RV College of Engineering®

Go, change the world

Department of AIML

Image segmentation with color detection of rice grains

Presented by

Aryan Sinha - 1RV22AI009
Ayush Chouhan - 1RV22AI012

Faculty Mentors:

Dr S Anupama Kumar,
Associate Professor,
Dept of AIML, RVCE

Somesh Nandi,
Assistant Professor,
Dept of AIML, RVCE

Problem Definition

Current methods for grain classification are manual or rely on basic image processing techniques, which face significant challenges. These include the inability to handle variations in grain color, lack of robust and automated systems for high accuracy, and sensitivity to environmental factors like lighting and background noise. To address these issues, the project aims to design a deep learning-based algorithm for a grain sorter machine that can accurately detect individual rice grains in an image, classify them as "good" or "bad" based on their color, and operate efficiently under varied conditions for real-time applications.

Object Detection

OpenCV

1. Image Preprocessing

The input image is converted to the HSV color space to better separate rice grains from the background. A mask is created by thresholding the HSV image, isolating the blue background. This mask is inverted to highlight the grains, focusing on the regions of interest. The grains are made distinct from the surrounding background for easier detection.

2. Contour Detection

The image is converted to grayscale to simplify contour detection. Binary thresholding enhances the difference between grains and background. The contours of the grains are detected, marking the boundaries of each grain. This prepares the image for bounding box detection.

3. Rice Grain Isolation

The mask created earlier is applied to isolate the rice grains from the background. Bitwise operations are used to retain only the relevant pixels corresponding to the grains. This step removes the background and highlights the grains. The isolated grains are now more prominent for further analysis.

Object Detection

OpenCV

4. Bounding Box Generation

- Contours detected from the mask are used to calculate bounding boxes around each rice grain. The algorithm uses cv2.boundingRect to determine the coordinates of each box. Green rectangles are drawn on the original image to visually locate the grains. This marks the area where each grain is located.

5. Final Image Visualization

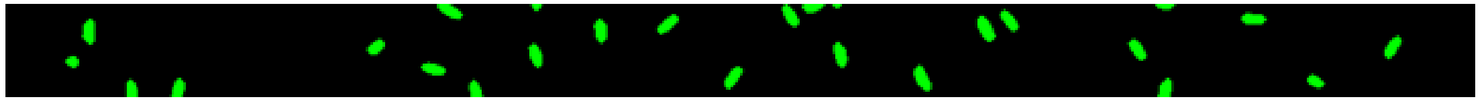
- Once bounding boxes are drawn, the image is displayed with the rice grains highlighted. The processed image is saved for further analysis or documentation. This allows for easy inspection of the detection results. It ensures that the bounding boxes are accurately placed around the grains.

Object Detection

OpenCV

Rice Grains Isolated

Contours



Original Image

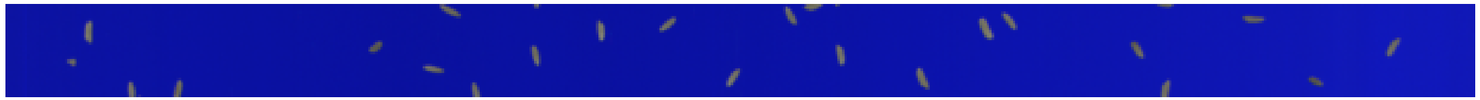
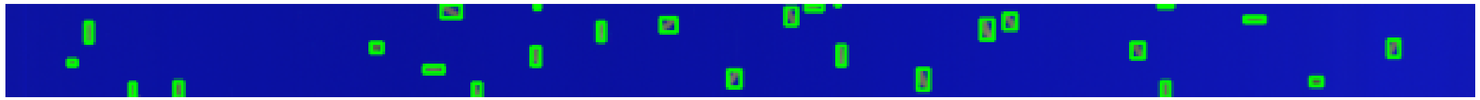


Image with Bounding Boxes



Object Detection

OpenCV

Why this Model is Not Suitable for FPGA Deployment

- Computational Complexity: Pixel-wise operations like RGB-to-HSV conversion, masking, and contour detection are resource-intensive for FPGA hardware.
- Memory and Libraries: Requires large intermediate data storage and relies on OpenCV, which isn't FPGA-friendly without significant customizations.

