Machine Learning – Final Project

🥚 Broken Egg Classification 🥚

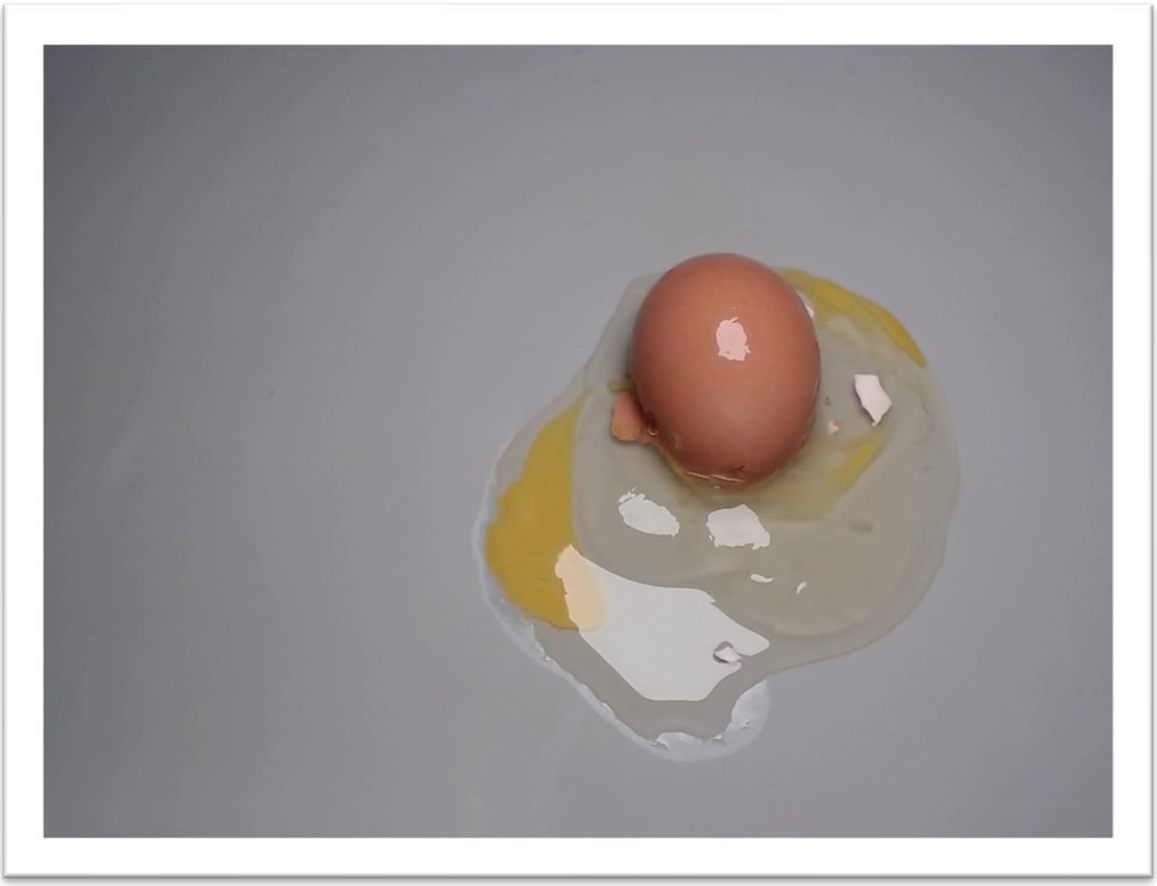
🍳 Dataset Analysis and Model Training 🍳

Alon Meshulam

Israel Gitler

GitHub Repository

https://github.com/AlonMesh/Broken-Egg-Classification

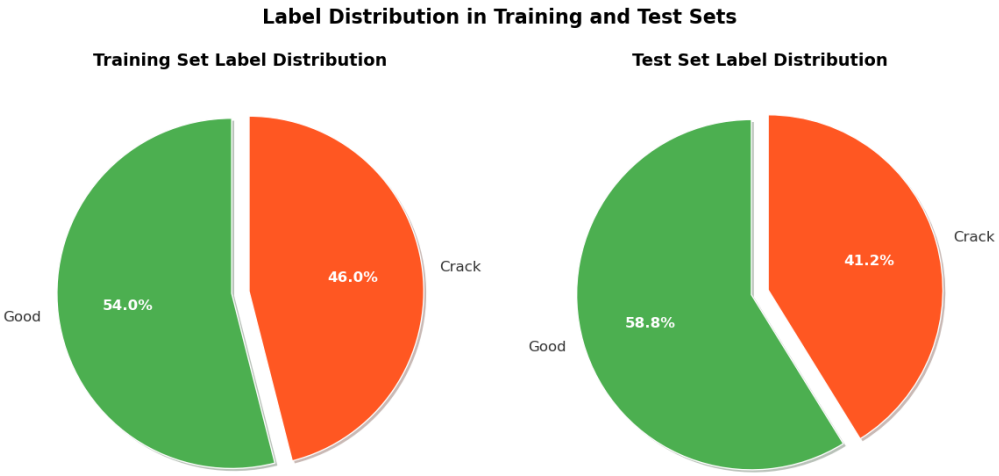


Dataset:

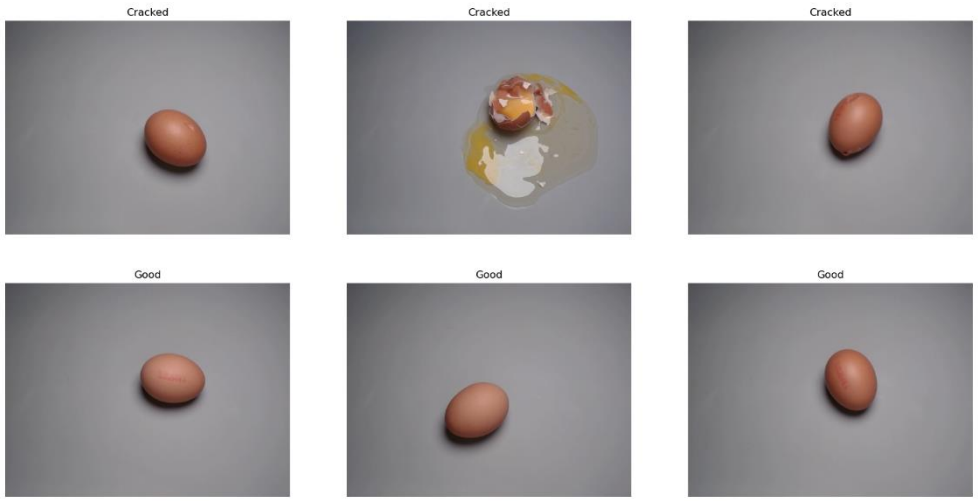
Our chosen dataset "Broken Eggs" comprises **369 images,** each with dimensions of 960 x 720 pixels, **depicting eggs.** The images are categorized into "good" and "cracked" within both the "test" (17 images) and "train" (352 images) directories. The "cracked" eggs exhibit **varying degrees of damage,** ranging from minor scratches to being completely destroyed. Additionally, there are images labeled as "empty," which we have chosen to exclude from our analysis.

You can get the dataset here:

- https://www.kaggle.com/datasets/frankpereny/broken-eggs



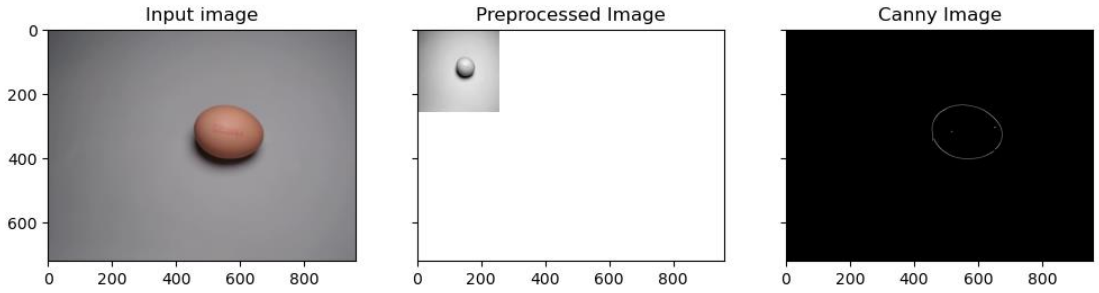
Visualize Sample Images



The main question we want to answer is whether machine learning models can improve the accuracy in classifying broken eggs compared to traditional methods (human eye in the factory).

Preprocessing:

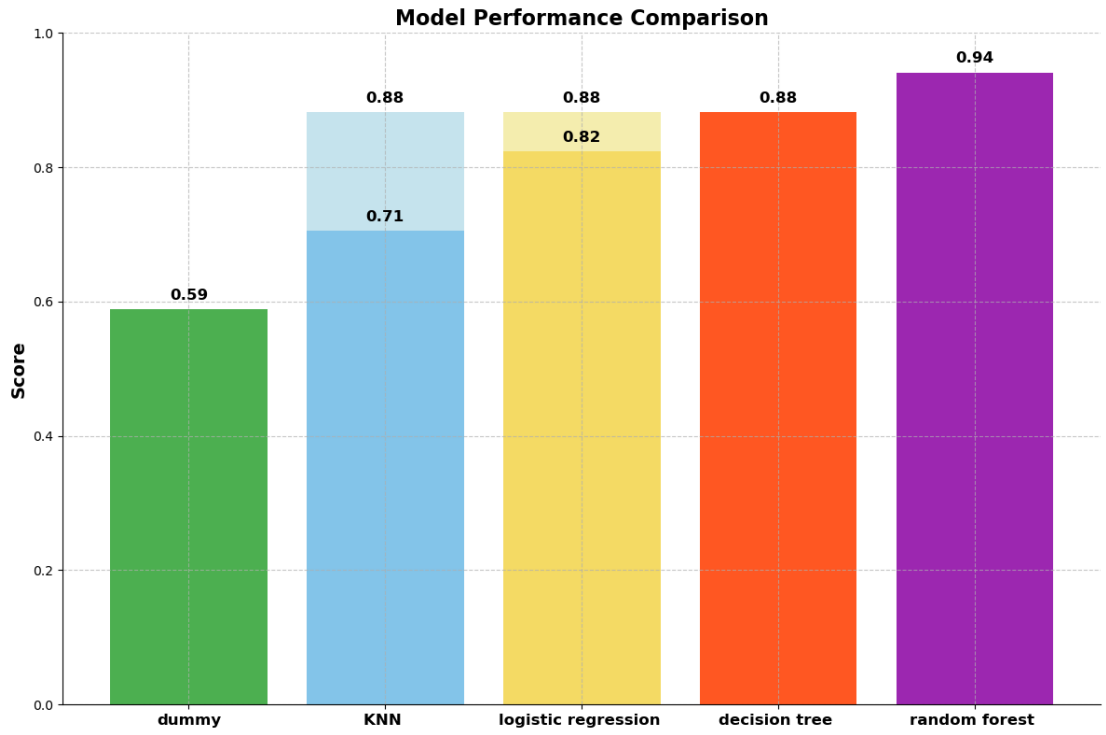
We applied several preprocessing steps to each image: converting to grayscale, resizing to 256x256 pixels, and normalizing pixel values to the range 0-1. Each image was then flattened into a vector.



