

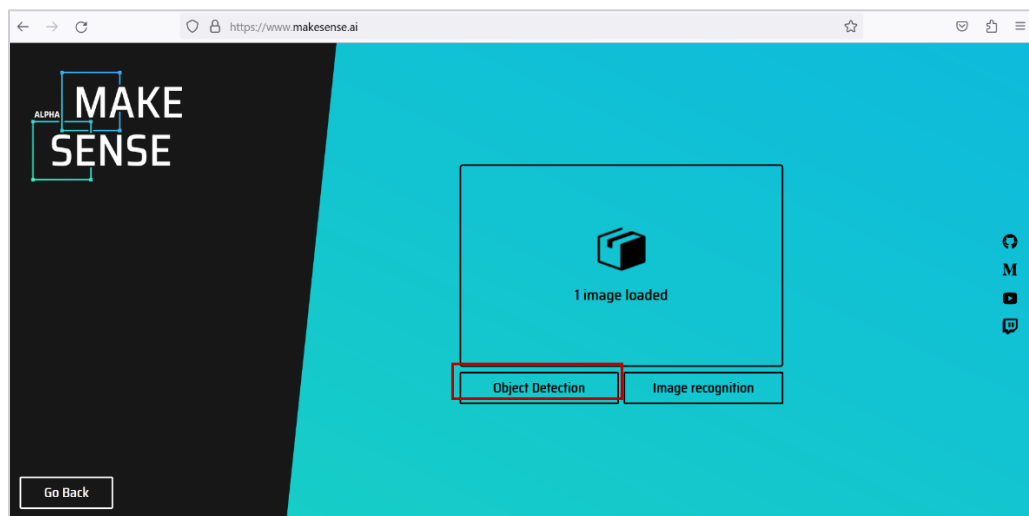
Defect detection process using AI-based object detection algorithm:

The objective of this project/ proof of concept is to detect different types of defects on a mechanical element (gear) using a detection algorithm. We have used a SSD MobileNet V2 pretrained on COCO dataset for this purpose. The following are the steps outlining the overall training and detection process:

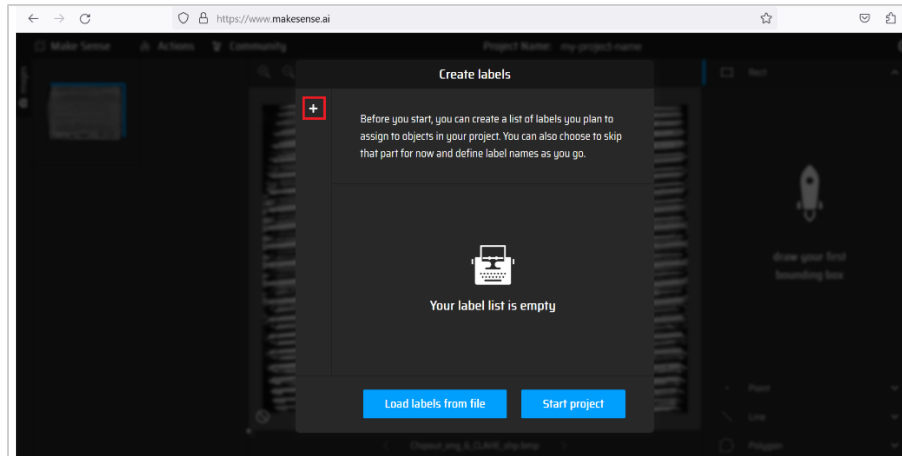
1. The dataset comprises 588 images belonging to 'good' class, 250 images belonging to 'chipout' class and 252 images belonging to 'damaged' class.
2. The dataset was split into 80 % - 20 %, where 80 % is used as training data and 20 % is used as test data.
3. The training data comprised 500 'good' class, 213 'chipout' class and 214 'damaged' class images.
4. The data (training and test) was preprocessed using the following steps:
 - i. Cropping using Canny edge detection, Morphological operations and Gaussian blur
 - ii. CLAHE (Contrast Limited Adaptive Histogram Equalization)
 - iii. Unsharp Masking

The preprocessing steps were implemented in order to enhance the feature effects at a local level.