Issue: Close Rice Particles Merging into the Same Bounding Box

- Thresholding and Contours: Binary thresholding lacks precision to separate close grains, leading to overlapping contours being treated as one object.
- Lack of Validation: No shape or size-based constraints are applied, allowing closely packed grains to merge into single bounding boxes

Object Detection

Autoencoding and DBSCAN

Autoencoder Design and Training

- Two convolutional layers with ReLU activation and a kernel size of 3×3, followed by max-pooling to downsample the input dimensions are used for encoder and decoder. The final layer uses a sigmoid activation to output reconstructed images with pixel values normalized between 0 and 1. The autoencoder is compiled using the adam optimizer and binary_crossentropy as the loss function.

Reconstruction Error Calculation

- The absolute difference between the original and reconstructed images is computed for each pixel, forming an error map.

Error Analysis and Thresholding

- A threshold is determined based on the 95th percentile of the error values. Pixels with errors above this threshold are considered significant. A binary mask is generated to isolate high-error regions.

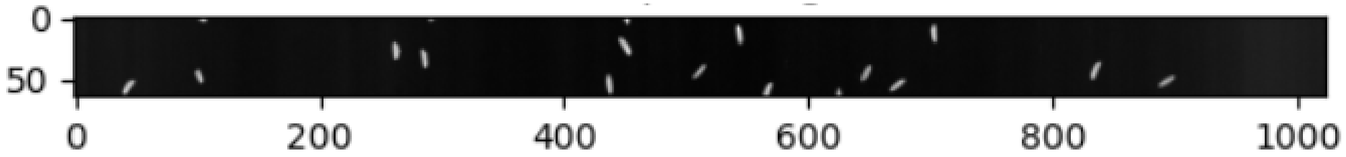
Object Detection

Autoencoding and DBSCAN

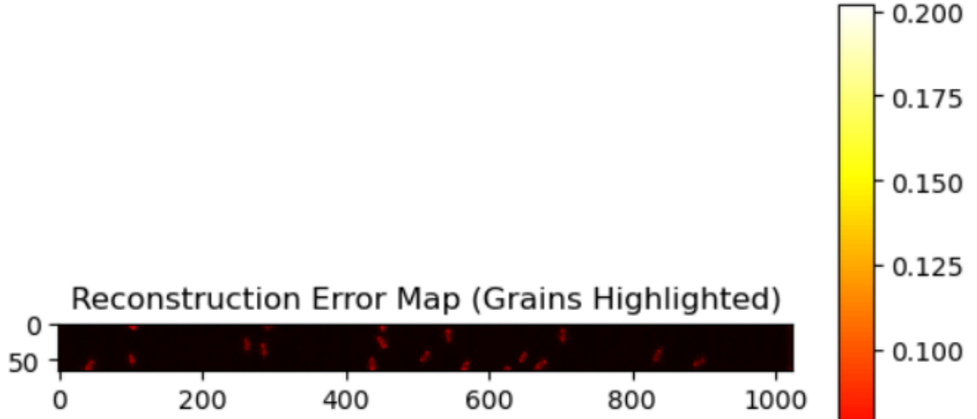
- **Clustering High-Error Regions -** The coordinates of high-error pixels are extracted from the binary mask and Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is applied on that.
- **Hyperparameters tuned:**
  - **eps: Defines the maximum distance for two points to be considered neighbors.**
  - **min_samples: Minimum number of points required to form a cluster.**
- **Bounding Box Generation**
  - Each cluster (ignoring noise) is analyzed to compute bounding boxes.
  - Bounding box coordinates are determined using the minimum and maximum x and y values of the points in the cluster.

Object Detection

- Grains isolated



- After autoencoding



- Bounding boxes

Object Detection

Autoencoding and DBSCAN

Why this model suitable ?

1. **Localized Error Detection**: Reconstruction error maps highlight specific regions with significant differences, enabling precise anomaly identification.
2. **Robust Clustering**: DBSCAN effectively identifies clusters of anomalies while filtering noise, making it suitable for irregular or scattered patterns.
3. **Scalability**: Autoencoder generalizes better to unseen datasets after training, as it captures a broad distribution of the data's characteristics.

Challenge

1. Latency may arise during clustering and error map generation in FPGA as autoencoders are computationally intensive
2. The algorithm can be optimized using quantization for FPGA deployment on low-power edge devices, enabling real-time detection in industrial settings. This can be done reducing numerical precision (e.g., to INT8 or fixed-point) to minimize computational complexity, memory usage, and power consumption while maintaining acceptable accuracy.

