Deep Learning for Soybean Classification

Binary Classification of Healthy vs Unhealthy Grains

Antonio Pilan
William Anselmo

Introduction to Machine Learning

Ribeirão Preto, 2025

Outline

Problem Context

Agricultural Challenge

- Brazil produces **40% of global soybean**
- Production losses exceed **20%** due to:
  - Diseases and pests
  - Climate change impacts
  - Quality control challenges
- Manual monitoring is **unfeasible at scale**
- Computer vision offers automation potential

Research Question: Can deep learning effectively classify healthy vs unhealthy soybean grains from proximal images?

Dataset Overview

Key Characteristics:

- Public dataset (Lin et al., 2023)
- Proximal grain images
- Automated segmentation
- Binary classification task

Class Distribution:

- Healthy: 1,201 samples (22%)
- Unhealthy: 4,312 samples (78%)
- Significant class imbalance!

Class	Count
Healthy	1,201
Unhealthy	4,312

Dataset Overview

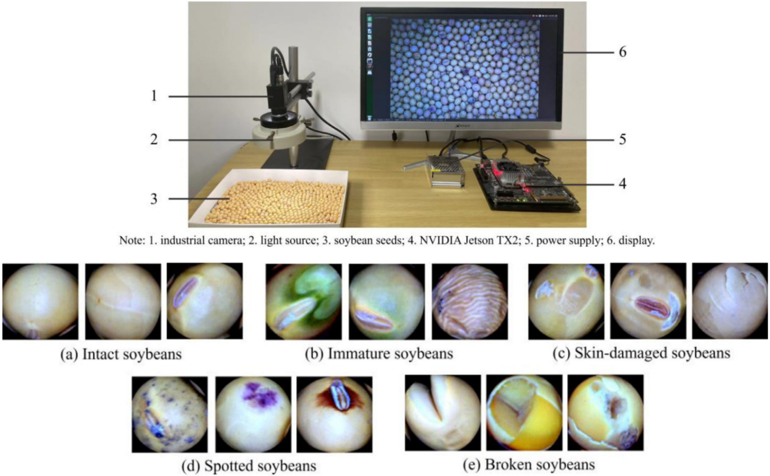


Figure: Data extraction process

Algorithm overview

Convolutional Neural Network (CNN):

```
model = Sequential([
    Conv2D(128, (3,3), activation='relu',
           input_shape=(224,224,1)),
    MaxPooling2D((2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Conv2D(32, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')  # Binary classification
])
```

Design Choices:

- Grayscale input (single channel)
- Progressive feature reduction
- MaxPooling for translation invariance
- Sigmoid output for probability

Training Strategy

Hyperparameters:

- Learning rate: 0.00001 (conservative approach)
- Batch size: 128
- Optimizer: Adam
- Class weights to handle imbalance

Evaluation Strategy:

- 5-fold cross-validation
- ROC curve analysis for threshold optimization
- Metrics: Accuracy, Precision, Recall, F1-score
- Early stopping to prevent overfitting

Data Split:

- Training: 40% (computational constraint)
- Testing: 60%

Addressing Class Imbalance

Challenge: 78% unhealthy vs 22% healthy samples

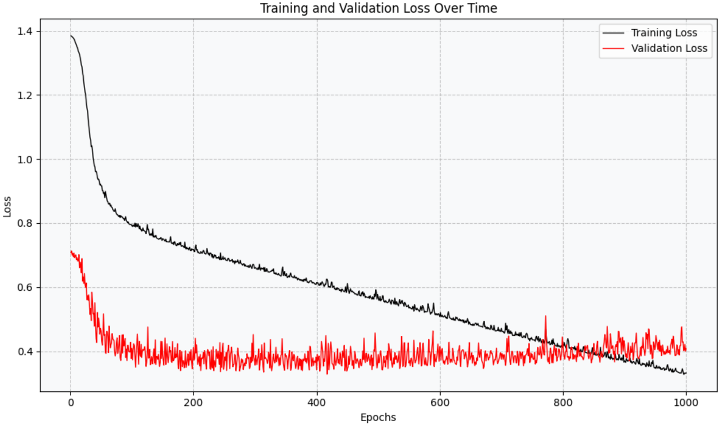
Solutions Implemented:

- **Class weighting:** Penalize misclassification of minority class more heavily
- **Threshold optimization:** Find optimal decision boundary using ROC curve
- **Appropriate metrics:** Focus F1-score rather than just accuracy. *Balance best Recall and Precision relation*

Results

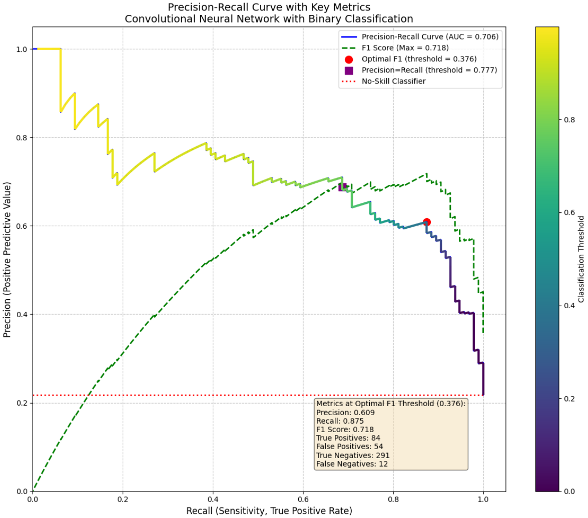
1-fold Simulation - Loss Over time

We are able to overfit, so a 5 fold training can give us a statistically better goal of epochs.



1-fold simulation - Decision Threshold

- Our final layer is a sigmoid function
- A threshold can be set for model decision-making



5-fold simulation - Training Dynamics

Overfitting Analysis:

- Training loss continuously decreases
- Validation loss shows more noise
- Early stopping around epoch 185

Cross-Validation Results:

Fold	F1 Score	Threshold	Epochs
1	0.689	0.649	174
2	0.692	0.684	188
3	0.687	0.729	184
4	0.744	0.580	185
5	0.683	0.706	197
Mean	**0.699**	**0.670**	**185**

Key Insight: Consistent performance across folds suggests stable learning.

ROC Analysis & Threshold Optimization

Key Findings:

- Optimal threshold: 67% probability
- AUC performance indicates discriminative ability
- Trade-off between precision and recall clearly visible

Decision Strategy:

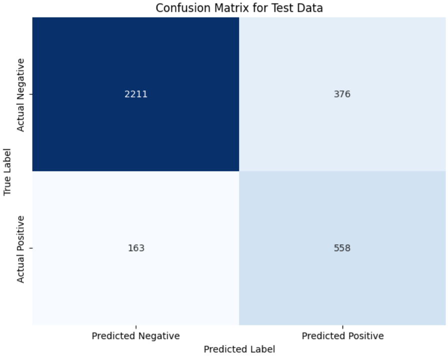
- Classify as "healthy" if $P(\text{healthy}) > 0.67$
- Maximizes F1-score (harmonic mean of precision/recall)
- Balances false positives vs false negatives

Business Implication: The threshold choice depends on whether it's more costly to reject good grains or accept bad ones.

Final Model

Final Model Performance

Test Results:

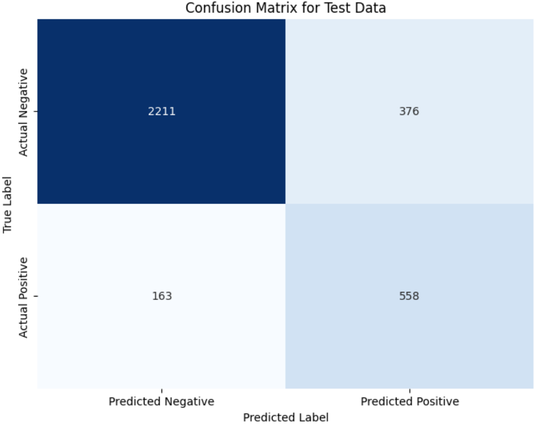


Interpretation:

- Model identifies most healthy grains (77% recall)
- But includes unhealthy grains in selection (60% precision)
- Better than random (50% baseline)
- Room for significant improvement

Practical Impact: Model would correctly identify healthy grains but would also incorrectly include a substantial number of unhealthy grains in the "healthy" classification.

