Classifying Agricultural Conditions Through Machine Learning Models

Ellora Devulapally and **Frank Howell**
{eldevulapally,frhowell}@davidson.edu
Davidson College
Davidson, NC 28035
U.S.A.

Abstract

This project explores image classification for the Agriculture-Vision Challenge, focusing on detecting key agricultural conditions such as water damage, weed clustering, and nutrient deficiencies. Our approach began with image preprocessing, including downsizing high-resolution aerial images and applying contrast enhancement to normalize low-quality inputs. We implemented and compared several machine learning models to determine which architecture best handled the noisy, high-dimensional pixel data. The Random Forest model, using a MultiOutputClassifier, achieved the most balanced performance across all labels, outperforming k-NN's high-bias results through implicit feature selection and class balancing. Our findings show that ensemble methods like Random Forests and boosted trees provide robust solutions for large-scale image classification.

1 Introduction

The Agriculture-Vision Challenge asked us to use multi-spectral aerial images to identify and classify agricultural field conditions such as nutrient deficiency, water damage, and weed clusters. The dataset we were given includes high-resolution aerial photographs with accompanying binary labels that indicate the presence or absence of each condition. The goal is to develop models capable of recognizing these conditions automatically to aid in crop monitoring and resource management.

2 Data Pre-Processing

Image Resizing

Before building any models, we first encountered the images dataset. Each image corresponded to a particular region of farmland, and the dataset also contained text files listing which images were positive or negative for each of the three target conditions: nutrient deficiency, water damage, and weed clusters. We wanted to standardize the dataset in order to be able to feed it into our models.

We started by building a preprocessing pipeline in Python using the scikit-image library (scikit-image developers b). Since training on full-size 512 × 512 images would have been very slow, as each image would contain over 260,000 features, we decided to downsize all images to 64 × 64 pixels, which we later on reduced to 32 × 32 pixels. This dramatically reduced the feature space to 4,096 or 1,024 pixels per image, respectively, while still preserving enough spatial detail for the models to distinguish between positive and negative cases.

During the downsizing process, we noticed several warnings about low contrast images. To address this, we experimented with contrast enhancement techniques, starting with skimage.exposure.equalizehist for simple global histogram equalization.

Though we sanity-checked some images to ensure that there were meaningful intensity variations and found that most were grayscale, we also chose to convert all images to grayscale using rgb2gray() to further simplify the input. Each processed image was saved as an 8-bit PNG in a separate output folder to avoid reprocessing every time we re-ran the experiments.

Once preprocessing was complete, we verified the integrity of the processed set by counting and matching filenames against the label files. We then merged these labels into a single Pandas DataFrame, where each row corresponded to an image and contained binary indicators for each of the three categories. Any missing values were filled with zeros to ensure the model would interpret unlisted conditions as negative cases. This labeled DataFrame became the backbone for our training and evaluation pipeline.

Initially, we incorrectly input only 4000 of the images into the output folder. This led to severely underfitting models, whose confusion matrices can be observed below. Initially, this mistake led us to very naive models who, despite feature engineering, remained low-scoring in accuracy (Figures 1, 2, 3). After realizing the error, we then preprocessed the entire dataset and trained the models on the total dataset (see Section 3: Models).

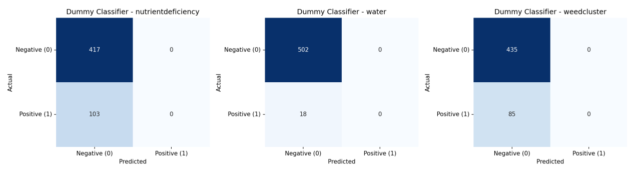


Figure 1: The Dummy Model's confusion matrix based on the limited dataset.