Methodology

Autoencoding and DBSCAN

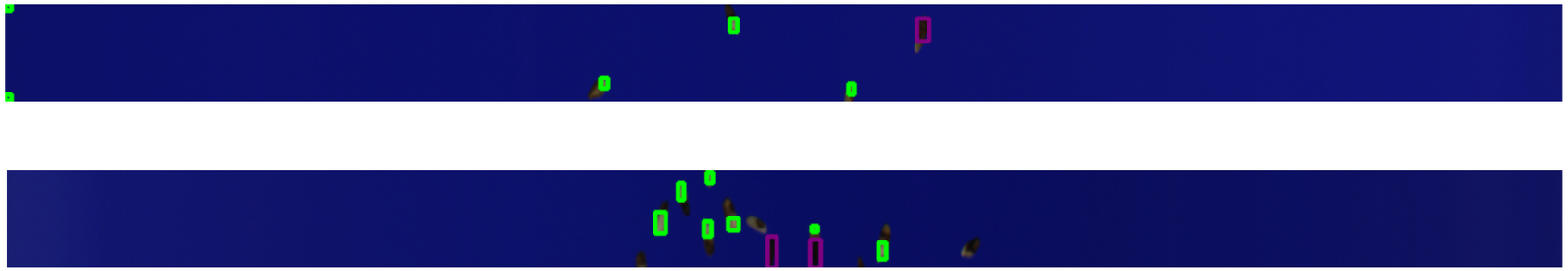
This creates a binary mask where pixels within the HSV range are white (255), and others are black (0).
This helps in separating good grains from bad ones based on color variations.

Morphological Operations (Optional)

We apply operations like cv2.morphologyEx() to remove noise and refine the segmented grains.

Feature Extraction

After segmentation, you may analyze properties like shape, texture, or size of the grains for classification.
You can use cv2.findContours() to extract individual grains.



Methodology

HLV Thresholding and Contour-Based Grain Detection

Overview

- This algorithm detects and classifies black or dark-tinted grains in an image using color-based thresholding and contour detection techniques. It processes images to filter out unwanted background noise and highlights defective grains for further analysis.

Algorithm Steps

Step 1: Image Preprocessing

- The input image is read using OpenCV (cv2.imread()).
- Converted from BGR to HSL (HLV) color space using cv2.cvtColor().
- HSL is used because it effectively separates intensity (L-channel) from color (H-channel), making color-based segmentation more accurate.

Step 2: Background Removal

- A blue mask is created using cv2.inRange() to detect and filter out the blue background.
- The mask is inverted (cv2.bitwise_not()) to retain only the rice grains while suppressing the background.

Step 3: Black Grain Detection

- A second mask is applied to detect dark or black grains using another cv2.inRange() function.
- This thresholding selects grains based on their low lightness (L-channel in HSL), isolating grains that appear black or have a dark tint.

Methodology

HLV Thresholding and Contour-Based Grain Detection

Step 4: Contour Detection and Bounding Boxes

- Contours of the detected black rice grains are found using cv2.findContours().
- Bounding boxes are drawn around these contours using cv2.rectangle() in red to highlight the dark or black rice grains.

Step 5: Result Visualization

- The final processed image with marked grains is displayed using Matplotlib (plt.imshow()).

Methodology

HLV Thresholding and Contour-Based Grain Detection

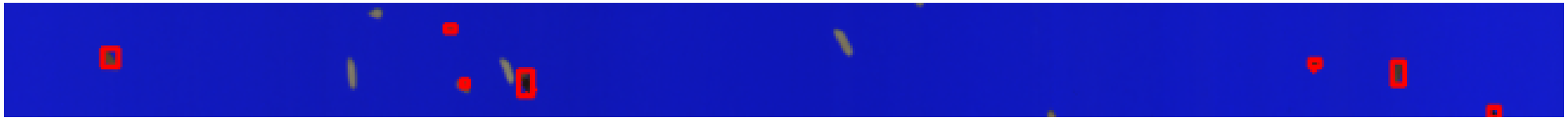
Real Images with bad rice particles

Marked the black rice particles

Methodology

HLV Thresholding and Contour-Based Grain Detection

Black or Dark-Tinted Rice Grains with Red Bounding Boxes



THANK YOU