# **METHODOLOGY FLOWCHART:**

A picture containing text, diagram, receipt

Description automatically generated

## **CODE EXPLANATION:**

* The processing of each frame of the video should begin in an endless loop.

A screen shot of a computer

Description automatically generated with low confidence

Figure 1(Process Frame Loop)

This line of code initiates an endless loop that will continue to process each frame of a video file until either the file is entirely read, or the infinite loop is explicitly terminated. The line of code that processes each frame is placed within the loop, where it will be iterated repeatedly until the loop is broken.

* Define a function called process\_frame(frame) that can count the grains and accepts a single frame as its input. This function should execute image processing.



A picture containing screenshot, text, font, black

Description automatically generated

Figure 2(image processing function)

This piece of code provides a method known as 'process\_frame ()' that is responsible for performing image processing operations on a single frame and accepts that frame as its input. In image processing, some of the processes that could be performed include converting the frame to grayscale, thresholding, detecting contours, performing morphological operations, and counting the number of objects. The count of the items that were found in the frame is what the function returns as its result.

The following is an explanation of the code:

'def process\_frame(frame):' is the first line of the function declaration. The word 'frame' refers to the argument that is being sent in.

A sequence of image processing procedures that are responsible for manipulating the input frame may be found within the function. The code that you gave does not display these actions in any way.

- 'count' is the name of the variable that is returned by the function. After various image processing procedures have been carried out on the frame, the number of objects that have been identified in the frame is stored in this variable.

* Convert the input frame to grayscale using the cv2.cvtColor() function, and then apply thresholding on the grayscale picture using the cv2.threshold() function. This should be done within the process\_frame () method.

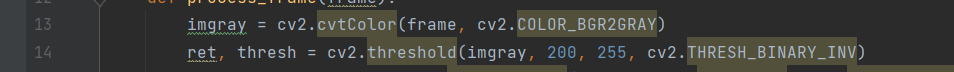


Figure 3(Threshold Images)

Using OpenCV's cvtColor function, this line of code converts the input colour picture frame to grayscale. It then applies a binary threshold to the grayscale image, which results in the creation of a binary image in which pixels that are above a particular threshold value are converted to white and pixels that are below that threshold value are set to black. In this scenario, the value of the threshold has been set to 200. The resultant binary picture is what's kept in the 'thresh' variable after it's been processed.

* Find the contours (the borders of objects) in the thresholded picture with the help of the cv2.findContours() function, and then draw the contours on the original colour image with the help of the cv2.drawContours() function.

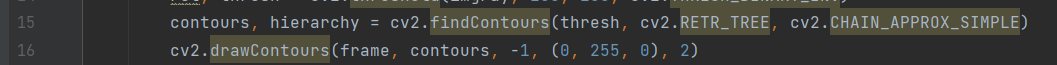


Figure 4(Drawing and Contours)

This piece of code searches for contours in a binary picture using the OpenCV library. Once the contours have been located, they are drawn on the original frame using a green colour and a thickness of 2. In the first line of code, the contours are located with the help of the find Contours () function. In the second line of code, the contours are drawn on the frame using the draw Contours () function.

* Find contours on the newly created picture after applying a morphological opening operation on the thresholded one using cv2.morphologyEx() with a 3x3 kernel.

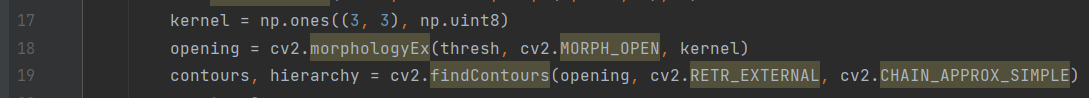


Figure 5(Morphological processes are used to smooth an image and find its edges)

Using the OpenCV package and the Python programming language, this piece of code analyses a binary picture to execute morphological opening operations and find contours. Morphological opening is a pre-processing approach that removes tiny items from a picture, while contour detection is a technique that is used to discover the contours of the remaining objects in the image. Morphological opening is a pre-processing technique. The contours are provided as a list of points, which may then be used for further analysis or for painting on the initial picture.