Metric	Value
Accuracy	83.71%
Precision	59.74%
Recall	77.39%
F1-Score	67.43%



Critical Analysis

Limitations & Challenges

Current Limitations:

- **Grayscale only:** Missing color information (maturity indicators)
- **Limited training data:** Only 40% used due to computational constraints
- **Local minima uncertainty:** No guarantee of global optimization
- **Architecture simplicity:** Basic CNN without advanced techniques

Immediate Enhancements:

- **RGB channels:** Incorporate color information for maturity detection
- **Data augmentation:** Rotation, flipping, brightness adjustments
- **Advanced architectures:** ResNet, EfficientNet, Vision Transformers
- **Hyperparameter optimization:** Grid search, Bayesian optimization

Thank You!

Questions & Discussion

"Complex problems require robust solutions"

Attachments

- **Manifold Hypothesis:** High-dimensional data lies on lower-dimensional manifolds

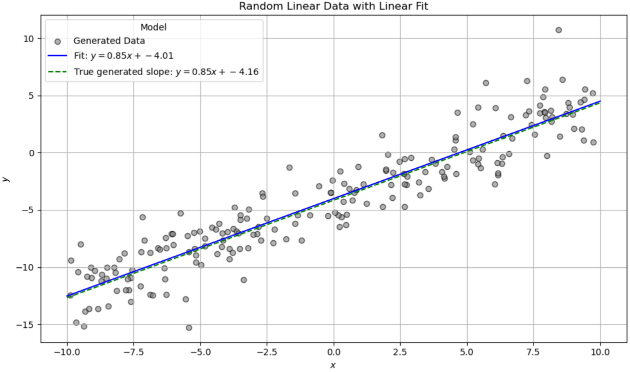


Figure: $y_{predict} = f(x) = \alpha x + \beta$

For real complex problems:

For complex problems, we stack multiple transformations:

$$f(x) = f_n(f_{n-1}(...f_1(x)))$$

Where each f_i represents a neural network layer with learnable parameters.

- **Core Concept:** "Deep learning is curve fitting, not magic" - François Chollet
- **Layer Structure:** Each layer performs space transformation

Training Steps

The training happens in 3 main steps

- Calculate **loss**
- Find its local **gradient**
- **Backpropagate** for $n-$layered system

Loss Function: Binary Cross-Entropy

Loss function is the machine learning metric we want to minimize

- Measures the "*distance*" between y_{real} and $y_{predicted}$
- Our goal is to minimize Loss to a optimal value

Why BCE for Binary Classification?

$$H_{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]$$

y_{true}	$p_{predicted} = 1$	$p_{predicted} = 0$
1	$H_{BCE} \to 0$	$H_{BCE} \to \infty$
0	$H_{BCE} \to \infty$	$H_{BCE} \to 0$

Key Properties:

- Penalizes confident wrong predictions heavily
- Differentiable for gradient-based optimization

Optimization: Gradient Descent & Backpropagation

The gradient of our loss function can lead us to a local minima

Gradient Descent:

$$\theta_{i+1} = \theta_i - \gamma \nabla H_{BCE}(\theta_i)$$

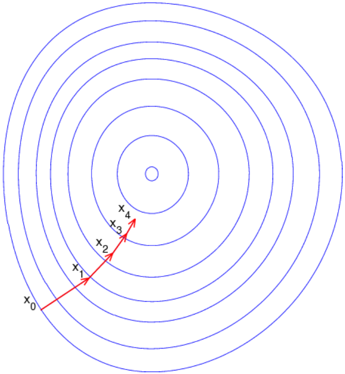


Figure: Gradient descent where each step leads us to a local minima

Optimization: Gradient Descent & Backpropagation

Gradient Descent:

$$\theta_{i+1} = \theta_i - \gamma \nabla H_{BCE}(\theta_i)$$

Global function $f(x)$

$$f(x) = f_n(f_{n-1}(\ldots f_1(x)))$$

Backpropagation Process:

1. Forward pass: compute predictions and loss
2. Backward pass: compute gradients using chain rule
3. Update parameters in direction of negative gradient
4. Repeat until convergence

Practical Implementation:

- TensorFlow/Keras handles automatic differentiation
- Adam optimizer for adaptive learning rates
- Batch processing for computational efficiency