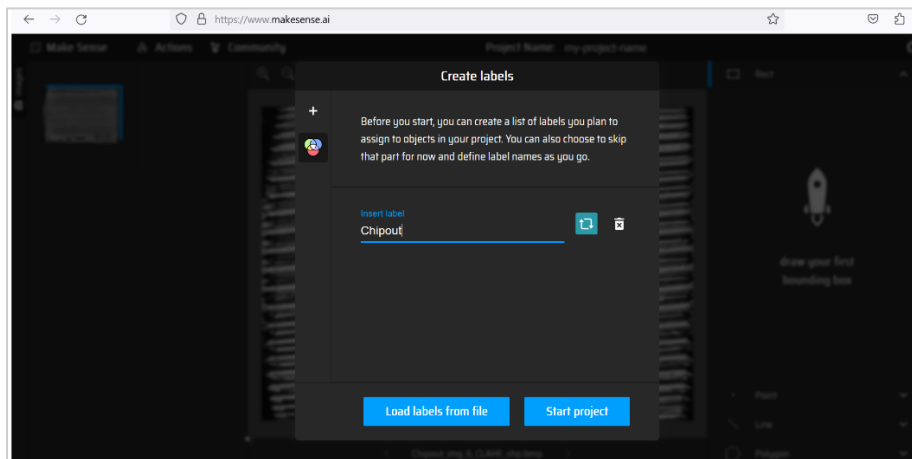
5. The preprocessed data was annotated using Makesense.ai tool, which is an open-source web application. The annotations can also be performed using other open-source tools namely, labellingm or LabelStudio



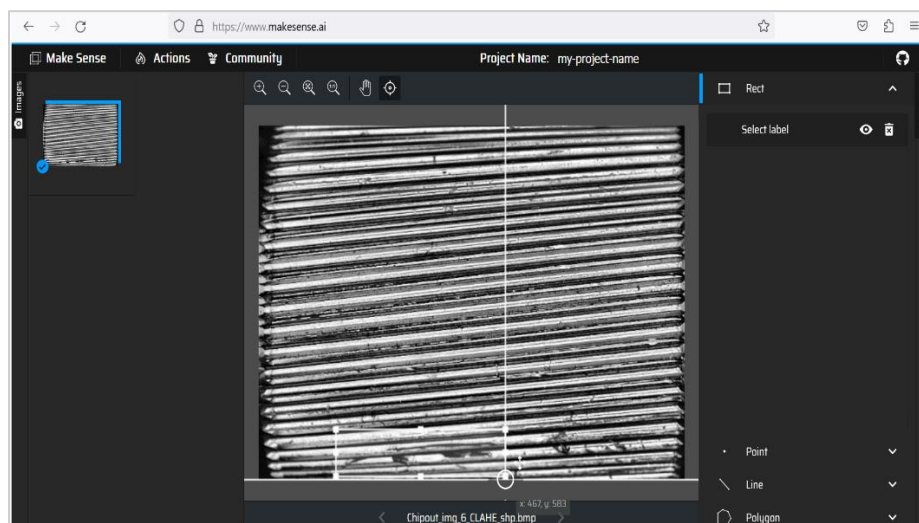
- i. Open the webpage, 'makesense.ai' and select image(s) to be uploaded. Select the mode as 'object detection' before starting the annotation process.



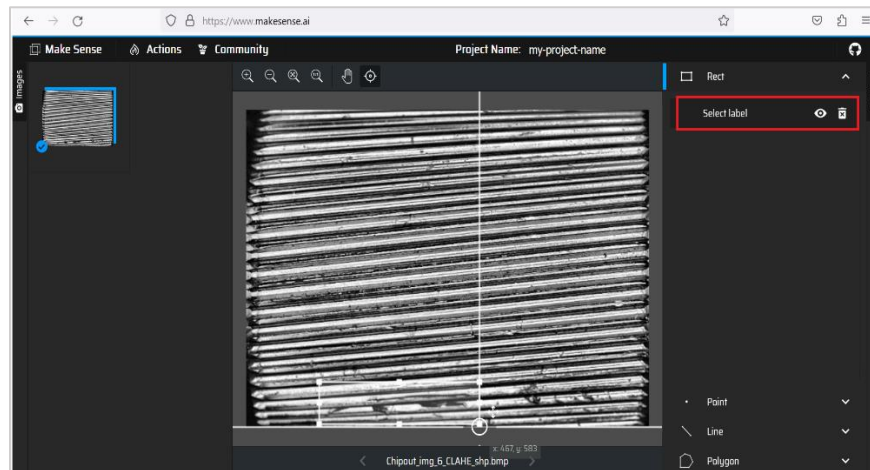
- ii. As shown above, click on '+' symbol (highlighted in red) to create labels for the annotation task.