* Set the value of a variable count to zero, and then loop over each of the contours discovered in the step before this one. Calculate the area, perimeter, and aspect ratio of each contour, as well as the bounding rectangle for each one. Increase the value of the count variable by one for each contour, and then use the cv2.putText() function to write the count number on each grain.

A picture containing text, multimedia software, software, font

Description automatically generated

Figure 6(Counting and labelling grains)

This piece of code iterates through the contours that have been identified in a picture and calculates a variety of parameters for each contour, such as its area, perimeter, and aspect ratio. Additionally, it gives each contour its own distinct count number and shows that number on the grain in the picture that corresponds to that number.

* Give back the value of the count variable that was created by the process\_frame () function.



Figure 7(Grain Counting)

Following processing of an image using OpenCV functions, this piece of code provides the total count of grains that were identified inside the picture.

* Capture the next frame from the video file using the cap.read() function, then exit the while loop as soon as there are no more frames to capture.

A picture containing text, screenshot, font

Description automatically generated

Figure 8(Frame Retrieval from Video Capture)

Using the 'cap.read()' method of the OpenCV library, this code takes a frame from a video capture device or a video file. If the frame is read correctly, the method gives the true value 'True' and stores the frame in the variable 'frame'. If the frame can't be read, the code gives 'False' as a boolean value, and the 'break' command is used to get out of the loop.

* Cap.read() reads each frame from the video file, and cv2 changes the size of each frame to (700, 500). resize () before being sent to the count\_grains method process\_frame().

A picture containing text, screenshot, software, multimedia software

Description automatically generated

Figure 9(Workflow for Frame Processing)

This code reads frames from a video file, resizes each frame to (700, 500) pixels, counts the grains in the resized frame using the process\_frame () function, and shows the frame with the grain count as text using the cv2.putText() function. The result is then shown in a window called "result" with the help of cv2.imshow(). The loop keeps going until the user hits the 'q' key. When that happens, the programme drops the video file and closes the window by calling cap.release() and cv2.destroyAllWindows(), respectively.

## **TESTING RESULTS:**

The code uses OpenCV's VideoCapture() method to read a video file with the name "charran.avi". The process\_frame () method is then used to go through each frame of the movie and count the number of grains in each frame.

Each frame is turned into a grayscale picture using the process\_frame () function, which then applies a threshold to turn it into a binary image, finds contours, and then paints them on the original frame. Then, after using morphological opening to eliminate minute amounts of picture noise, it locates the exterior outlines. A bounding rectangle is then drawn around each contour once it has calculated its area, perimeter, and aspect ratio for each. The returned value is the number of grains found.

The code's main loop uses cap.read() to read each frame of the video and then sends it to the process\_frame() method to calculate the number of grains. Following that, it uses cv2.putText() to display the count on the frame and cv2.imshow() to display the outcome. The loop runs until the video has finished playing or the user clicks "q" to terminate it.

The process\_frame() function's ability to precisely count the number of grains in each frame of the movie is being tested throughout the code. By executing the code and monitoring the grain count shown on each frame of the movie, the testing results can be seen visually.

## **PICTURES EXPLANATION:**

It has been determined that there are fourteen grains in the figure 10.

A picture containing screenshot

Description automatically generated

Figure 10(final result1)

It has been determined that there are eight grains in the figure 11.

A picture containing screenshot, child art, colorfulness

Description automatically generated

Figure 11(final result2)

It has been determined that there are 17 grains in the figure 12.

A picture containing green

Description automatically generated

Figure 12(final result3)

It is essential to be aware that the precision of the algorithm used for counting grains may be affected by a wide range of variables, including the lighting conditions, the orientation and location of the camera, the size and form of the grains, and so on. Consequently, the count may not be entirely precise, but it may provide a rough approximation of the total number of grains in the frame.

## **PROBLEMS AND SOLUTIONS:**

Problem I ran across when working with this code was related to inaccurate counting: It is possible that the counting method is not precise and that it either overcounts or undercounts the grains depending on the features of the input video and the processing settings that are employed. Changing the settings and trying out a variety of counting methods might assist increase the accuracy of the counting.

The technique that helped me get rid of the erroneous counting is called "remove small contours," and it's as simple as that. The counting method may provide inaccurate results if it encounters outlines that are too small to accurately represent the grains being counted. You may get rid of contours like this by setting a minimum size threshold below which contours are disregarded. This will do the trick.

**TASK-2**

**METHODOLOGY OF THE AI SYSTEM**

A picture containing text, sketch, black and white, receipt

Description automatically generated

Figure 13 FLOW CHART OF COUNTING GRAINS

Popular deep learning algorithm YOLOv5 used for object detection tasks such as rice grain counting. Here is a general procedure for counting rice with YOLOv5:

Collect and label data: assemble a large dataset of images of rice grains and manually label each image to indicate where the rice grains are located.

Separate the dataset into training and validation sets. The labelled dataset should be divided into a training set and a validation set.

Train the model: Train a YOLOv5 model using a deep learning framework such as PyTorch on the training set. Based on the labelled data, the model will discover how to detect rice grains in images.

Verify the model: Use the validation set to evaluate the trained model's performance. Modify the model's parameters as necessary to enhance its efficacy.

Test the model by using the trained model to detect and count rice grains in new images. You can use techniques like non-maximum suppression to eliminate redundant detections and enhance counting precision.

Improve the model: As necessary, collect additional data, relabel the data, and retrain the model to increase its accuracy.

Once the model is accurate enough for your requirements, it can be deployed in a production environment to automatically count rice grains in real-time.

**DATASET CREATION**

Robo flow is an online platform that enables users to generate code for training machine learning models on custom datasets by uploading and annotating images. Once the dataset has been generated, it can be utilised with various computer vision frameworks, such as YOLOv5, to train and deploy models.

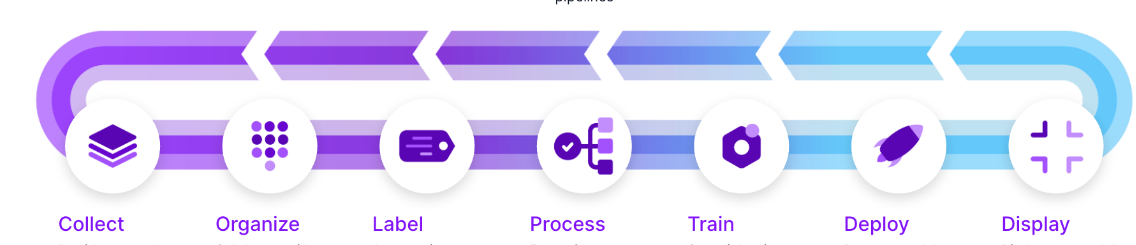


Figure 14 total process of dataset creation

Follow these steps to generate a customised dataset using Roboflow:

Create an account for free on the Rob flow website.

ADDING DATA

Click the "Datasets" tab at the top of the screen and then choose "Create New Dataset."

Name the dataset and choose "Object Detection" as the task type.

Select "Create from Scratch" from the menu.

Upload the video mp4 of rice grains for use in your dataset, click the "Add Images" icon.

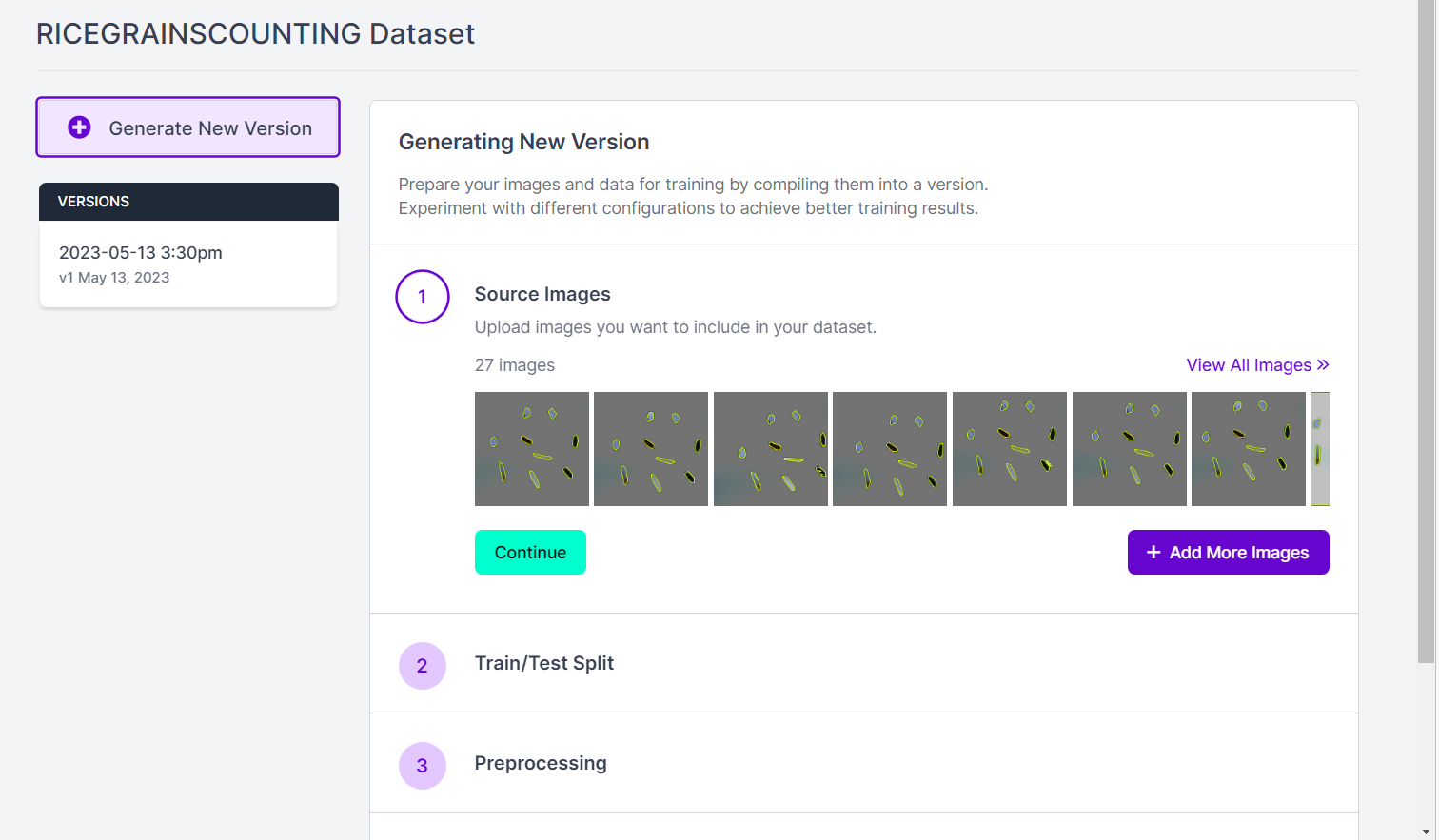


Figure 15 uploading the video

Once the images have been uploaded, you can designate the rice grains using Roboflow's built-in tools. Select an image and then click the "Annotate" icon to accomplish this.

Utilise the bounding box tool on the annotation screen to outline each rice particle in the image. You may also add a label for each rice particle,

A screenshot of a computer

Description automatically generated

Figure 16 annotating the video

Apply the annotation procedure to every image in your dataset.

Once all of your images have been annotated, you can export the dataset by selecting the "Export" button.

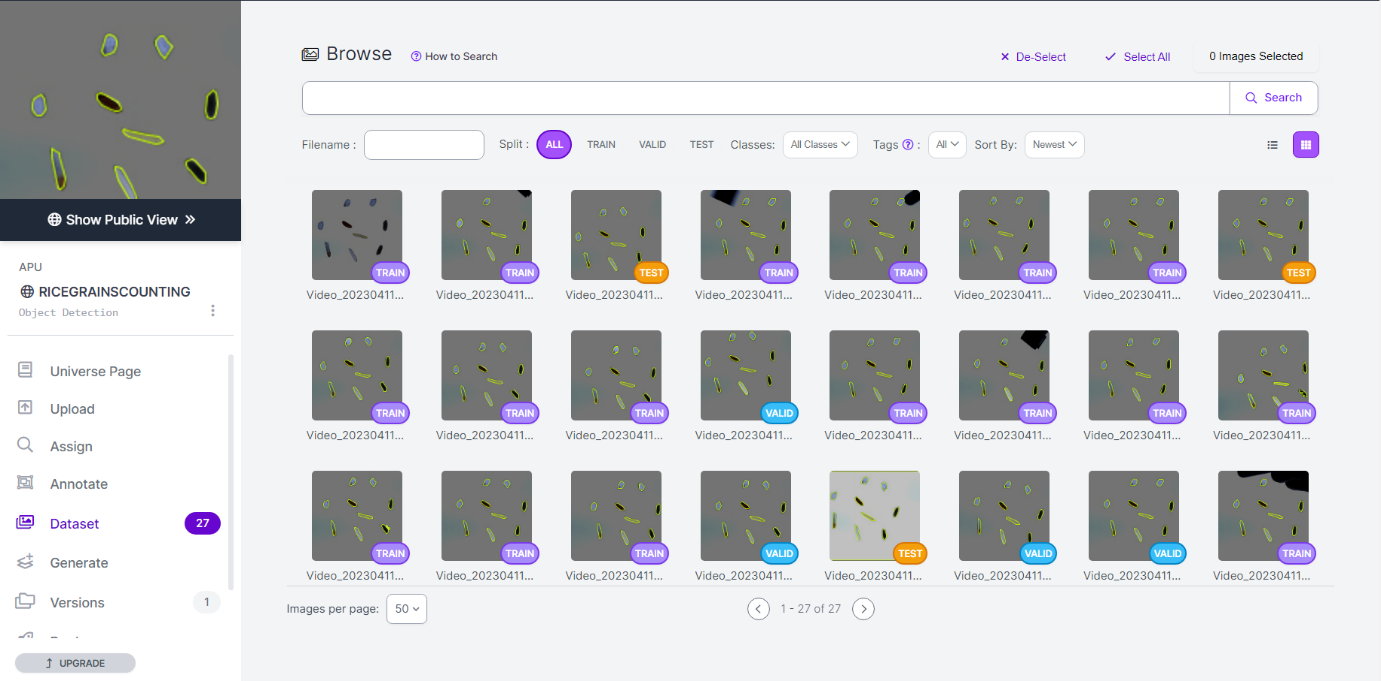


Figure 17 creating the dataset

Select the desired export format for the dataset, such as YOLOv5 or TensorFlow, and then select the "Export" button.

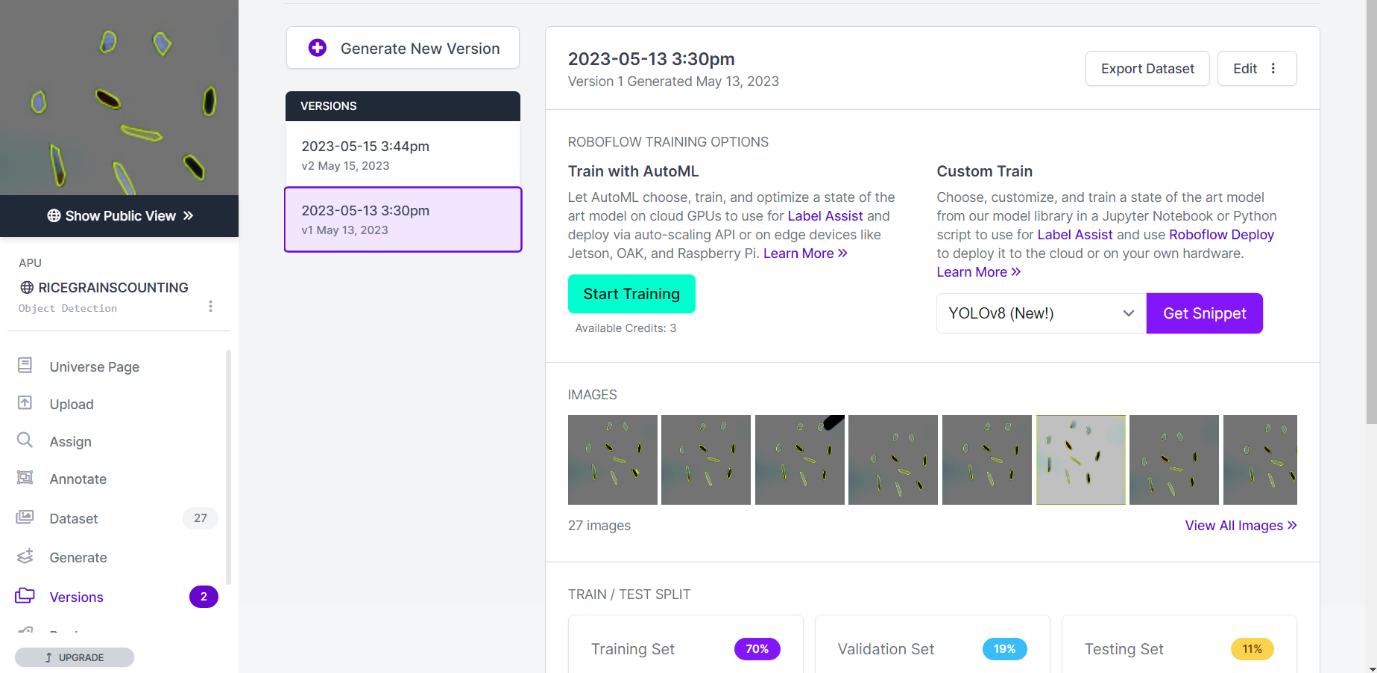


Figure 18 generating the dataset.

Download the exported dataset and use it to train your artificial intelligence model to count rice grains.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 19 exporting the dataset.

**PROGRAM CONFIGURATION**

There is requirement.txt we need istall in yolov5 type

Pip install -r requirement.txt command

A screen shot of a computer

Description automatically generated with medium confidence

Figure 20 reqirements for yolo

**Modify the yolo v5 parameters.**

According to the training requirements we need to modify the parameters of the train file. These are their places we need to change in the train.py file.

A screen shot of a computer screen

Description automatically generated with low confidence

Figure 21 train parameters

In the 1st parameter yolo 5s is a network model , s is the most basic and fastest to run so I choose that

after that need to modify the number of categories in the corresponding yaml

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 22 yolo5s parameters

Next, we need to keep the data set folder which we created from the rob flow and in that data.yaml file we need to change the path of the respective train, val , and test paths

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 23 data.yaml parameters

**START TRAINING**

After changing the train.py parameters we need to run the train.py file this how the training process looks like

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 24 training

The whole training process will takes some time , after the completion of the training process the results will be produced and save in train file in the run folder

A picture containing text, screenshot

Description automatically generated

After completion of the training the results will be saved in the respective directory which Is produces in the above picture

**TRAINING RESULTS**

After the results saved in the folder this is how all the results looks like in the folder

A screenshot of a computer

Description automatically generated with medium confidence

Figure 25 training results

The best.pt is the best results which will saved in the weights folder after this we need to change the command line parameters in the detect.py file.

**START DETECT.PY**

After changing the parameters in the detect.py file we need to run the detect.py file

A screen shot of a computer program

Description automatically generated with low confidence

Figure 26 detect parameters.

The results will be generated and saved in the below path.

A screen shot of a computer

Description automatically generated with medium confidence

Figure 27 detect results

After that we need to run the code which written detect the count of the rice using ai yolov5

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 28 count file using yolov5 parameters

We need to change the parameters in the loading t the YOLOv5 model is being loaded using the `torch.hub.load()` function from the `ultralytics/yolov5` repository. The model being loaded is a custom model, which means it has been trained on a custom dataset specific to your project.

The `torch.hub.load()` function allows you to load pre-trained models from various repositories. In this case, it loads the YOLOv5 model from the `ultralytics/yolov5` repository.

The function takes two arguments: the repository name (`ultralytics/yolov5`) and the model’s name (`custom`). The third argument is the path to the weights file (`best.pt`) of the trained YOLOv5 model. Make sure to provide the correct path to the `best.pt` file, which should contain the weights of the best-performing model during training.

Once the model is loaded, you can use it for various tasks, such as making predictions on images or videos. The `model` object will have methods available for object detection and other related tasks. Refer to the YOLOv5 documentation or the `ultralytics/yolov5` repository for detailed usage instructions and examples.

A screenshot of a cell phone

Description automatically generated with medium confidence

Figure 29 final result of counting using yolov5

This is how the result of the total rice grains looks like using AI yoloV5