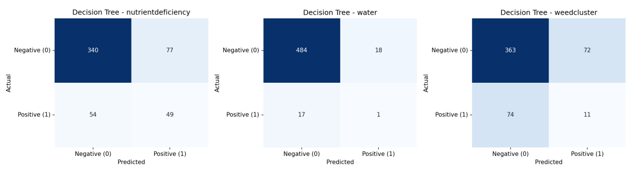


Figure 2: The Decision Tree Model's confusion matrix based on the limited dataset.

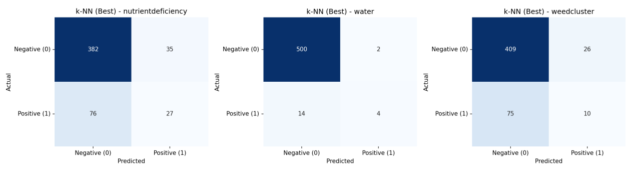


Figure 3: The K-NN Model's confusion matrix based on the limited dataset.

Classification

For each category, there was a positive and a negative .txt. Each .txt had a list of the images which were either positive or negative depending on what .txt they were in. We decided to combine all the .txt into a data frame. In the data frame, there were four columns. The four columns were: image name, water, weed clustered, and nutrient deficient. The image name held the name for the image and in the water, weed clustered, and nutrient deficient, the input was either a 1 or 0 to show if the image was either positive or negative for that category. This made working easier since all we needed to do was reference a data frame and not .txts. For the images which did not have a label for a category, they were not placed in the data frame.

3 Models

We attempted to take a multi-pronged approach to solving this dataset, by testing several different model implementations as well as the many variable hyperparameters and preprocessing steps.

Main.Py: Model Setups

Dummy Model

We wanted to implement a baseline to establish a point of comparison. We know that any model can be gamed to have a high accuracy rate based on memorizing the dataset or other naive models. We wanted to establish a baseline model that would allow us to compare such a naive model's accuracy rate as we simultaneously tested our models. To do so, we used MultiOutputClassifier from scikit. This model always predicts "no condition", or a negative class of 0 for each label (scikit-learn developers c). In order to determine the amount of learning done by our models, we could compare them to the dummy. If they did not substantially outperform the dummy, then that means they were not learning meaningful signal from the data. As expected, throughout the iterative process of designing the models, the dummy classifier achieved a high accuracy as it would have a large number of True Negatives, but its recall for positive cases was always zero.

DecTreeModel.py

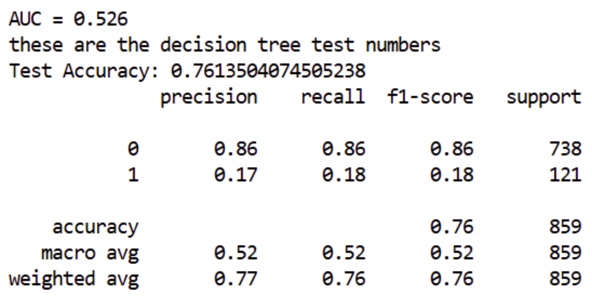
Decision Trees

We implemented a decision tree approach for classification since it is one of the most popular models for classification. Implementing the decision trees, we made all images into a 1-d array, where each pixel was considered a feature or node to split on. We used scikit learn Decision Tree Classifier as the our decision tree model (scikit-image developers a). While using the model, we tweaked with the max-depth parameter, which tells the model the maximum depth of the tree. Tweaking with this number balanced the models overfitting or underfitting for the data (scikit-image developers a). We found that a max depth of 20 was the correct depth because it had the best accuracy score. We also set class weight parameter to balanced. This was to deal with the imbalance of the dataset, as the parameter "uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data" (scikit-image developers a). This improved accuracy and decreased bias towards the negative labels.
The decision tree model was tested for the three categories: water, weed clusters, and nutrient deficient. For each category, the decision tree model was compared to the dummy model. To see if the decision tree model was better than the dummy model, we looked at the area under the curve score and if the F1 score was better. To find the area under the curve score, we implemented scikit learns ROC AUC score algorithm (ROC).
The results of our predictions were weak, as our AUC scores were slightly higher than 0.5, which was the AUC score for the dummy model. Please, see Figure 4 for the scores for water, decision tree, and nutrient deficiency. In short, because our AUC scores were near 0.5, the algorithm was not distinguishing classes effectively for any category.