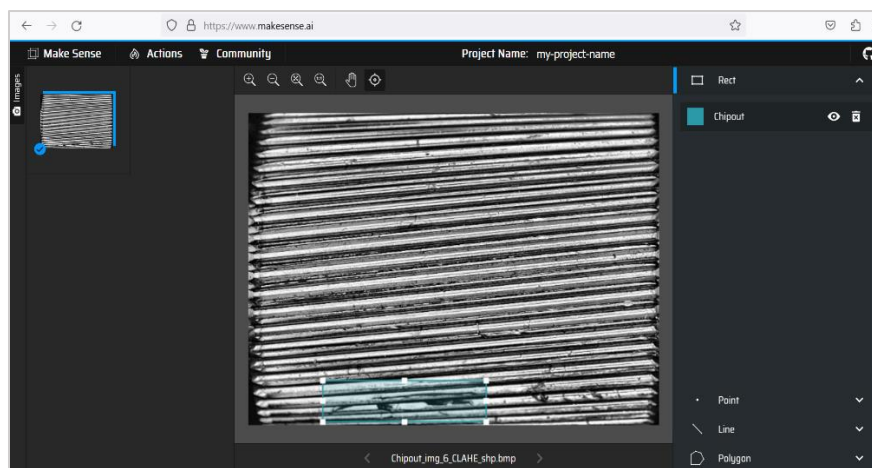
- iii. Click on 'Start project' after creating label(s).



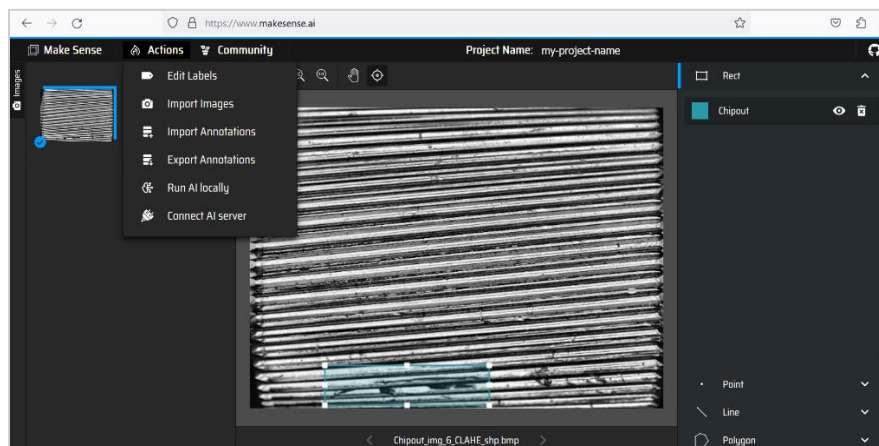
- iv. Annotate the region of interest by drawing a rectangle over it.



- v. Select the label as highlighted in the above image and choose the appropriate class.



- vi. Export the annotations as .XML file(s).



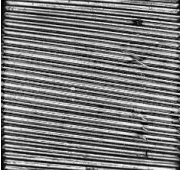
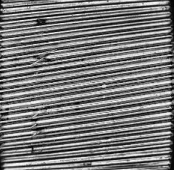
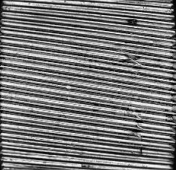
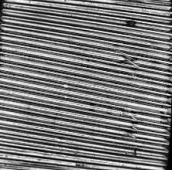
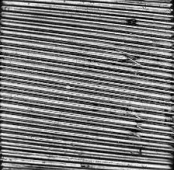
- Similarly, the annotations have to be performed for all training images. For each image, there is a corresponding (.XML) file.
- Resize the images to 320×320 using a resizing tool (Link: https://github.com/italojs/resize_dataset_pascalvoc) . Please follow the instructions in the readme file to use the resizing tool.
- The XML file comprises the class of the annotated object and also the coordinates of the bounding box (x_{min} , y_{min} , x_{max} , y_{max}).