We ran several machine learning algorithms:

1. **K-Nearest Neighbors (KNN)**: Achieved an initial performance of 0.71, which improved to 0.88 after hyperparameter tuning.
2. **Logistic Regression**: Initially performed at 0.82, increasing to 0.88 after hyperparameter tuning.
3. **Decision Tree**: Performed consistently at 0.88.
4. **Random Forest**: Achieved the highest performance at 0.94.

You can see the results in the following image:



To achieve better performance, we experimented with another method: applying the Canny Edge detector to each image. We then reran all the models with various parameter options (t1, t2) for the algorithm. Our best performance was achieved with parameters 55 and 110 on the Random Forest, reaching 100%. However, it's important to note that the test set only contained 17 images.

```
For parameters 40, 80 we got:
KNN-canny classifier score: 0.5882
Logistic Regression - canny classifier score: 0.7647
Decision Tree - canny classifier score: 0.9412
Random Forest - canny classifier score: 0.9412
========================================
For parameters 50, 100 we got:
KNN-canny classifier score: 0.5882
Logistic Regression - canny classifier score: 0.7059
Decision Tree - canny classifier score: 0.9412
Random Forest - canny classifier score: 0.8824
========================================
For parameters 45, 90 we got:
KNN-canny classifier score: 0.5882
Logistic Regression - canny classifier score: 0.7059
Decision Tree - canny classifier score: 0.9412
Random Forest - canny classifier score: 0.9412
========================================
For parameters 35, 70 we got:
KNN-canny classifier score: 0.5882
Logistic Regression - canny classifier score: 0.7059
Decision Tree - canny classifier score: 0.7647
Random Forest - canny classifier score: 0.9412
========================================
For parameters 50, 60 we got:
KNN-canny classifier score: 0.5882
Logistic Regression - canny classifier score: 0.7647
Decision Tree - canny classifier score: 0.8235
Random Forest - canny classifier score: 0.9412
========================================
For parameters 30, 60 we got:
KNN-canny classifier score: 0.5882
Logistic Regression - canny classifier score: 0.7059
Decision Tree - canny classifier score: 0.7647
Random Forest - canny classifier score: 0.8824
========================================
For parameters 55, 110 we got:
KNN-canny classifier score: 0.5882
Logistic Regression - canny classifier score: 0.7059
Decision Tree - canny classifier score: 0.8235
Random Forest - canny classifier score: 1.0000
========================================
For parameters 80, 120 we got:
KNN-canny classifier score: 0.5882
Logistic Regression - canny classifier score: 0.7059
Decision Tree - canny classifier score: 0.6471
Random Forest - canny classifier score: 0.8235
```

Techniques we tried:

Initially, we trained the models on the original RGB images at their original size. This resulted in heavy models with lower performance. To address this, we decided to preprocess the images, which yielded higher results.

We also tried using the Histogram of Oriented Gradients (HOG) technique, but it was extremely slow and provided poor performance. We assume this was because HOG detected unnecessary objects such as shadows and printed characters on the eggs.

On the other hand, the Canny Edge detector worked well. It is much lighter and, given our simple data (egg and background), performed effectively.

Challenges we faced:

Finding the optimal parameters for each model was challenging. To address this, we used RandomSearch, which improved our model performance. This was particularly necessary for the KNN and logistic regression models.

Finding the best parameters for the Canny algorithm was also difficult. We ran numerous parameter combinations until we found the best ones. Small parameter values detected non-existent edges, while large values missed many edges.

Future Work:

1. Acquiring more labeled data to expand the dataset from 369 images to thousands of images.
2. Applying deep learning techniques, such as Convolutional Neural Networks (CNN), which may provide better recognition than traditional ML tools.