1) Downloaded the Roboflow dataset (Rust Detection v8) in YOLOv8 format

- Your dataset came as:
    - `train/images`, `train/labels`
    - `valid/images`, `valid/labels`
    - `test/images`, `test/labels`
- Each image has a `.txt` label file in YOLO format with bounding boxes if rust exists.

2) Converted YOLO object detection labels → binary classification dataset

We created a new dataset structured like **ImageFolder**:

```csharp
rust_cls/
  train/
    rust/      (images where label file has ≥1 box)
    no_rust/   (images where label file is empty/missing)
  valid/
    rust/
    no_rust/
  test/
    rust/
    no_rust/
```

Rule used:

- If `labels/<image>.txt` exists and has content → `rust`
  Else → `no_rust`

Your class counts ended up:

- **train:** rust 35020, no_rust 2168
  valid: rust 2968, no_rust 188
  test: rust 1471, no_rust 129

  So we *do* have both classes (great), but it's very imbalanced (~94% rust).

3) Trained a **binary image classifier** on GPU

Model we used : **MobileNetV3 Small** (from `torchvision`)

Why this model:

- Fast to train
- Lightweight for deployment
- Good baseline for images
- Easy ONNX export

Training details:

- Input: images resized to **224×224**
- Augmentations: horizontal flip + slight color jitter (to generalize better)
- Loss: **CrossEntropyLoss with class weights**
- We used class weights because dataset is imbalanced (many more rust images than no_rust)
- Optimizer: AdamW
- Trained for a few epochs (like 5)

4) Exported the trained classifier to **ONNX**

After training, we exported:

✅ `rust_model.onnx`

- This is the trained MobileNetV3 classifier in ONNX format.
- ONNX is perfect because we can run inference in Docker using **onnxruntime** (lightweight, CPU-friendly).

✅ `rust_labels.json`

- This is the label mapping like: ["no_rust", "rust"]

5) Downloaded these artifacts to your local machine

From Colab, we downloaded:

- `rust_model.onnx`
- `rust_labels.json`

And on your local repo, you will place them in:

C:\oxmaint-predictive-agent\artifacts\rust_model.onnx
C:\oxmaint-predictive-agent\artifacts\rust_labels.json

Why we're doing this

Now your pipeline will have **two modalities**:

1. **Sensor model (LightGBM)** → failure_probability + time_to_breakdown
2. **Image rust classifier (ONNX)** → rust_probability / rust_detected

Then in the FastAPI service we do **fusion**:

- Sensor predicts baseline failure risk
- Rust image increases risk + sets predicted_fault_type to corrosion/rust when strong

So the image modality will finally help fill:
✅ `predicted_fault_type` (corrosion/rust)
✅ `top_signals` (rust_detected, rust_severity_high, etc.)