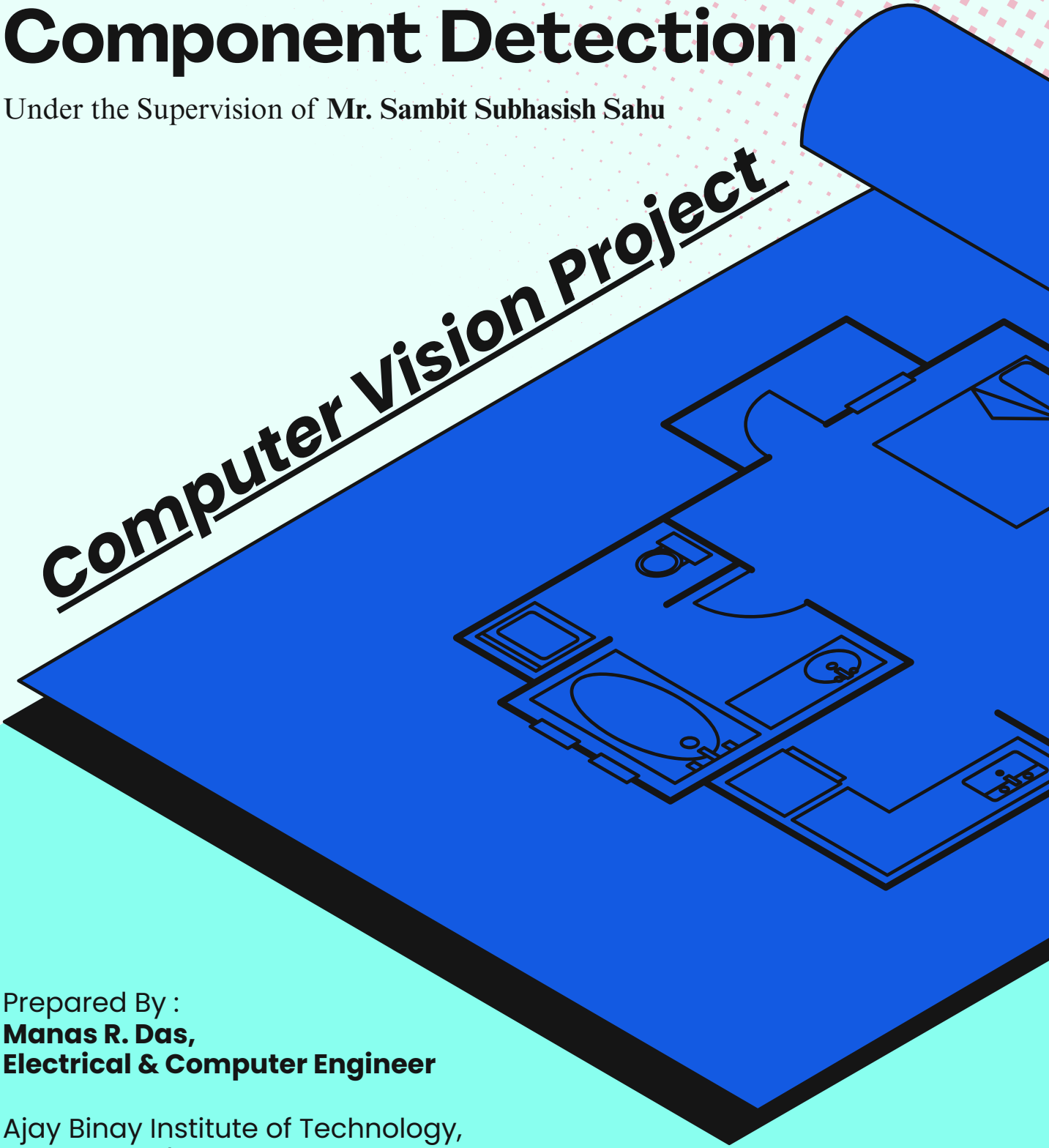
Odisha Computer
Application Centre

# Real-Time PCB Component Detection

Under the Supervision of **Mr. Sambit Subhasish Sahu**

## Computer Vision Project




Prepared By :
**Manas R. Das,**
**Electrical & Computer Engineer**

Ajay Binay Institute of Technology,
CDA, Cuttack

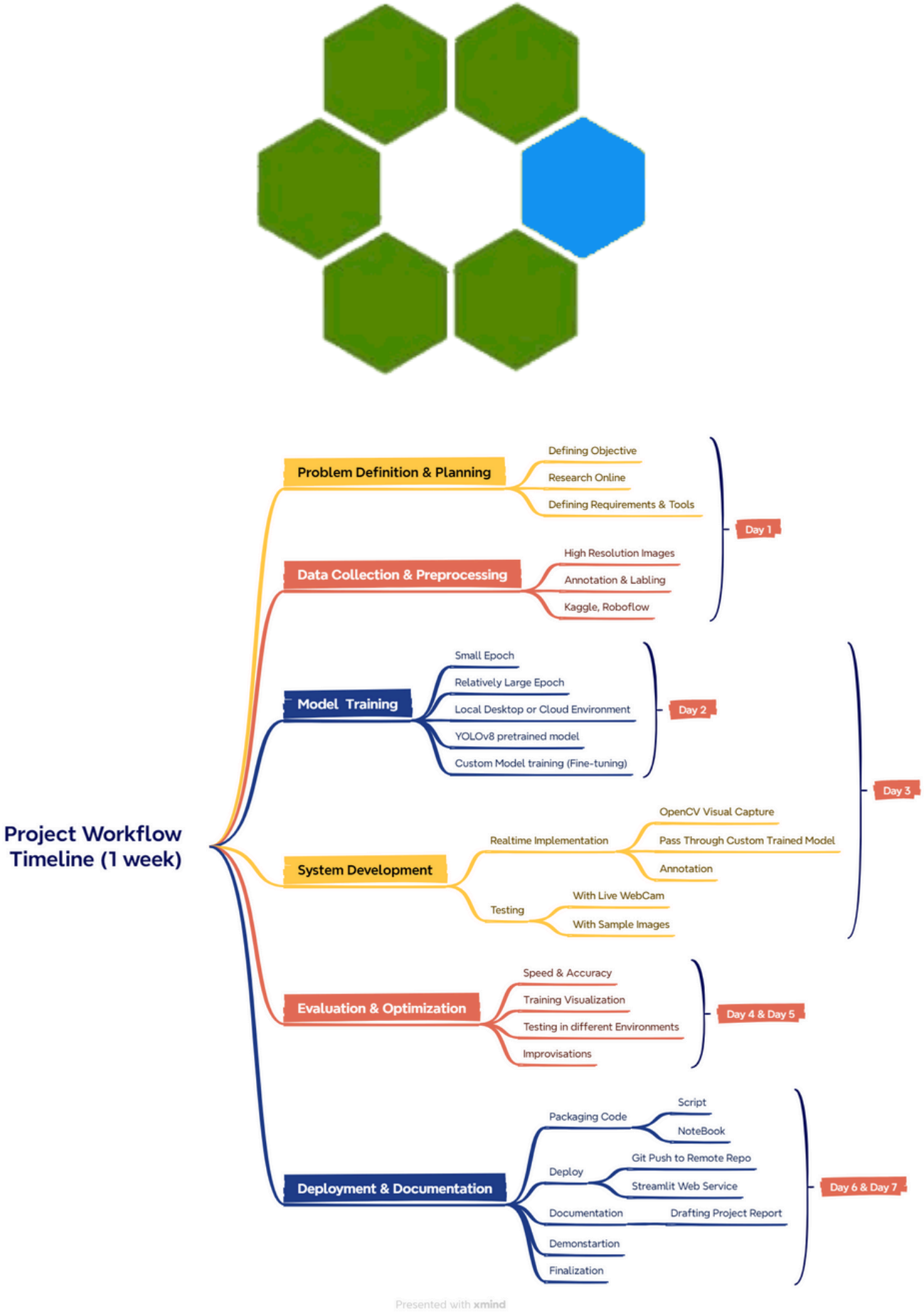



Fig. 1: Project Workflow Timeline

## *Abstract:*

This project presents an end-to-end pipeline for the automatic detection and classification of Printed Circuit Board (PCB) components from high-resolution images. The methodology integrates classical computer vision techniques with advanced deep learning for robust and scalable industrial inspection. The workflow encompasses image preprocessing, including resizing, grayscale conversion, noise reduction, and contrast enhancement to optimize images for analysis. It then employs contour detection and segmentation to isolate potential component regions. For component identification, the project leverages both traditional template matching for standard components (e.g., resistors, capacitors, diodes) and a rule-based numerical operation approach using features like mean color, area, aspect ratio, and white pixel ratio. Furthermore, a deep learning-based object detection model, specifically YOLOv8, is utilized and trained to accurately detect and classify various PCB components. The project includes a comprehensive evaluation of the detection and classification performance using metrics such as precision, recall, and F1-score. Emphasizing scalability for real-time inspection, the solution demonstrates optimizations for efficient processing and provides instructions for deployment using Streamlit, making it suitable for practical, real-time industrial applications.


## *Introduction:*

Printed Circuit Boards (PCBs) are the backbone of modern electronic devices, containing a variety of components such as resistors, capacitors, and integrated circuits (ICs). Manual inspection of PCBs is time-consuming and prone to human error, especially in high-volume manufacturing environments. Automating the detection and classification of PCB components can significantly enhance quality control, reduce costs, and improve production efficiency.

## *Objective & Scope:*

The objective of this project was to design and implement a real-time computer vision system capable of automatically detecting and classifying PCB components from live camera feeds. The system identifies and annotates components such as resistors, capacitors, and ICs, etc. providing immediate visual feedback for inspection and quality assurance.

## *Tools Used in This Project:*

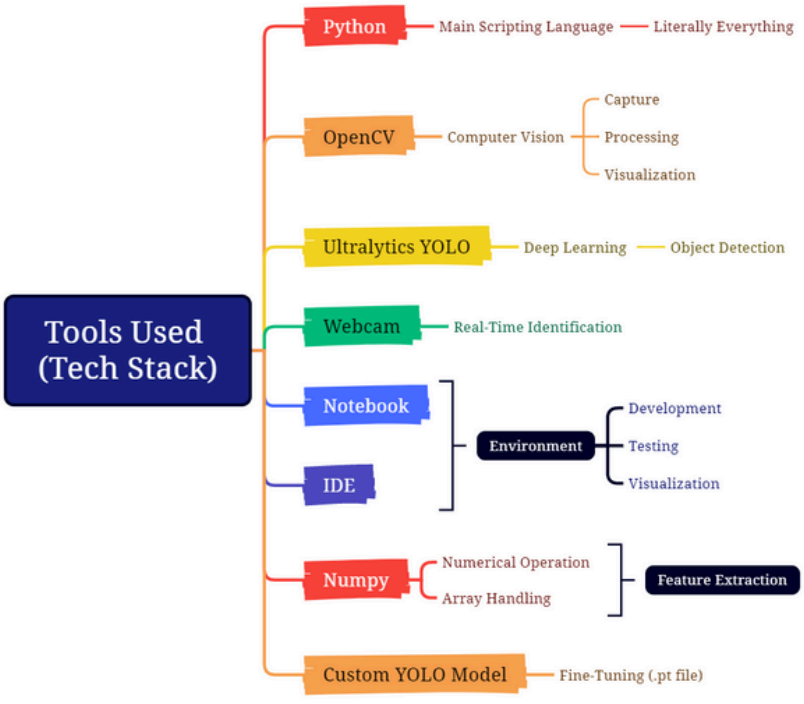



Fig. 2: Tools & Technologies

## *Project Architecture:*

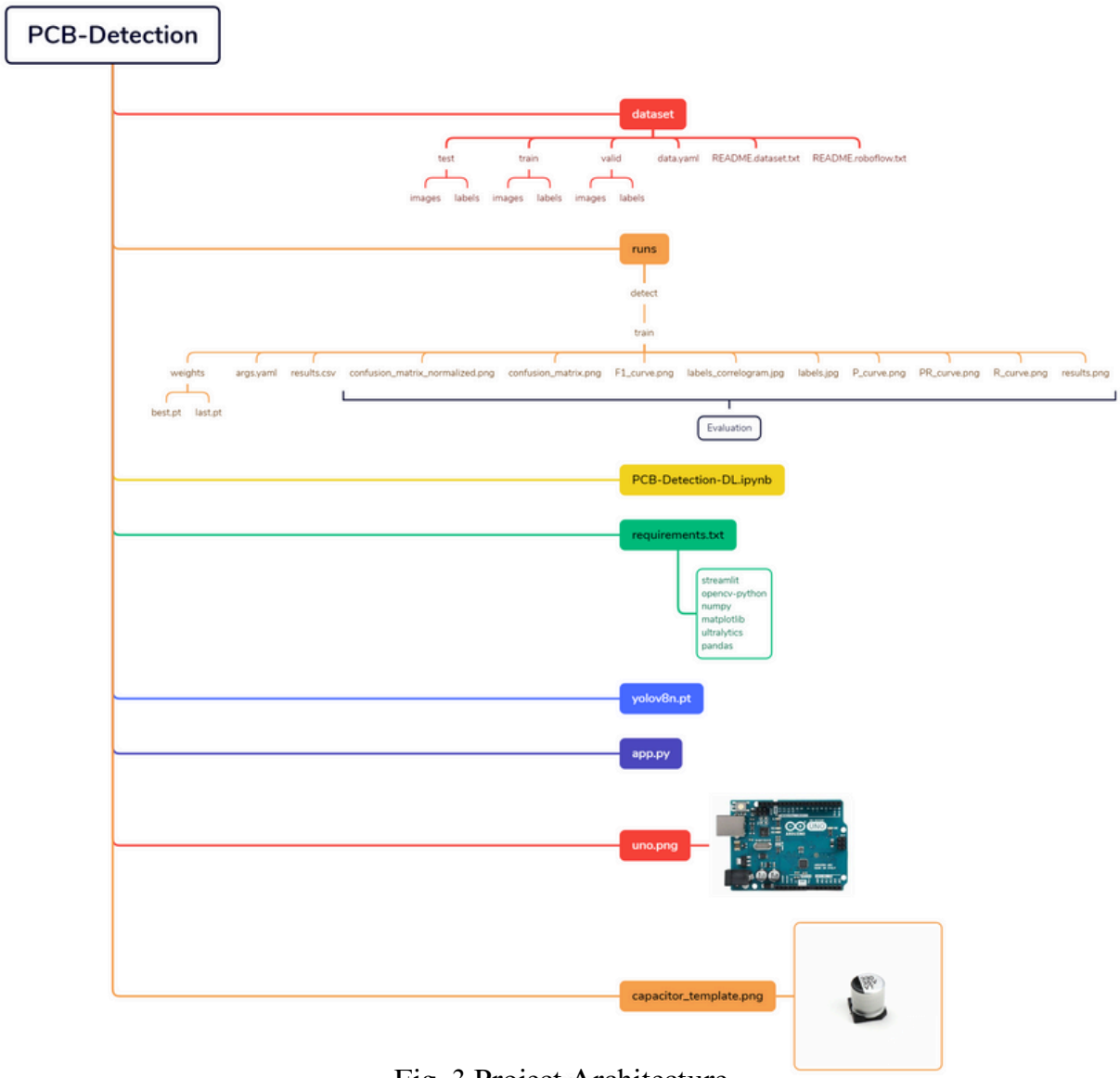



Fig. 3 Project Architecture

## *Methodology:*

**A) Data Handling:**

A search was conducted online on Public Dataset providing domains (e.g. Kaggle, Roboflow, Google Datasets, etc.) The set of data had to contain images of various components fabricated into a Printed Circuit Board (PCB). The images if pre-labeled would be very useful in-order to save our time. If not pre-labeled then we got to label the expected contour of components ensuring accuracy via different labelling tools available in the market (e.g. labelme, CVAT, etc.).

Since we're using **Ultralytics-YOLO** we must arrange the dataset to proper yolo-format. The dataset has to contain testing, training & validation series of labeled & non-labeled image (binary format) data & a compulsory **data.yaml** having all the object classes & original author information.

The fig. 4 represents the proper yolo-formatted dataset.

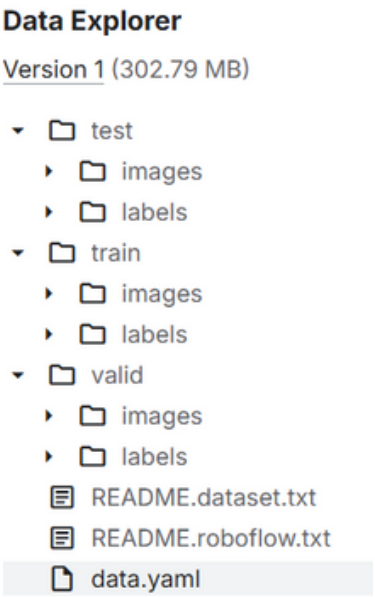



Fig. 4 Dataset Format

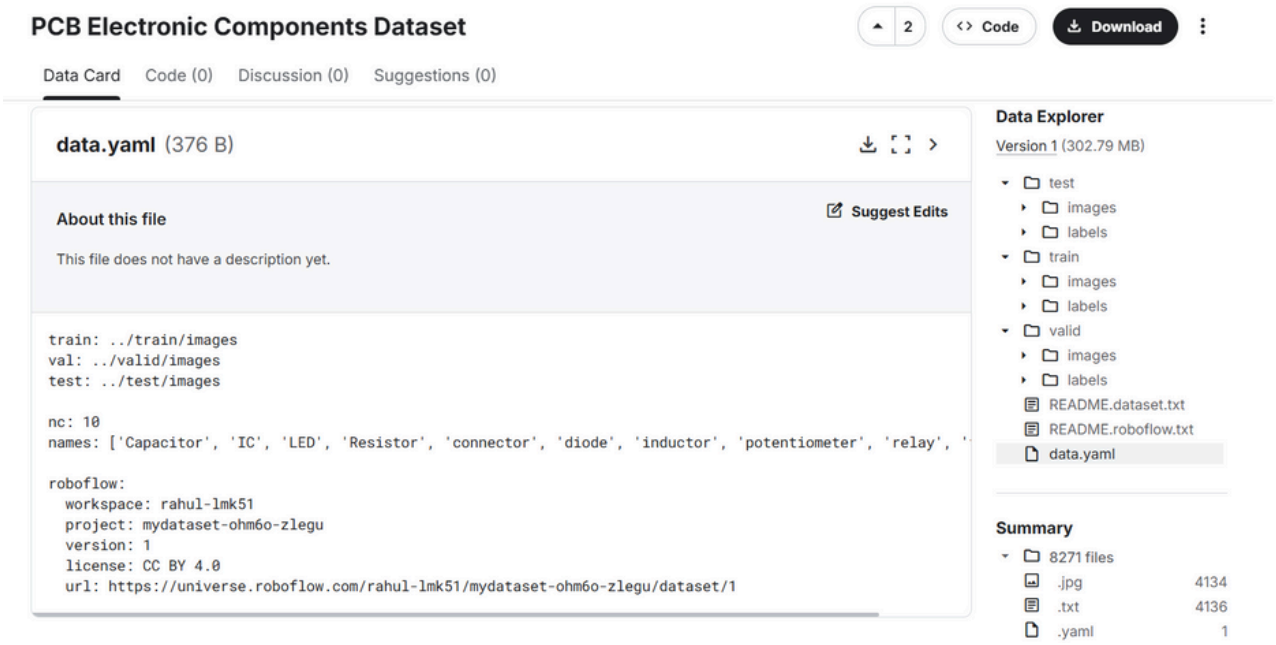



Fig. 5 PCB Electronic Components Dataset

The dataset used in this project was taken from **Kaggle** which was originally available on **roboflow**. More information available in the **"References"** section.

*The whole project is created in the form of a **Jupyter Python Notebook** named **"PCB-Detection-DL.ipynb"** for the ease of handling & portability, This notebook demonstrates an end-to-end pipeline for automatic detection and classification of PCB (Printed Circuit Board) components from high-resolution images using computer vision and deep learning techniques with a focus on scalability for real-time industrial inspection shown as follows:*

**1. Import Required Libraries**

Import essential libraries for image processing, visualization, and deep learning.
We use:

- OpenCV and NumPy for computer vision tasks
- Matplotlib for visualization
- Ultralytics YOLO for deep learning-based object detection.

```python
# Install required libraries (run this cell)
%pip install opencv-contrib-python numpy pandas matplotlib ultralytics
```

Python

```python
# Import libraries
import cv2
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from ultralytics import YOLO

# Suppress warnings for cleaner output
import warnings
warnings.filterwarnings('ignore')
```

Python

**2. Load and Visualize PCB Image**

Load a high-resolution PCB image and display it for initial inspection. Replace the image path with your own PCB image as needed. I this case we used an image of **Arduino UNO R3 IoT Board** named **'uno.png'** present in the same working directory.

```python
# Load and visualize PCB image
pcb_image_path = 'uno.png'  # Replace with your image path

# Read image using OpenCV
image = cv2.imread(pcb_image_path)
if image is None:
    raise FileNotFoundError(f"Image not found at {pcb_image_path}")

# Convert BGR (OpenCV default) to RGB for visualization
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(12, 8))
plt.imshow(image_rgb)
plt.title('Original High-Resolution PCB Image')
plt.axis('off')
plt.show()
```

Python

Original High-Resolution PCB Image

**3. Image Preprocessing**

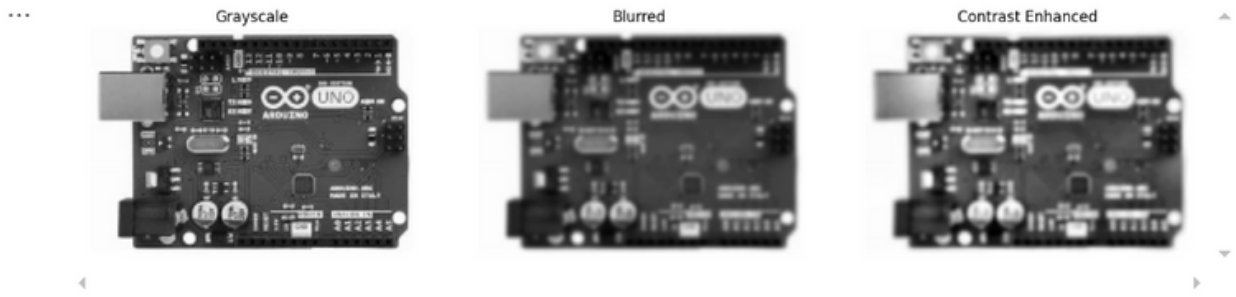
Apply preprocessing steps such as:

- Resizing
- Grayscale Conversion
- Noise Reduction
- Contrast Enhancement

to prepare the image for analysis.

```python
# Image Preprocessing
# Resize for faster processing (optional, depending on image size)
resized = cv2.resize(image, (0, 0), fx=0.5, fy=0.5)

# Convert to grayscale
gray = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY)

# Noise reduction using Gaussian blur
blurred = cv2.GaussianBlur(gray, (5, 5), 0)

# Contrast enhancement using CLAHE
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
enhanced = clahe.apply(blurred)

plt.figure(figsize=(16, 4))
plt.subplot(1, 3, 1)
plt.imshow(gray, cmap='gray')
plt.title('Grayscale')
plt.axis('off')
plt.subplot(1, 3, 2)
plt.imshow(blurred, cmap='gray')
plt.title('Blurred')
plt.axis('off')
plt.subplot(1, 3, 3)
plt.imshow(enhanced, cmap='gray')
plt.title('Contrast Enhanced')
plt.axis('off')
plt.show()
```

Python

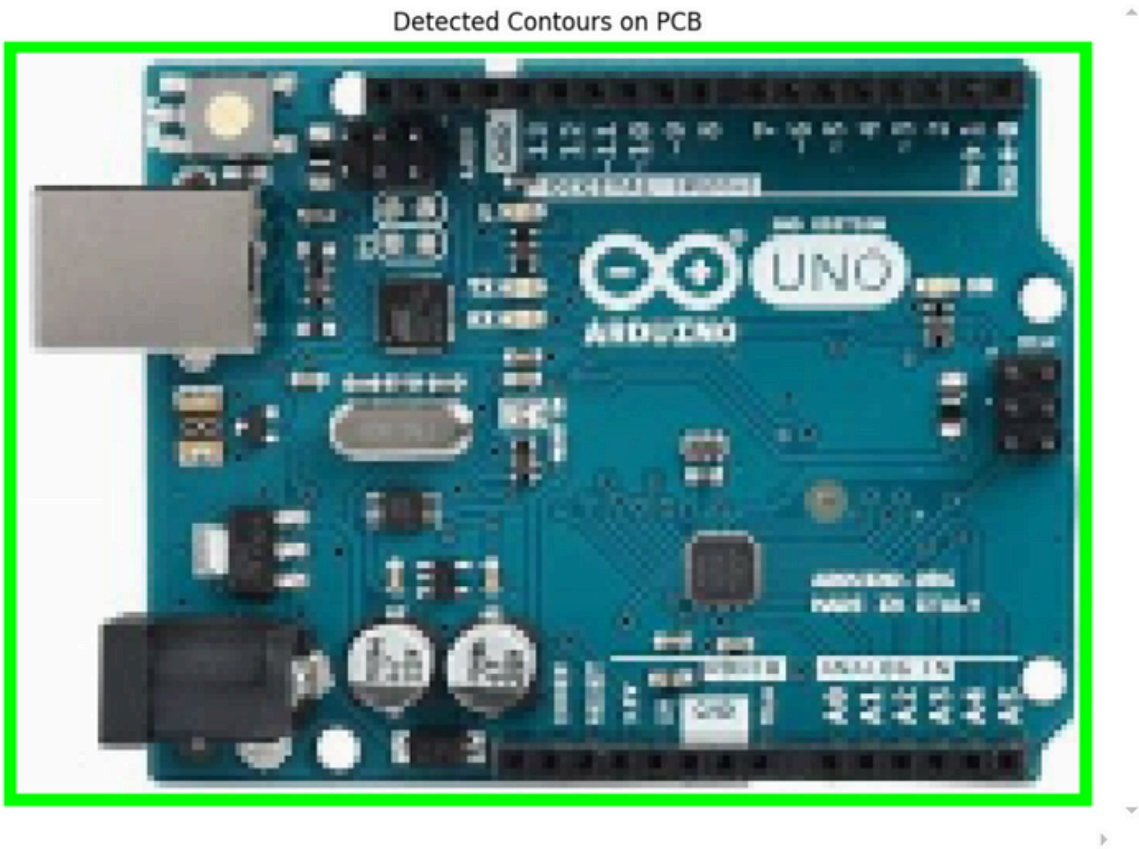


## 4. Contour Detection and Segmentation

Detect contours to segment potential component regions from the PCB image using OpenCV.

```python
# Contour Detection and Segmentation
# Thresholding to create binary image
_, thresh = cv2.threshold(enhanced, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Find contours
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Draw contours on a copy of the image
contour_img = resized.copy()
cv2.drawContours(contour_img, contours, -1, (0, 255, 0), 2)

plt.figure(figsize=(10, 8))
plt.imshow(cv2.cvtColor(contour_img, cv2.COLOR_BGR2RGB))
plt.title('Detected Contours on PCB')
plt.axis('off')
plt.show()

print(f"Number of contours detected: {len(contours)}")
```

Python




```
Number of contours detected: 1
```

5. **Template Matching for Component Identification**

Apply template matching to identify standard components such as resistors, capacitors, and diodes. You need to provide template images for each component type. Here we have the image of a sample capacitor **'capacitor_template.png'** in the same working directory.

```python
# Template Matching for Component Identification
# Example: Matching a capacitor template

template_path = 'capacitor_template.png'  # Replace with your template path
template = cv2.imread(template_path, 0)
if template is None:
    print("Capacitor template not found. Skipping template matching.")
else:
    h, w = template.shape[:2]
    cap = cv2.matchTemplate(gray, template, cv2.TM_CCOEFF_NORMED)
    threshold = 0.7
    loc = np.where(cap >= threshold)

    matched_img = resized.copy()
    for pt in zip(*loc[::-1]):
        cv2.rectangle(matched_img, pt, (pt[0] + w, pt[1] + h), (255, 0, 0), 2)

    plt.figure(figsize=(10, 8))
    plt.imshow(cv2.cvtColor(matched_img, cv2.COLOR_BGR2RGB))
    plt.title('Capacitor Template Matching Results')
    plt.axis('off')
    plt.show()
```

[7] ✓ 2.2s Python

Original High-Resolution PCB Image

**6. Numpy Logic for Component Identification**

Apply Numerical Operation to identify standard components such as resistors, capacitors, and diodes on the basis of pixel area, aspect ratio & white-balance after image processing.

However, this process may not be very efficient since lighting condition & distance form detector camera may affect the process of detection & might be misleading one.

Hence, Template Matching & Traditional Numpy Implementation is mostly not preferred.

At this point a Deep Learning Classification model got an upper hand over these two method of PCB components detection. It's discussed in the next section.

The Numpy logic used for this project is provided below. The process involves:

1. Importing **OpenCV** & **NumPy**
2. Image Processing using OpenCV
3. Feature Extraction with OpenCV & NumPy
4. Component classification using Python control structure
5. Contour Detection using OpenCv
6. Real-time Component detection using OpenCv

```python
# 📷 Advanced Real-Time PCB Component Detection using OpenCV

import cv2
import numpy as np
```

[8] ✓ 0.1s Python

```python
# Preprocessing function to enhance image quality
def preprocess(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) # Convert to grayscale
    blurred = cv2.GaussianBlur(gray, (5, 5), 0) # Apply Gaussian blur
    edges = cv2.Canny(blurred, 30, 120) # Edge detection
    # Morphological operations to close gaps in edges
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
    closed = cv2.morphologyEx(edges, cv2.MORPH_CLOSE, kernel)
    dilated = cv2.dilate(closed, kernel, iterations=2)
    return dilated
```

[9] ✓ 0.0s Python

```python
# Feature extraction function to analyze regions of interest (ROIs)
def extract_features(roi):
    mean_color = cv2.mean(roi)[:3] # Get mean color in BGR format
    h, w, _ = roi.shape  # Height and width of the ROI
    area = w * h  # Calculate area of the ROI
    # Calculate aspect ratio
    aspect_ratio = w / float(h)
    gray_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY) # Convert ROI to grayscale
    # Thresholding to create binary image
    _, binary = cv2.threshold(gray_roi, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    white_px = cv2.countNonZero(binary) # Count white pixels
    # Calculate black pixels
    black_px = area - white_px
    # Calculate ratio of white pixels to total area
    white_ratio = white_px / area if area > 0 else 0
    return [*mean_color, area, aspect_ratio, white_ratio]
```

[10] ✓ 0.1s Python

```python
# Function to classify components based on extracted features
def classify_component(features):
    _, g, _, area, ratio, white_ratio = features
    # More refined rules, can be replaced with ML model
    if area < 300:
        return "Noise"
    elif 300 <= area <= 1000:
        if ratio > 2.5:
            return "Resistor"
        elif white_ratio > 0.7:
            return "LED"
        else:
            return "Small Cap."
    elif 1000 < area <= 3000:
        if ratio < 1.3:
            return "Capacitor"
        elif white_ratio > 0.7:
            return "Connector"
        else:
            return "Medium IC"
    elif area > 3000:
        if ratio > 1.1:
            return "IC"
        elif white_ratio > 0.7:
            return "Socket"
        else:
            return "Large Comp."
    return "Unknown"
```