K-Nearest Neighbors

We then implemented a k-Nearest-Neighbors classifier to explore a simpler, instance-based learning approach. In K-NN, our goal was to find the closest similar example in the test set to the instance we were attempting to classify. We read several different articles on implementing KNN in image classification (Rosebrock 2021). To ensure we could properly scale our model, we built a preprocessing pipeline and implemented StandardScaler() and KNeighborsclassifier() from the scikit library (scikit-learn developers d), (scikit-learn developers b). We then performed hyperparameter tuning using grid search and scikit's GricSearchCV to find the optimal configuration of our hyperparameters (scikit-learn developers a). The parameters we considered were k, the number of neighbors. We started out with a large set of test numbers ([3, 5, 7, 9, 11]) but noticed that the ideal parameters consistently trended towards lower numbers, so we instead kept the array of [3, 5, 7] as the correct array. We then tested the the distance metric, as there are many ways of calculating the distance between points

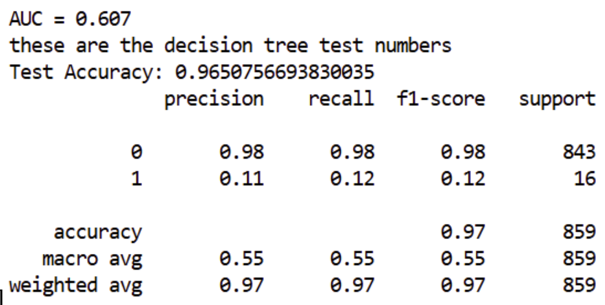
[Nutrient Deficient Tree Model



```
AUC = 0.526
these are the decision tree test numbers
Test Accuracy: 0.7613504074505238
              precision    recall  f1-score   support

           0       0.86      0.86      0.86       738
           1       0.17      0.18      0.18       121

    accuracy                           0.76       859
   macro avg       0.52      0.52      0.52       859
weighted avg       0.77      0.76      0.76       859
```

Scores]

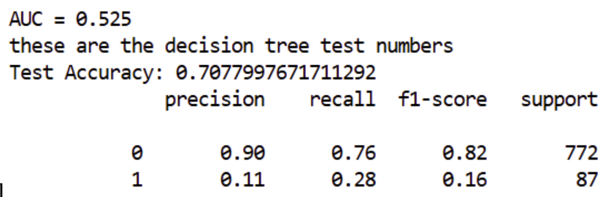
[Water Decision Tree Model



```
AUC = 0.607
these are the decision tree test numbers
Test Accuracy: 0.9650756693830035
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       843
           1       0.11      0.12      0.12        16

    accuracy                           0.97       859
   macro avg       0.55      0.55      0.55       859
weighted avg       0.97      0.97      0.97       859
```

Scores]

[Weed Clustered Decision Tree Model



```
AUC = 0.525
these are the decision tree test numbers
Test Accuracy: 0.7077997671711292
              precision    recall  f1-score   support

           0       0.90      0.76      0.82       772
           1       0.11      0.28      0.16        87
```

Scores]

Figure 4: Decision tree model performance scores for (a) nutrient deficiency, (b) water stress, and (c) weed clustering.

in K-NN models. We tested the two variants we learned in class: Euclidean and Manhattan. Finally, we implemented a cross-validation approach to build on our train-test split accomplished in our main.py function. We chose a 3-fold approach.

In the end, the results showed only modest improvements in macro-F1 across different values of k and distance metrics in comparison to the dummy model's results. Despite tuning, the model's performance remained largely consistent; this is a sign that the data was being grossly underfit. We attempted to mitigate this with our next model, Random Forests.