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<annotation>
  <folder>my-project-name</folder>
  <filename>Chipout_img_6_CLAHE_shp.bmp</filename>
  <path>my-project-name/Chipout_img_6_CLAHE_shp.bmp</path>
  <source>
    <database>Unspecified</database>
  </source>
  <size>
    <width>806</width>
    <height>584</height>
    <depth>3</depth>
  </size>
  <object>
    <name>Chipout</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>146</xmin>
      <ymin>500</ymin>
      <xmax>467</xmax>
      <ymax>577</ymax>
    </bndbox>
  </object>
</annotation>
```

- Since the dataset is very less for training and has class imbalance, additional data is created by using image augmentation. The script 'general_augmentation_updated.py' is run on a command line terminal to implement the process. The augmentation types used were – Flip, rotation and translation (X/ Y axes)

	Training data	Augmentation	Augmented data
Good	500	4x	2000
Chipout	213	9x	1917
Damaged	214	9x	1926

Base image	Augmented images			
	Horizontal Flip	Vertical Flip	Rotation	Translation
			 (+2°)	 (+5 px)

10. The augmented training dataset is given as an input to the model, which is implemented in the training script (SSDMobileNetV2.ipynb)
11. Create a folder structure – One folder for images, within which you have training and test data folder. One folder should be created for labels, and finally one folder for training script/ notebook along with pretrained models.
12. Download the SSD Mobilenet V2 pretrained model (comprises pretrained checkpoints along with a pipeline.config file)
13. Within the training script, in the 'pipeline.config' file, modify the mode to "detection" instead of "classification" and set the training steps to 30000.
14. Once the model has been trained, in the checkpoints folder, the latest checkpoint along with a test image has to be provided in the script in order to obtain the detections. The 'confidence_score' can be varied between 0.5 and 0.95.
15. The trained checkpoints can be converted into a frozen (.pb) file by running the following command on the command line interface/ terminal:

```
python3 [PATH_TO_TENSORFLOW_MODELS]/research/object_detection/exporter_main_v2.py --
input_type=image_tensor --pipeline_config=[PATH_TO_WORKSPACE_DIRECTORY]/my_ssd_mobnet/ pipeline.config --
trained_checkpoint_dir=[PATH_TO_WORKSPACE_DIRECTORY]/my_ssd_mobnet/ --
output_directory=[PATH_TO_WORKSPACE_DIRECTORY]/my_ssd_mobnet/export
```

16. To convert a frozen/ saved model (.pb) file into a tflite model, the following command has to be run on the command line interface/ terminal:

```
python3 [PATH_TO_TENSORFLOW_MODELS]/research/object_detection/export_tflite_graph_tf2.py --
pipeline_config=[PATH_TO_WORKSPACE_DIRECTORY]/my_ssd_mobnet/ pipeline.config --
trained_checkpoint_dir=[PATH_TO_WORKSPACE_DIRECTORY]/my_ssd_mobnet/ --
output_directory=[PATH_TO_WORKSPACE_DIRECTORY]/my_ssd_mobnet/tfliteexport
```

With this, we get an intermediate tflite file in (.pb) format. To complete the conversion to tflite model with full-integer quantization, run the script/ notebook "TFLite_conversion". In order to realize this, a representative dataset is defined, followed by defining the path containing the intermediate tflite saved model (.pb) file.

17. The quantized tflite model is created by using a representative dataset (~ 200 images from the training dataset) in order to recalibrate the weights into uint8 data type and an appropriate normalization factor $\left((Resized\ Image) * \frac{1}{127.5} - 1 \right)$. The quantized model has the operations of backbone architecture quantized from fp32 to uint8 format, while maintaining the full precision of input and output nodes (fp32 format). Although the input node can be quantized to uint8 format, it is recommended to maintain the output node format as fp32.

18. The quantized tflite model is then deployed on Polarfire SOC Icicle kit, by invoking the tflite interpreter.

Detections on images:

