Anomaly Detection System for Submersible Pump Impeller Castings

---

1. Project Overview

In modern manufacturing industries, detecting product defects at early stages is essential for quality assurance and process optimization. This project presents an **Anomaly Detection System** built using **Teachable Machine by Google** and deployed via a **Streamlit-based Web Application**.

The aim is to classify industrial casting products as either **Normal** or **Defective** based on their visual features. We used a real-life industrial dataset from Kaggle containing casting product images, trained a classification model using Teachable Machine, and exported it as a TensorFlow Lite model (.tflite).

The trained model was integrated into a user-friendly web interface developed using Streamlit. Users can upload an image of a casting product, and the system will return whether it is defective or not. This project demonstrates the power of no-code model training combined with Python-based deployment for solving real-world quality inspection problems.

---

2. Problem Statement

In manufacturing industries, manual inspection of products for defects is time-consuming, subjective, and prone to human error. Consistent and automated quality inspection is critical for reducing production losses and maintaining high product standards.

This system is designed to simulate real-world quality control processes and explore how AI tools can be used to assist inspectors in manufacturing units.

---

3. Dataset Description

The dataset used for this project is titled **“Real-life Industrial Dataset of Casting Product”** and is publicly available on Kaggle.

This dataset contains grayscale images of metal casted products used in manufacturing. The goal is to detect whether a product is **defective** or **normal** based on visible surface features

Normal: 600 images (60%)

Defective: 400 images (40%)

All images were uploaded into **Google Teachable Machine’s Image Project** tool under two distinct classes. Images were resized and auto-augmented by Teachable Machine during training.

Class labels assigned:

- **Class 1:** Normal
- **Class 2:** Defective

Dataset was split automatically by Teachable Machine into training and validation sets.

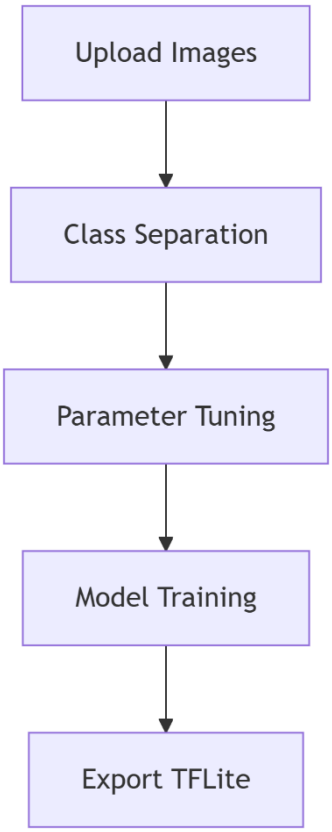
4. Model Building using Teachable Machine

Teachable Machine is a web-based tool by Google that allows users to train custom machine learning models without writing any code. It supports image, audio, and pose-based classification and is suitable for rapid prototyping.

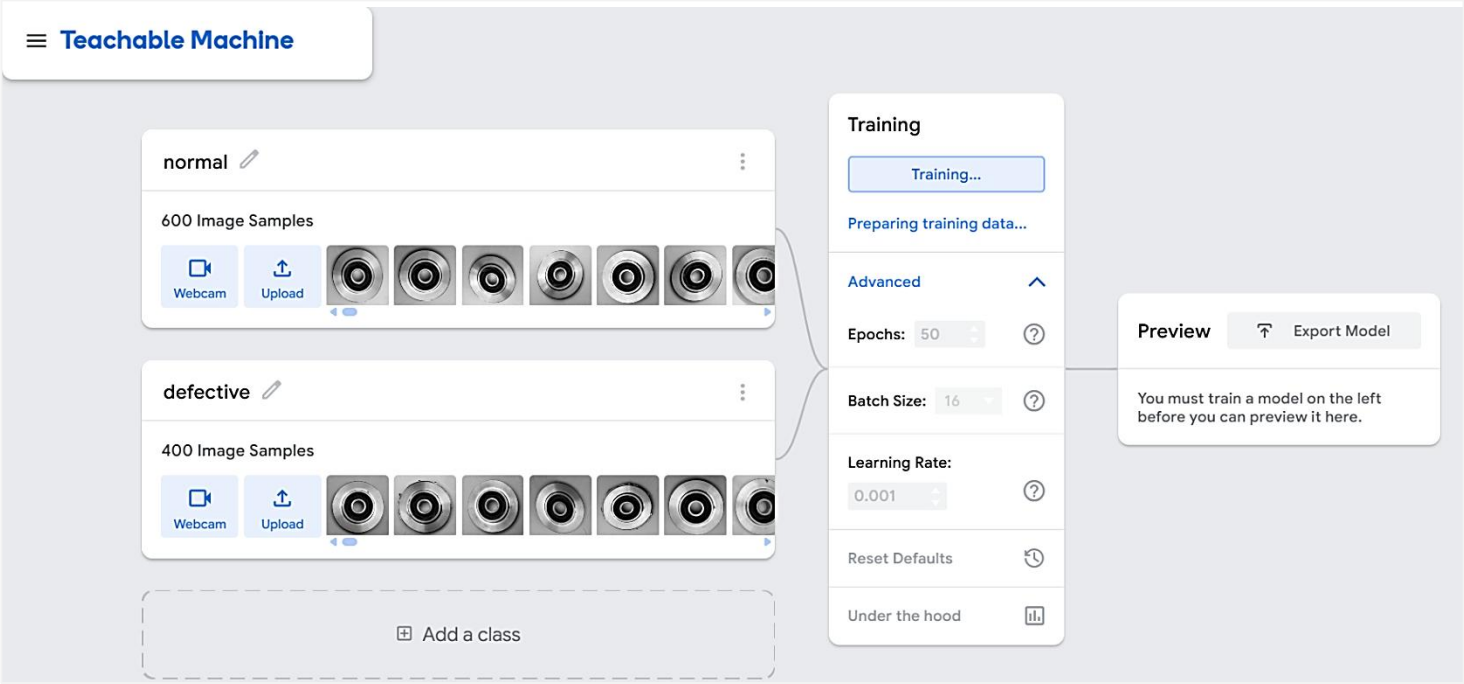
In this project, Teachable Machine's **Image Project** feature was used to build a binary classification model for **Anomaly Detection**.

Steps Followed:

Parameter	Value	Optimization Strategy
Epochs	50	Increased from 30 → 50
Batch Size	16	Tested 16/32 variants
Learning Rate	0.001	Fine-tuned from default

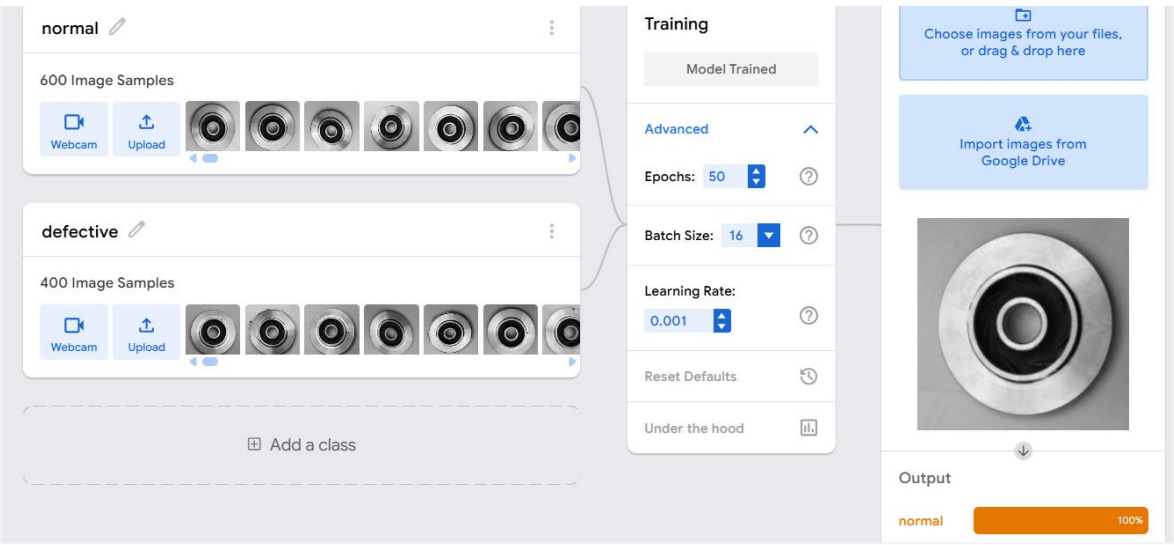


- **Training Process**:
  1. Image upload to Teachable Machine
  2. Class separation (Normal/Defective)
  3. Model architecture selection (MobileNetV2)
  4. Augmentation techniques (rotation, flip, zoom)
  5. Accuracy progression monitoring
- **Export Configuration**:
  - TensorFlow Lite format
  - Floating point quantization
  - `model.tflite` and `labels.txt` files

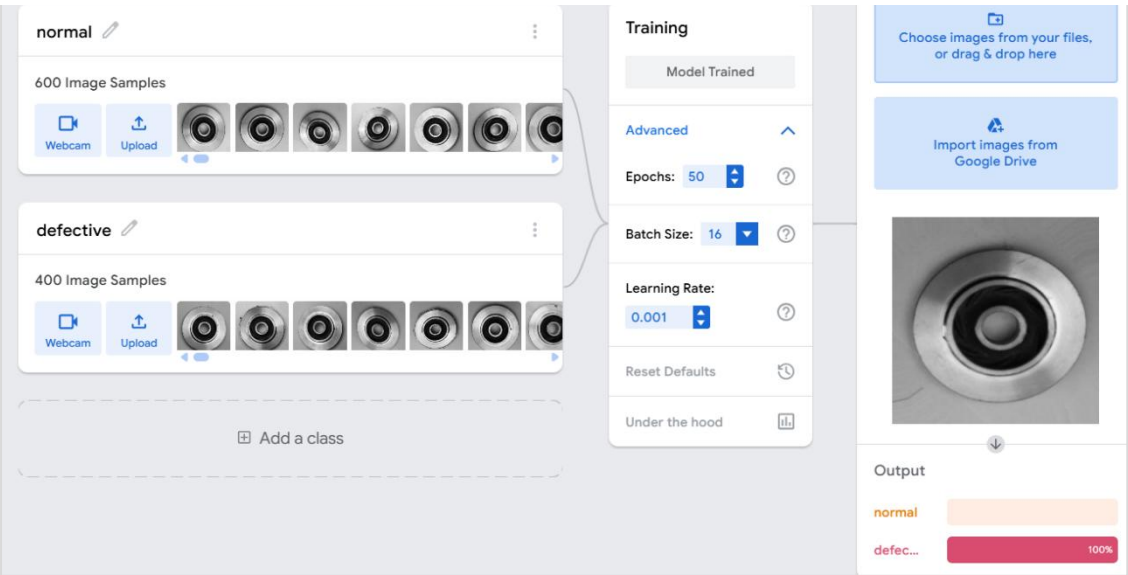


Teachable Machine Image Project interface with two classes: Normal and Defective.

1. *Teachable Machine: A normal casting product image is correctly classified as "Normal".*



2. *Teachable Machine: A defective casting product image is correctly classified as "Defective".*



---

5. Streamlit App Development

The final step in this project involved deploying the trained anomaly detection model in a web-based interface using **Streamlit**, a Python library used for building interactive web apps quickly and easily.

The main purpose of the app was to allow users to upload an image of a casting product and receive a classification result—**Normal** or **Defective**—based on the model exported from Teachable Machine.

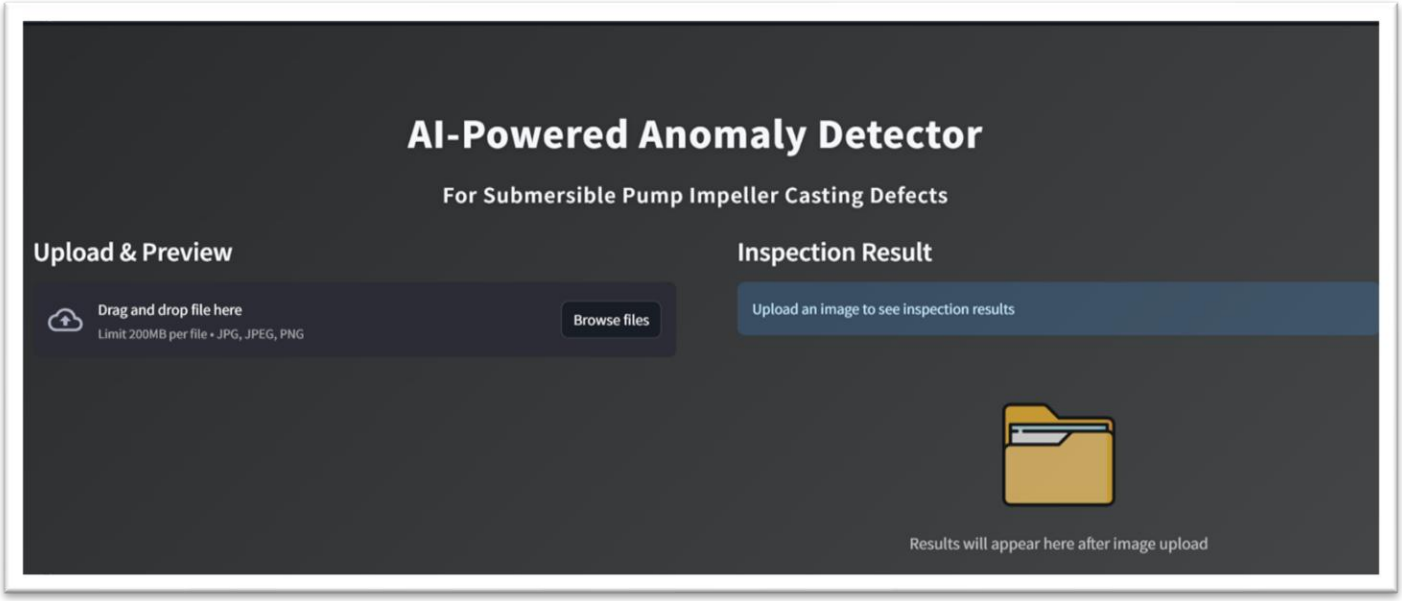
Tech Stack Used:

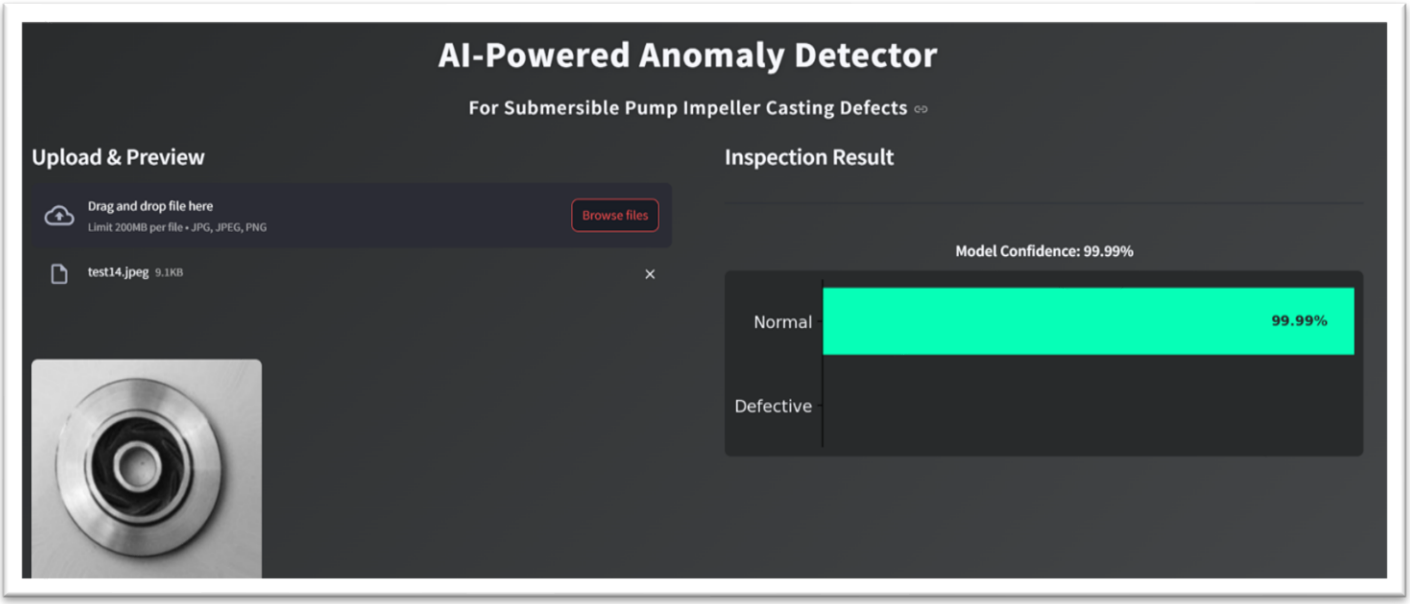
Tool/Library	**Purpose**
Streamlit	Web app UI and deployment
TensorFlow Lite (TFLite)	Lightweight ML model integration
PIL (Python Imaging Library)	Image handling and display
NumPy	Image array processing

Key Features of the App:

1. **Image Upload:** Users can upload .jpg or .png images of casting products via a file uploader widget.
2. **Model Loading:** The exported .tflite model from Teachable Machine is loaded using TensorFlow Lite Interpreter.
3. **Preprocessing:** Uploaded images are resized and normalized to match the input shape expected by the model.
4. **Inference:** The model predicts whether the uploaded product is *Normal* or *Defective* with confidence scores.
5. **Output Display:** The prediction is displayed clearly along with confidence percentage. Optionally, the uploaded image is also shown to the user.

Streamlit GUI : [Open in Streamlit]





Example Code:

```python
import streamlit as st
import numpy as np
from PIL import Image
import tflite_runtime.interpreter as tflite

# Load TFLite model
interpreter = tflite.Interpreter(model_path="model.tflite")
interpreter.allocate_tensors()

# Get input/output details
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Load labels
with open("labels.txt", "r") as f:
    labels = [line.strip() for line in f.readlines()]

# Streamlit app
st.title("Defect Detection")
uploaded_file = st.file_uploader("Upload casting image", type=["jpg", "png", "jpeg"])

if uploaded_file is not None:
    # Load and preprocess image
    img = Image.open(uploaded_file).convert("RGB")
    img = img.resize((300, 300))  # Match model input size
    img_array = np.array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0).astype(np.float32)

    # Display preview
    st.image(img, caption="Uploaded Image", width=200)

    # Run inference
    interpreter.set_tensor(input_details[0]['index'], img_array)
    interpreter.invoke()
    prediction = interpreter.get_tensor(output_details[0]['index'])

    # Get results
    class_idx = np.argmax(prediction[0])
    confidence = np.max(prediction[0])
    class_label = labels[class_idx]

    # Display result
    if class_label == "Normal":
        st.success(f"✅ Normal Casting ({confidence:.2%})")
    else:
        st.error(f"❌ Defect Detected ({confidence:.2%})")
```

How to Run:

1. Export model from Teachable Machine as TFLite
2. Saved as `model_unquant.tflite with labels.txt`
3. Install requirements:

   ```
   pip install -r requirements.txt
   ```

4. Run the app:

   ```
   streamlit run app.py
   ```

Deployment to Streamlit Sharing:

1. Create GitHub repository with:

```
project_folder/
├── app.py
├── model.tflite
├── labels.txt
├── requirements.txt
└── sample_images/
```

2. Go to : [Streamlit Community Cloud]
3. Connect repository and deploy

Conclusion

The project successfully demonstrated the integration of an image-based anomaly detection model built using Teachable Machine into a Streamlit web application.

Summary:

- Used real-world industrial casting product dataset for binary classification (normal vs defective).
- Developed and trained the model on Teachable Machine and exported as a TFLite file.
- Integrated the model into a responsive Streamlit app with real-time inference.

Outcome:

- The system correctly identified defective vs non-defective parts with high confidence.
- This project serves as a minimal prototype for an industrial visual inspection tool.

Future Scope:

- Model retraining with larger and more diverse datasets.
- Integration with camera input for live streaming inspection.

Useful Links and References:

- Teachable Machine: https://teachablemachine.withgoogle.com/
- Dataset Used: Casting Product Dataset (Kaggle)
- Streamlit Cloud: https://share.streamlit.io/
- Complete Project: Github Repository
- **Live App:** https://casting-anomaly-detection.streamlit.app/