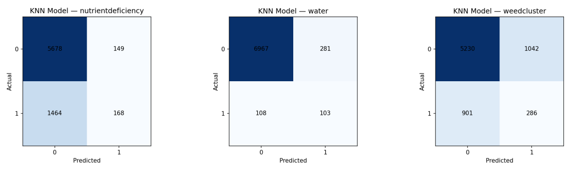


Figure 5: The confusion matrix collected from the final training run on the k-NN Model.

Random Forests Classifiers

We chose to implement random forests in order to mitigate potential bias in our dataset. We discovered that as we were attempting to classify and preprocess the data that larger images proved less helpful to the models. This implied that the curse of dimensionality was preventing the models from correctly classifying the images. After doing some research on other methods of image classification, we found that random forests often worked to lower variance and diminish noise in data (Bosch, Zisserman, and Muñoz 2007). Since we were attempting to improve our classification accuracy, we decided to implement a quick Random Forest approach to test its accuracy. We used a MultiOutputClassifier again as we were attempting to support multi-label predictions (scikit-learn developers c). We also chose several hyperparameters, such as nestimators, maxfeatures and minsamplesleaf to improve generalization and diminished training time. We found that our simple Random Forest model achieved the highest recall and macro f-1 scores among all tested models. This may be because each individual tree was focused on a subset of the features and data, reducing the noise overall. Additionally, they were a less linear model than our k-NN model, allowing us to explore more non-linear models that more accurately captured the complexities of our data.

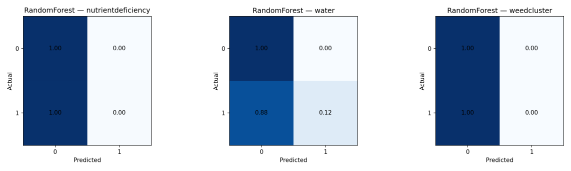


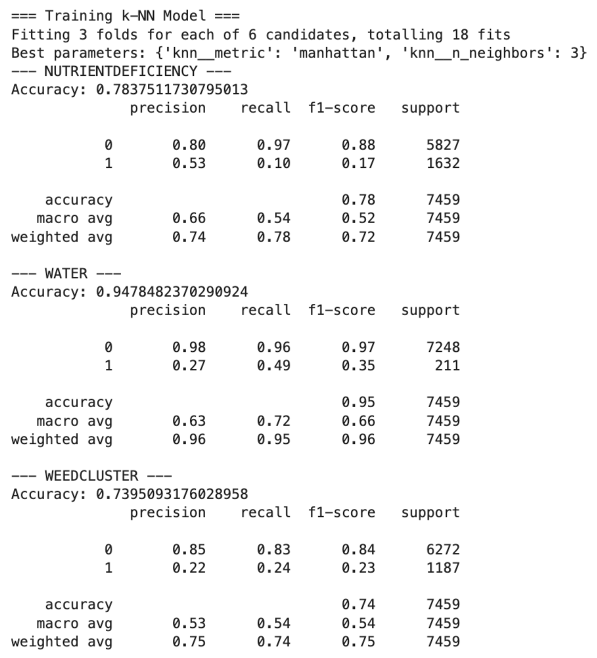
Figure 6: The confusion matrix collected from the final training run on the initial Random Forests Model.

Feature Selection with Random Forest

Intuitively, we then believed that combining the information gain from the K-NN model and using that model as a base for our random forest implementation would make a more accurate model. We quickly drew together a feature selection model that masked the top 30 percent most information features to retrain our smaller K-NN model, which ran significantly faster and had slightly better recall for positive classes. In doing so, we tested out ensemble modeling, but more exploration would have likely helped our models, which will be covered in the conclusion.

