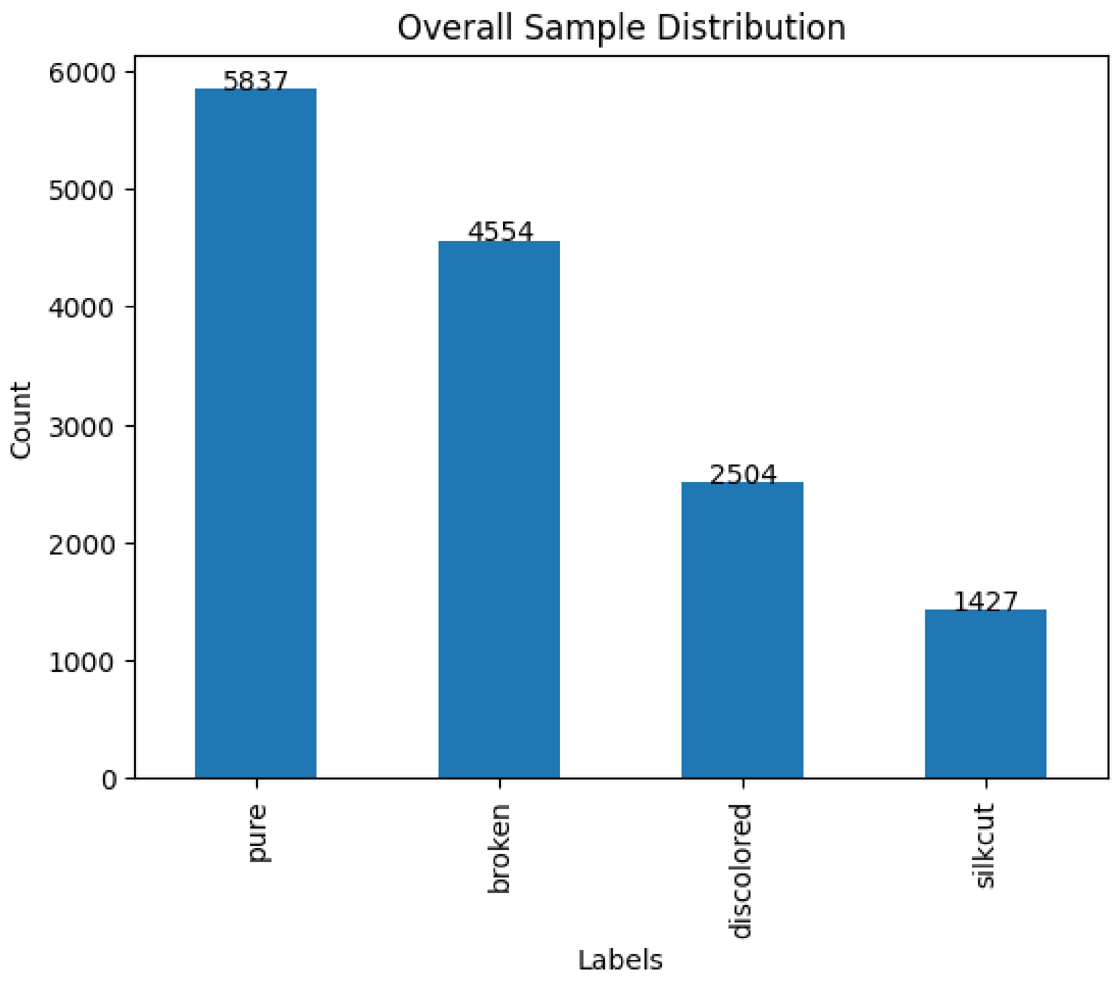
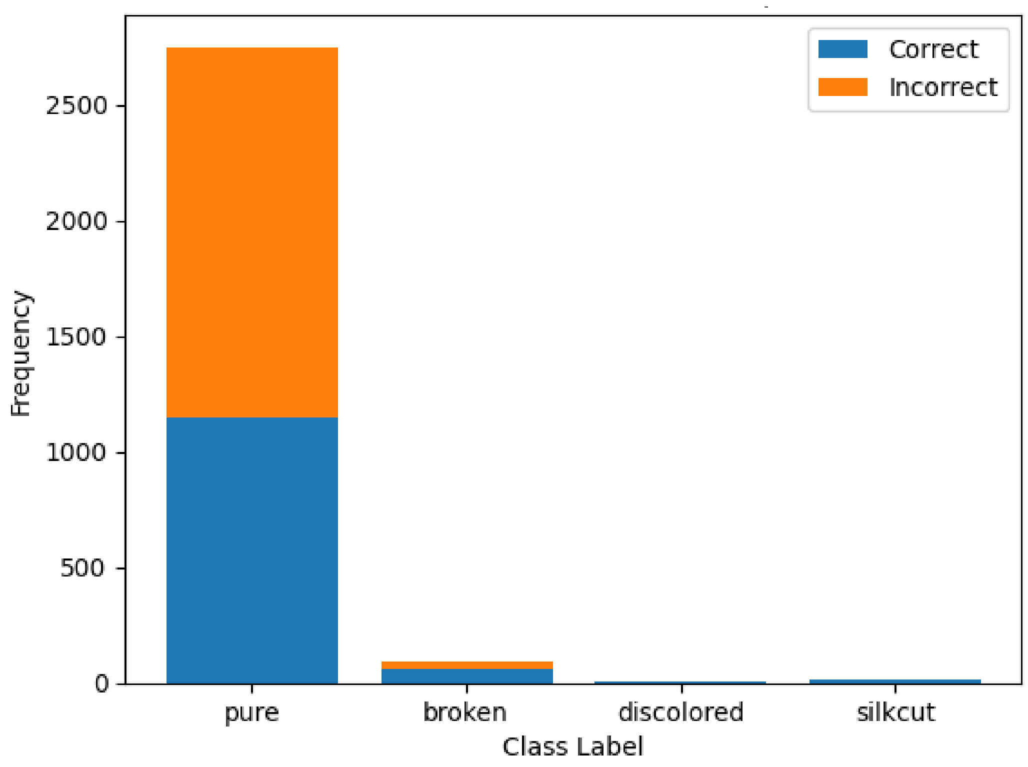
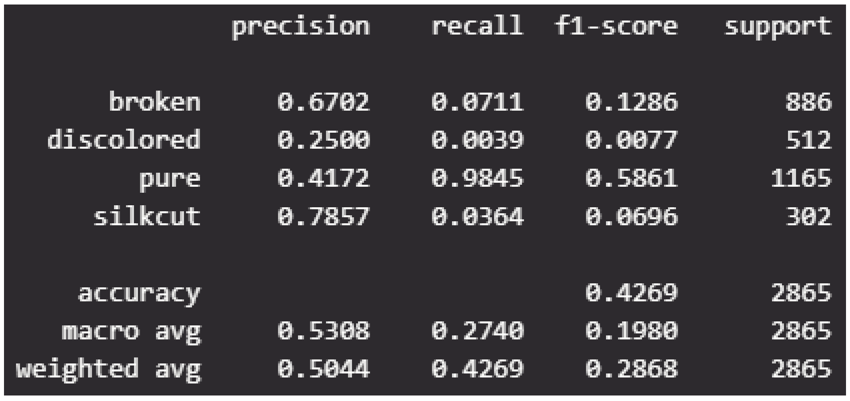


Fig-2: Percentages of each sample class: pure: 40.76%, broken: 31.8%, discolored: 17.48%, silkcut: 9.96%. The imbalanced data set suggests that accuracy is a bad metric to evaluate model efficacy by, which is strongly supported by model predictions made with naive settings.

KNN:

HOG + KNN, no Hyperparameter Tuning





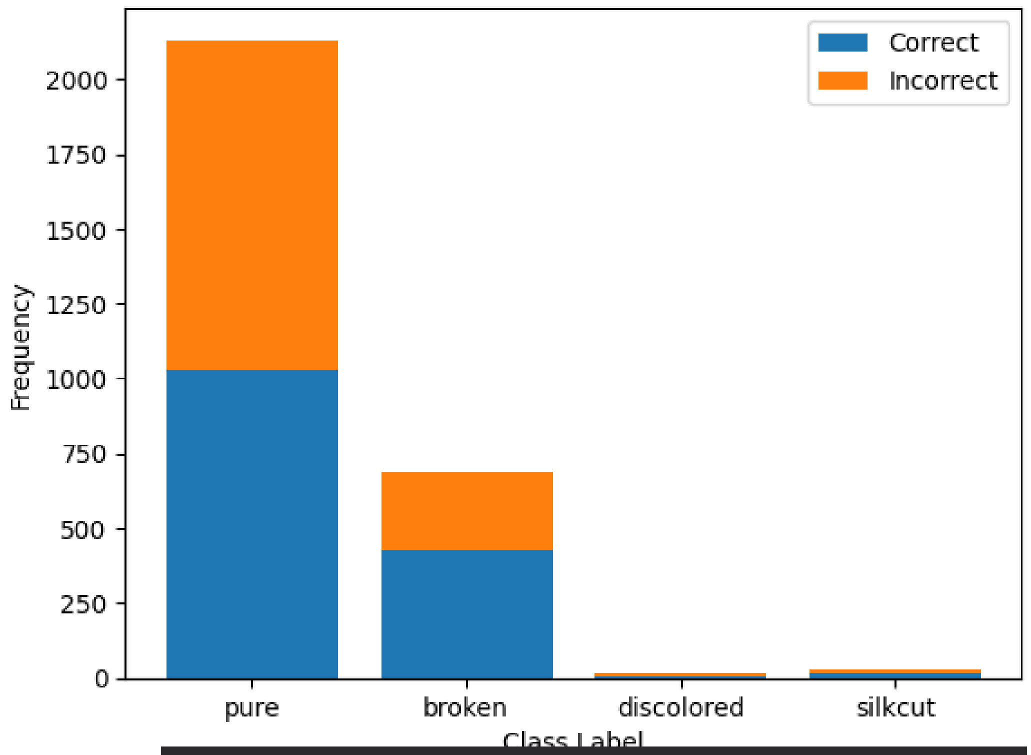
```
              precision    recall  f1-score   support

      broken     0.6702    0.0711    0.1286       886
  discolored     0.2500    0.0039    0.0077       512
        pure     0.4172    0.9845    0.5861      1165
     silkcut     0.7857    0.0364    0.0696       302

    accuracy                         0.4269      2865
   macro avg     0.5308    0.2740    0.1980      2865
weighted avg     0.5044    0.4269    0.2868      2865
```

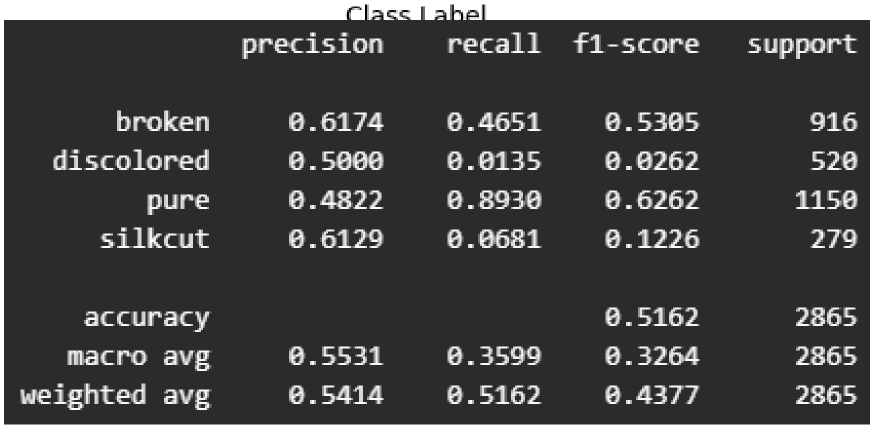
Naive training of KNN lead to almost 100% 'pure' predictions. Resulting in what appears to be reasonable accuracy but what is actually completely ineffective. The macro average is the truly interesting metric to pay mind to throughout these model iterations, though I admit I was not as focused on improving this metric during my process as I should have been

The parameters which produced these results were:

- n_neighbors = 10

HOG + PCA + KNN with Hyperparameter Tuning





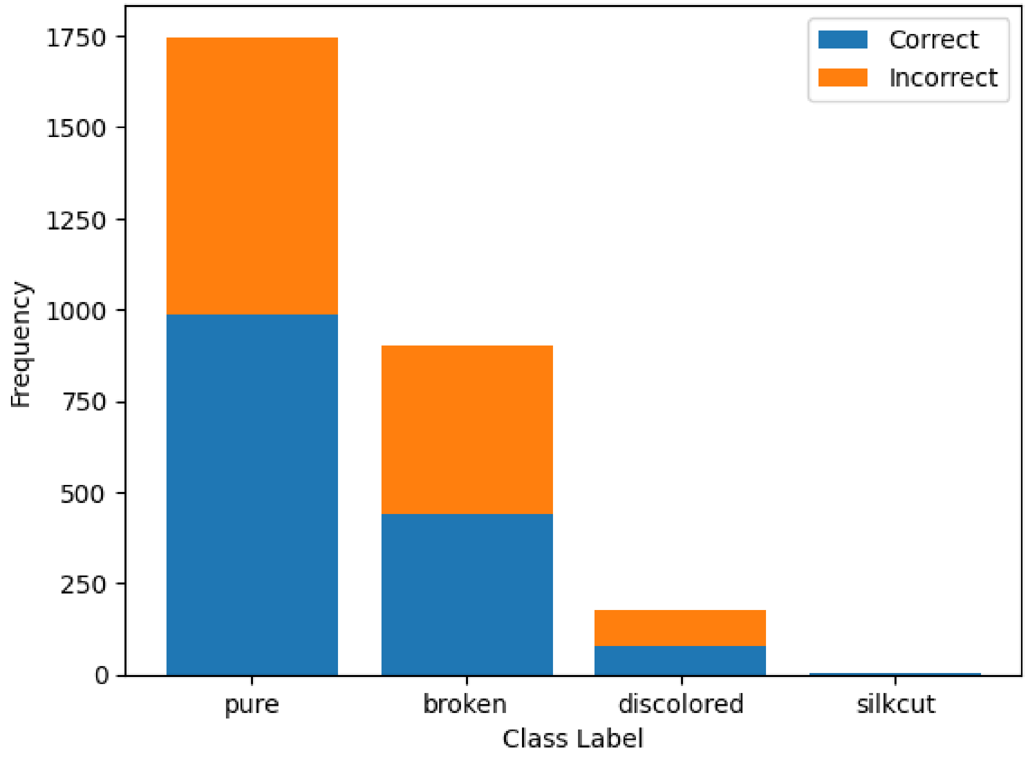
```
              precision    recall  f1-score   support

      broken     0.6174    0.4651    0.5305       916
  discolored     0.5000    0.0135    0.0262       520
        pure     0.4822    0.8930    0.6262      1150
     silkcut     0.6129    0.0681    0.1226       279

    accuracy                         0.5162      2865
   macro avg     0.5531    0.3599    0.3264      2865
weighted avg     0.5414    0.5162    0.4377      2865
```

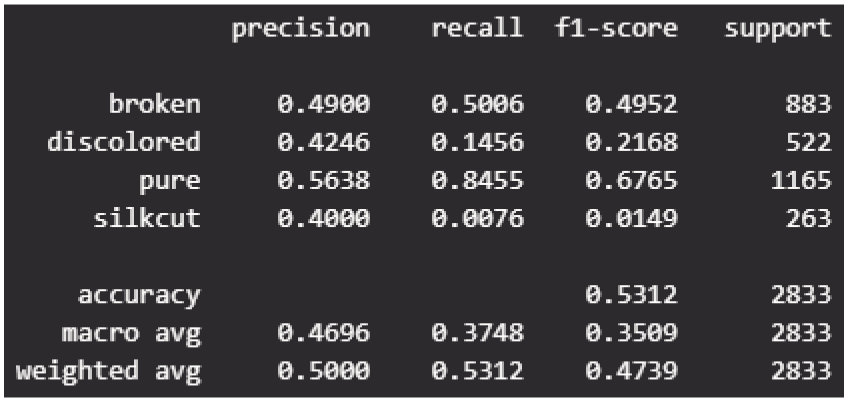
After performing hyperparameter tuning using basic cross validation with 5 folds and a grid search across the n_neighbors parameter for the KNN model and n_components for PCA dimensionality reduction the results have improved substantially. Noting the macro average as improving by ~12%, and the fact that the two most dominant classes in the sample set are seeing representation in predictions suggests that there is some meaningful classification happening at this point.

The parameters which produced these results were:

- n_neighbors = 50,
- n_components = 35

SIFT + PCA + KNN with Hyperparameter Tuning





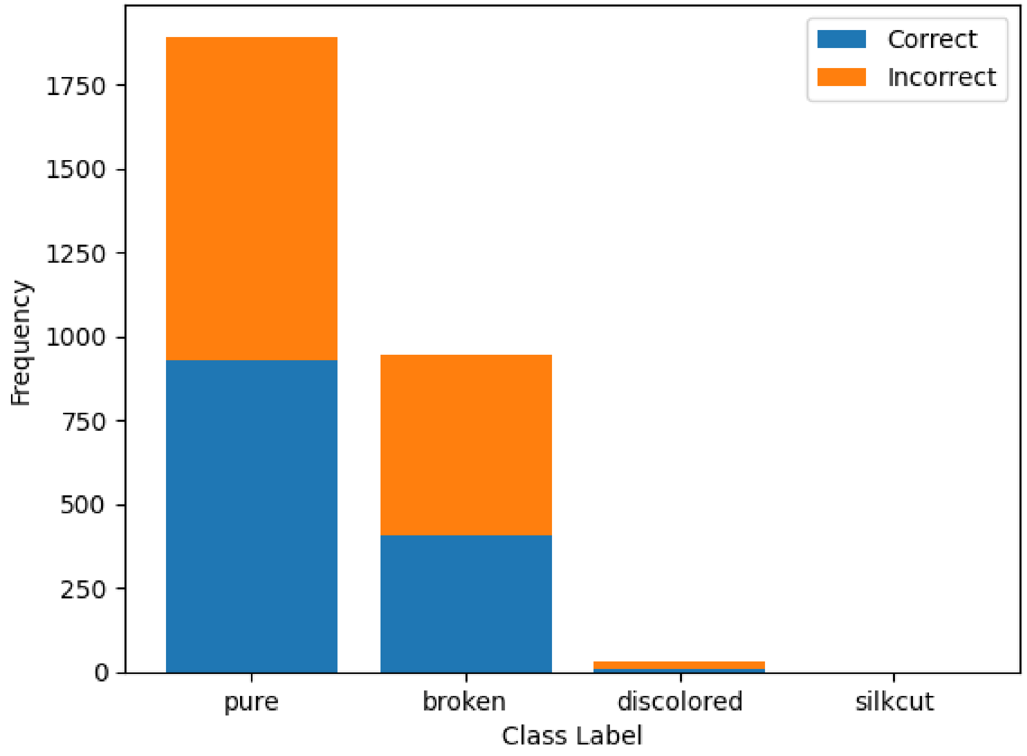
```
              precision    recall  f1-score   support

      broken     0.4900    0.5006    0.4952       883
  discolored     0.4246    0.1456    0.2168       522
        pure     0.5638    0.8455    0.6765      1165
     silkcut     0.4000    0.0076    0.0149       263

    accuracy                         0.5312      2833
   macro avg     0.4696    0.3748    0.3509      2833
weighted avg     0.5000    0.5312    0.4739      2833
```

Using SIFT in grid searches generally took an incredibly long amount of time unless cross validation folds were reduced to 2 or 3 and the number of hyperparameters to test was limited heavily. Due to multiplicative growth in time, testing additional hyperparameters as well as a wider range of values proved impossible with exhaustive grid search. For this result 2 fold cross validation was used against 3 hyperparameters, n_neighbors, n_components and n_clusters. The number of clusters relates to an intermediate feature processing step used to adjust the size of SIFT features meaningfully, where each SIFT feature is clustered using Kmeans clustering in order to eventually produce same-shape output from SIFT for classification.

The parameters which produced these results were:

- n_neighbors = 65,
- n_components = 10
- n_clusters = 30

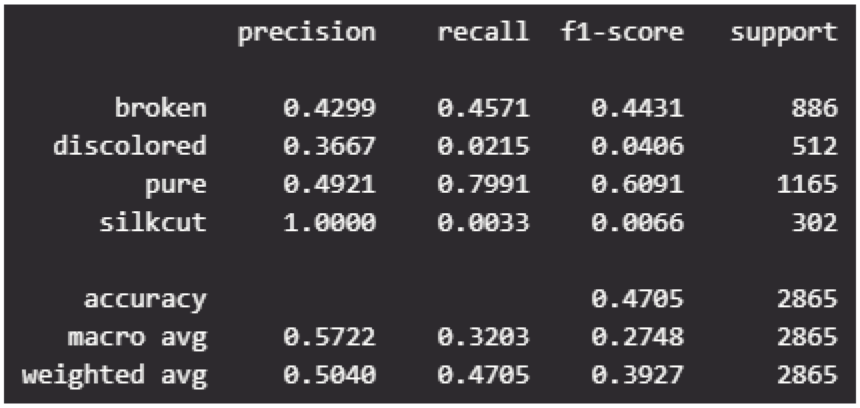
HOG + RandomForest, no Hyperparameter Tuning



Initial performance of the random forest model was better than the KNN model without any hyperparameter tuning, though inferior to both HOG + KNN and SIFT + KNN.

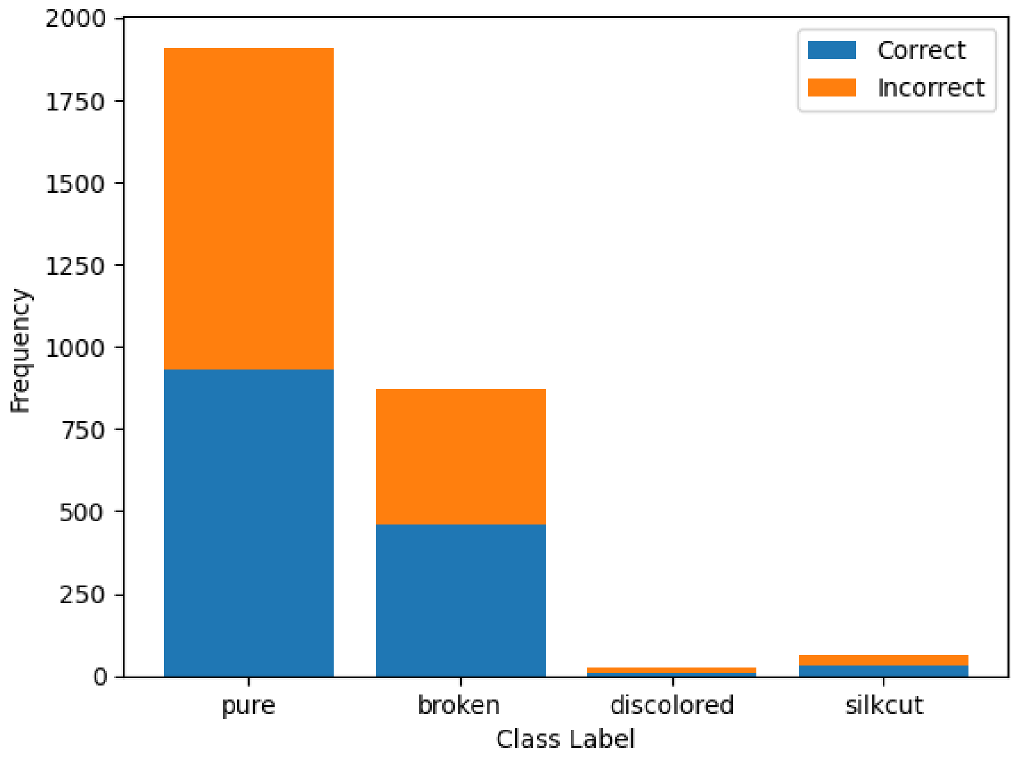
The parameters which produced these results were:

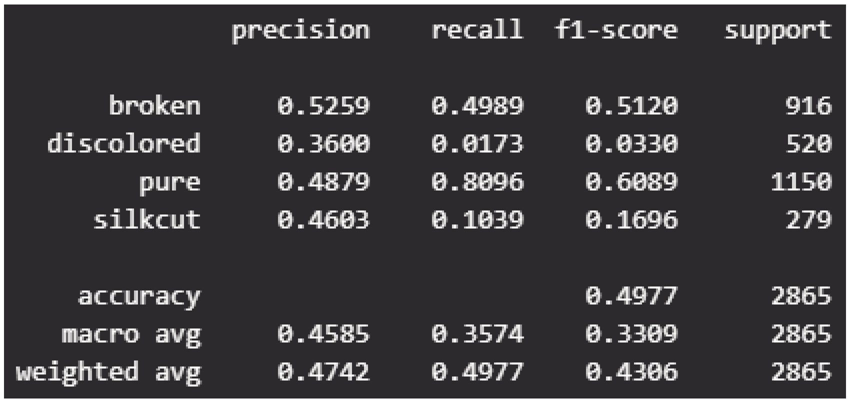
- default



```
              precision    recall  f1-score   support

      broken     0.4299    0.4571    0.4431       886
  discolored     0.3667    0.0215    0.0406       512
        pure     0.4921    0.7991    0.6091      1165
     silkcut     1.0000    0.0033    0.0066       302

    accuracy                         0.4705      2865
   macro avg     0.5722    0.3203    0.2748      2865
weighted avg     0.5040    0.4705    0.3927      2865
```

HOG + PCA + RandomForest with Hyperparameter Tuning





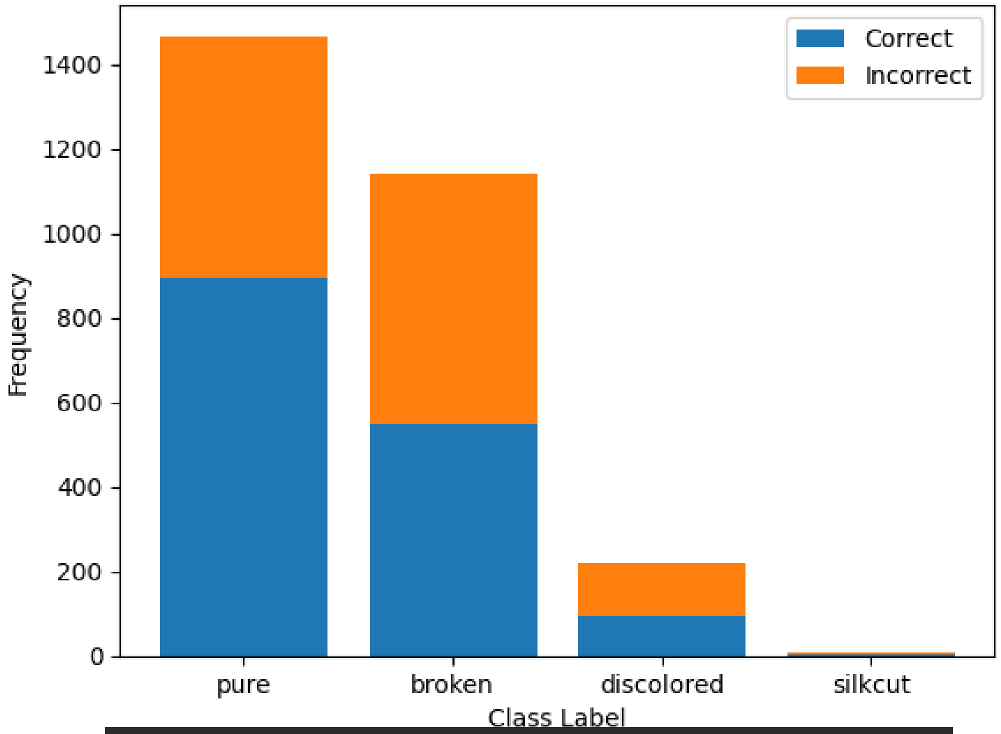
```
              precision    recall  f1-score   support

      broken     0.5259    0.4989    0.5120       916
  discolored     0.3600    0.0173    0.0330       520
        pure     0.4879    0.8096    0.6089      1150
     silkcut     0.4603    0.1039    0.1696       279

    accuracy                         0.4977      2865
   macro avg     0.4585    0.3574    0.3309      2865
weighted avg     0.4742    0.4977    0.4306      2865
```

After tuning hyperparameters I was unable to improve this combination beyond the performance of the same preprocessing with the KNN model with respect to accuracy, however the macro accuracy has improved by .005 suggesting it is comparable overall. I don't feel as though I was able to fully explore the potential of tuning the random forest model here, as I ran into some time constraint (or perhaps code inefficiency issues) when doing grid searches against the hyperparameters, though with some manual exploration this was the best result produced.

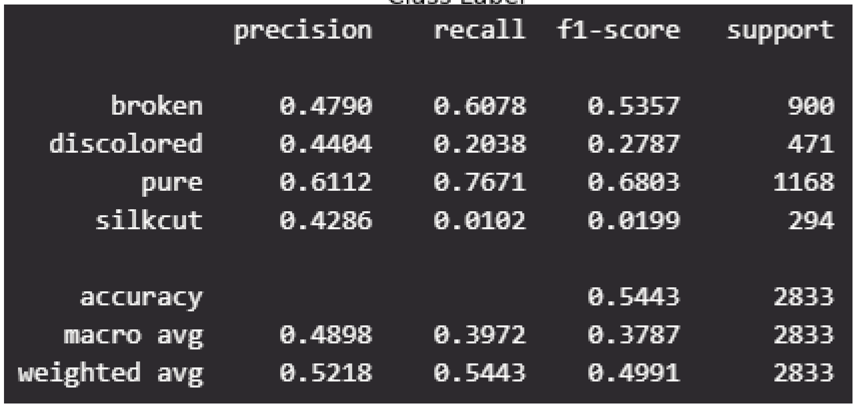
The parameters which produced these results were:

- n_estimators = 1000,
- n_components = 15
- default otherwise

SIFT + RandomForest with Hyperparameter Tuning



The macro average of this combination was the best of all feature extraction + preprocessing + model combinations I tried. At roughly .025 high macro average than the best performing KNN model I wouldn't say that the random forest model met my expectations in performance. However I do have some theories on why both models performed quite poorly which I will espouse on in the conclusion.



```
              precision    recall  f1-score   support

      broken     0.4790    0.6078    0.5357       900
  discolored     0.4404    0.2038    0.2787       471
        pure     0.6112    0.7671    0.6803      1168
     silkcut     0.4286    0.0102    0.0199       294

    accuracy                         0.5443      2833
   macro avg     0.4898    0.3972    0.3787      2833
weighted avg     0.5218    0.5443    0.4991      2833
```

The parameters which produced these results were:

- sigma = 0.5
- edgeThreshold = 15
- n_estimators = 1000,
- n_clusters = 170

Conclusion:

The results are not very impressive for this assignment, however I was still pleased with the amount of novel territory covered and amount of quality practice with the tools I was able to achieve. There were a few major oversights I didn't have time to address with this project which I believe would serve as a solid basis to further improve the performance of the models used thus far.

Given that the data set was rather unbalanced I belive that some means of addressing this fact would have been beneficial. Currently every performance graph shows roughly the same pattern of a strong majority of predictions for the 'pure' class and almost none for the 'silkcut' class. I think that the data set is sufficiently large that undersampling may be a valid method of improving the ability for models to differentiate key features of the classes. I could also use some means of oversampling too in order to increase the number of under-represented classes, particularly 'discolored' and 'silkcut' to balance the data set as well. Regardless of how I balanced the data set, I imagine there would be some performance improvement by taking this step in preprocessing.

A second change in preprocessing of the data I think which may be helpful is to focus on the importance of color for classification, as a whole class which represents 17.48% of the data–'discolored'--is likely to be more detectable when all color information preserved. The methods of feature extraction which I used may not be compatible with color images, which also suggests that other forms of feature extraction or no feature extraction at all may have been better choices. Further the HOG and SIFT methods were more well suited to detecting the shapes and forms of different objects than they were for picking up on subtle differences in similar objects from what I have come to understand, and thus I think testing other methods of preprocessing the image set would be fruitful.

Finally other models, and obviously neural networks would be wise to explore if I really wanted to see performance improvements in this task. From the start of this project I was not intent on getting maximum performance however, I was instead interested in exploring the space carefully, such that a foundation for future learning could be built.

Acknowledgements:

The help of OpenAI's ChatGPT was instrumental in exploring this novel space of machine learning. Rob Mulla's Kaggle competition provided context and data which is crucial in learning by doing with project.

References:

1. https://ai.googleblog.com/2023/03/palm-e-embodied-multimodal-language.html
2. https://openai.com/research/gpt-4
3. https://www.kaggle.com/competitions/kaggle-pog-series-s01e03/overview
4. https://www.youtube.com/watch?v=kSqxn6zGE0c
5. https://sbme-tutorials.github.io/2019/cv/notes/7_week7.html
6. https://www.youtube.com/playlist?list=PL2zRqk16wsdqXEMpHrc4Qnb5rA1Cylrhx
7. https://towardsdatascience.com/understanding-random-forest-58381e0602d2