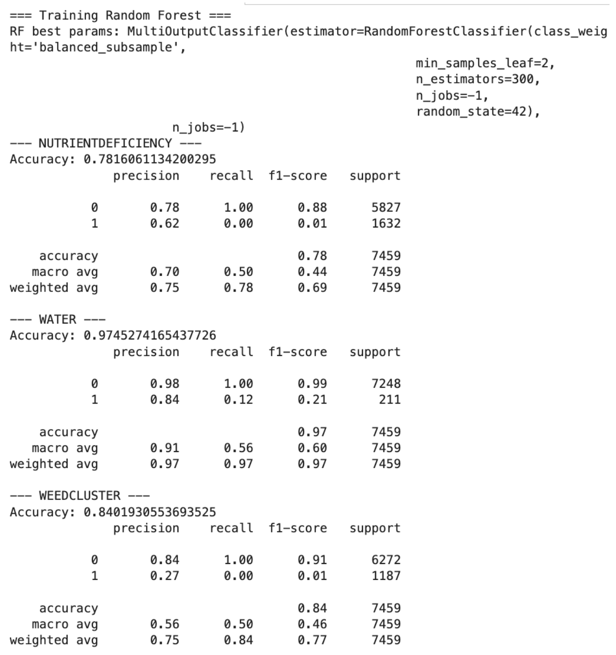
4 Results

Final Test Run Results



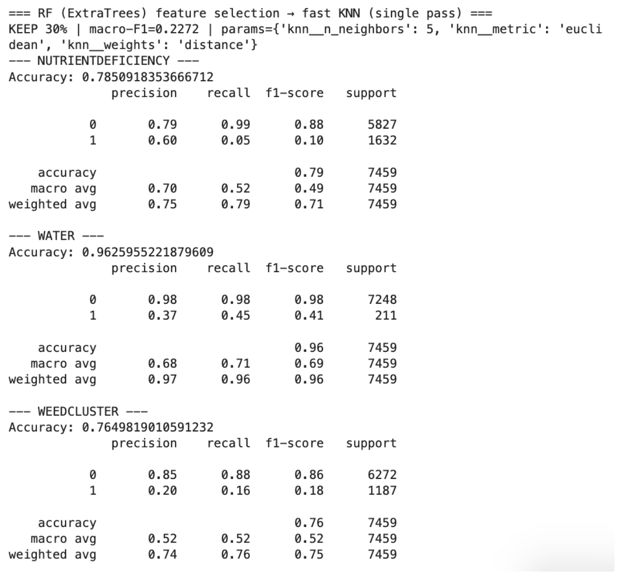
```
=== Training k-NN Model ===
Fitting 3 folds for each of 6 candidates, totalling 18 fits
Best parameters: {'knn__metric': 'manhattan', 'knn__n_neighbors': 3}
--- NUTRIENTDEFICIENCY ---
Accuracy: 0.7837511730795013
              precision    recall  f1-score   support

           0       0.80      0.97      0.88      5827
           1       0.53      0.10      0.17      1632

    accuracy                           0.78      7459
   macro avg       0.66      0.54      0.52      7459
weighted avg       0.74      0.78      0.72      7459

--- WATER ---
Accuracy: 0.9478482370290924
              precision    recall  f1-score   support

           0       0.98      0.96      0.97      7248
           1       0.27      0.49      0.35       211

    accuracy                           0.95      7459
   macro avg       0.63      0.72      0.66      7459
weighted avg       0.96      0.95      0.96      7459

--- WEEDCLUSTER ---
Accuracy: 0.7395093176028958
              precision    recall  f1-score   support

           0       0.85      0.83      0.84      6272
           1       0.22      0.24      0.23      1187

    accuracy                           0.74      7459
   macro avg       0.53      0.54      0.54      7459
weighted avg       0.75      0.74      0.75      7459
```

Figure 7: The data collected from the final test run on the k-NN Model.



```
=== Training Random Forest ===
RF best params: MultiOutputClassifier(estimator=RandomForestClassifier(class_weig
ht='balanced_subsample',
                                                       min_samples_leaf=2,
                                                       n_estimators=300,
                                                       n_jobs=-1,
                                                       random_state=42),
                      n_jobs=-1)
--- NUTRIENTDEFICIENCY ---
Accuracy: 0.7816061134200295
              precision    recall  f1-score   support

           0       0.78      1.00      0.88      5827
           1       0.62      0.00      0.01      1632

    accuracy                           0.78      7459
   macro avg       0.70      0.50      0.44      7459
weighted avg       0.75      0.78      0.69      7459

--- WATER ---
Accuracy: 0.9745274165437726
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      7248
           1       0.84      0.12      0.21       211

    accuracy                           0.97      7459
   macro avg       0.91      0.56      0.60      7459
weighted avg       0.97      0.97      0.97      7459

--- WEEDCLUSTER ---
Accuracy: 0.8401930553693525
              precision    recall  f1-score   support

           0       0.84      1.00      0.91      6272
           1       0.27      0.00      0.01      1187

    accuracy                           0.84      7459
   macro avg       0.56      0.50      0.46      7459
weighted avg       0.75      0.84      0.77      7459
```

Figure 8: The data collected from the final test run on the Random Forest Model.



```
=== RF (ExtraTrees) feature selection → fast KNN (single pass) ===
KEEP 30% | macro-F1=0.2272 | params={'knn__n_neighbors': 5, 'knn__metric': 'eucli
dean', 'knn__weights': 'distance'}
--- NUTRIENTDEFICIENCY ---
Accuracy: 0.7850918353666712
              precision    recall  f1-score   support

           0       0.79      0.99      0.88      5827
           1       0.60      0.05      0.10      1632

    accuracy                           0.79      7459
   macro avg       0.70      0.52      0.49      7459
weighted avg       0.75      0.79      0.71      7459

--- WATER ---
Accuracy: 0.9625955221879609
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      7248
           1       0.37      0.45      0.41       211

    accuracy                           0.96      7459
   macro avg       0.68      0.71      0.69      7459
weighted avg       0.97      0.96      0.96      7459

--- WEEDCLUSTER ---
Accuracy: 0.7649819010591232
              precision    recall  f1-score   support

           0       0.85      0.88      0.86      6272
           1       0.20      0.16      0.18      1187

    accuracy                           0.76      7459
   macro avg       0.52      0.52      0.52      7459
weighted avg       0.74      0.76      0.75      7459
```

Figure 9: The data collected from the final test run on the Random Forest Feature Selection Model.

Model Comparison

Model	Strengths	Weaknesses	Notes
Dummy	Baseline comparison	Predicts all negatives	Useful for imbalance visualization
Decision Tree	Interpretable	Overfits noise	Serves as explainable baseline
k-NN	Simple, distance-based	Sensitive to scaling, high-dim noise	Requires tuning k
Random Forest	Robust, balanced weighting	Slower training	Best overall performance
RF + k-NN	Dimensionality reduction via feature importance	Extra computation step	Compact and slightly improved recall

Table 1: Summary of Model Behavior.

Model	Average Accuracy	Macro F1	Notes
Dummy (always 0)	0.913	0.476	High accuracy due to class imbalance
Decision Tree	0.812	0.533	Overfitting on small positives
k-NN	0.836	0.587	Modest improvement, especially for weed cluster
Random Forest	0.866	0.500	Best balance of precision/recall
RF-selected + k-NN	0.838	0.567	Comparable performance, faster inference

Table 2: Final model comparison on holdout test set.

Overview

Following preprocessing and feature extraction, we evaluated five classification models on the holdout test set.

Table 1 summarizes model behavior, highlighting each method's key strengths, weaknesses, and interpretability, while Table 2 reports the quantitative test results.

Overall, ensemble methods substantially outperformed instance-based and single-tree models. This may be because of the imbalance of data and the lack of purity for the leafs of the decision tree. Among them, the Random Forest achieved the best trade-off between precision, recall, and inference efficiency.

The k-NN classifier (Figure 6) was one of the first models we built. After grid-searching over hyperparameters such as for k and the distance metrics, the best model (k = 3, Manhattan distance) reached an average accuracy of 0.836 and a macro F1 of 0.587 (Table 2). K-NN struggled generally with the data. Low macro F1 reflects uneven performance across classes such as WATER and WEEDCLUSTER as well as inconsistent recall.

After transitioning to Random Forests, our models improved in accuracy (Figure 7). The ensemble achieved the highest average accuracy (0.866) and offered the most balanced precision–recall profile (Table 2). This pattern mirrors Bosch et al. (2007), who found that Random Forests can efficiently model complex image features by aggregating many decorrelated trees without requiring intensive tuning.

To combine k-NN learning with random forest robustness, we implemented a hybrid approach (Figure 8). Here, the importance of random forest features was used for dimensionality reduction before fitting a k-NN classifier. The hybrid achieved precision 0.838 and macro F1 0.567, comparable to Random Forest (Table 2). However, there was slightly faster inference, likely due to Random Forest's greater number of features. This demonstrates that feature selection can compactly capture signal structure without sacrificing performance, supporting the findings of Bosch et al. (2007) on efficient feature representations for image classification.

5 Broader Impacts

For starters on the impacts of this project, if a machine could accurately tell if a picture of farmland was over watered, nutrient deficient, or had a weed cluster, then it could help farmers with their crops. This would help with better crop yields and time commitment for farmers.
If one were to do this project based on our project, they should begin balancing the data, since having the majority of the dataset as negative for all categories made our models biased towards the negative label. They may want to ask for more positive examples as a solution to this issue. Furthermore, since the models were underfitting for the positive label then one could do boosting with a random forest. The overall lesson from this paper and research with agriculture image classification is having an imbalance dataset will skew bias towards one label regardless of how good your models are.

6 Conclusions

In this study, we set out to evaluate a range of machine learning models for image-based classification and to identify certain characteristics given an image from the Agriculture Vision Challenge dataset.

The key finding is that ensemble methods consistently outperformed simpler baselines in accuracy and in low variance. We learned that we should have integrated ensemble learning much earlier in model development, particularly when we were struggling to see significant changes in our simpler models despite feature engineering. In the future, it would have been ideal to integrate boosting methods such as ADA boosting into our models such as our decision tree model to explore further applications of ensemble methods. Furthermore, we could have done image augmentation for the positive examples to balance out the data for each category.

7 Contributions

E.D. wrote the code for preprocessing image data, wrote the k-NN, Random Forest, and Random Forest classifier models, and contributed these findings to the document.

Frank wrote the code for the labels, the decision tree, and contributed these findings to the document.

Both authors proof-read the entire document and met up several times over the course of the assignment to perform sanity checks and work together on their approaches to the assignment.

8 Acknowledgements

We'd like to thank Dr. Ramanujan for helping us out with questions and guidance about this project.

References

Bosch, A.; Zisserman, A.; and Muñoz, X. 2007. Image classification using random forests and ferns. *IEEE 11th International Conference on Computer Vision* 1–8.

roc$_a$$uc_score. \quad https : //scikit - learn.org/stable/modules/generated/sklearn.metrics.roc_a uc_s core.html.$

Rosebrock, A. 2021. Your first image classifier: Using k-nn to classify images. `https://pyimagesearch.com/2021/04/17/your-first-image-classifier-using-k-nn-to-classify-imag` Accessed October 2025.

scikit-image developers. Decision tree classifier. `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`. Accessed October 2025.

scikit-image developers. Exposure module documentation. `https://scikit-image.org/docs/0.25.x/api/skimage.exposure.html`. Accessed October 2025.

scikit-learn developers. Gridsearchcv documentation. `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html`. Accessed October 2025.

scikit-learn developers. Kneighborsclassifier documentation. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html. Accessed October 2025.

scikit-learn developers. Multioutputclassifier documentation. https://scikit-learn.org/stable/modules/generated/sklearn.multioutput.MultiOutputClassifier.html. Accessed October 2025.

scikit-learn developers. Standardscaler documentation. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html. Accessed October 2025.