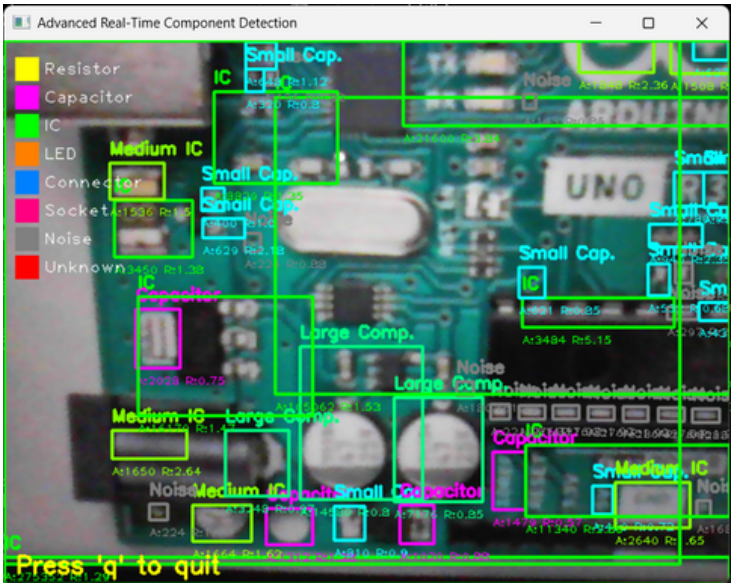
[11] ✓ 0.1s Python

```python
# Function to draw contour information on the output image
def draw_contour_info(output, x, y, w, h, label, features):
    color_map = {
        "Resistor": (0, 255, 255), # Yellow
        "Capacitor": (255, 0, 255), # Magenta
        "IC": (0, 255, 0), # Green
        "LED": (0, 128, 255), # Orange
        "Connector": (255, 128, 0), # Cyan
        "Socket": (128, 0, 255), # Purple
        "Small Cap.": (255, 255, 0), # Light Blue
        "Medium IC": (0, 255, 128), # Light Green
        "Large Comp.": (128, 255, 0), # Light Yellow
        "Noise": (128, 128, 128), # Gray
        "Unknown": (0, 0, 255) # Red
    }
    color = color_map.get(label, (0, 0, 255)) # Default to red for unknown
    cv2.rectangle(output, (x, y), (x+w, y+h), color, 2) # Draw rectangle around contour
    # Draw label above rectangle
    cv2.putText(output, label, (x, y-8), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
    # Show area and aspect ratio for debugging
    area = int(features[3])
    ratio = round(features[4], 2)
    # Display area and ratio below the rectangle
    cv2.putText(output,
                f"A:{area} R:{ratio}", (x, y+h+15), cv2.FONT_HERSHEY_PLAIN, 0.7, color, 1)
```

[13] ✓ 0.0s Python

```python
# Real-time Component Detection using OpenCV
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    processed = preprocess(frame)
    contours, _ = cv2.findContours(processed, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    output = frame.copy()

    for cnt in contours:
        x, y, w, h = cv2.boundingRect(cnt)
        if w > 10 and h > 10:
            roi = frame[y:y+h, x:x+w]
            features = extract_features(roi)
            label = classify_component(features)
            draw_contour_info(output, x, y, w, h, label, features)

    # Add legend
    legend_y = 25
    for name, color in [
        ("Resistor", (0,255,255)), ("Capacitor", (255,0,255)), ("IC", (0,255,0)),
        ("LED", (0,128,255)), ("Connector", (255,128,0)), ("Socket", (128,0,255)),
        ("Noise", (128,128,128)), ("Unknown", (0,0,255))
    ]:
        cv2.rectangle(output, (10, legend_y-10), (30, legend_y+10), color, -1)
        cv2.putText(output, name, (35, legend_y+5), cv2.FONT_HERSHEY_PLAIN, 1,
                    (255,255,255), 1)
        legend_y += 25
    cv2.putText(output, "Press 'q' to quit", (10, output.shape[0]-10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,255,255), 2)
    cv2.imshow("Advanced Real-Time Component Detection", output)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

[ ] Python



**7. Deep Learning-Based Object Detection**

Use a pre-trained or custom-trained deep learning model (e.g., YOLOv8) to detect and classify PCB components. For demonstration, we use a pre-trained YOLOv8 model via Ultralytics. Base version of YOLO may not detect components accurately as when we used yolov5s pretained model.

```
# Deep Learning-Based Object Detection with YOLOv5 (PyTorch Hub)
# Note: Requires internet connection for first-time model download

# Load YOLOv5 model
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)

# Prepare image for model
img_pil = Image.fromarray(cv2.cvtColor(resized, cv2.COLOR_BGR2RGB))

# Inference
results = model(img_pil, size=640)

# Results
results.print()
results.show()  # Opens a window with results (may not work in all environments)
```

[10] Python


There for use of a custom trained model is suitable for more accuracy & precision. However it may also depend upon the model size, parameter range, number of training epoch, dataset architecture, system architecture, CPU & GPU also the development environment & various other factors like lighting, distance, etc.

```
# Load a YOLOv8 model (choose 'yolov8n.pt', 'yolov8s.pt', etc.)
model = YOLO('yolov8n.pt')

# Train the model
model.train(
    data='dataset/data.yaml',  # path to your data.yaml
    epochs=5,                  # number of training epochs
    imgsz=640,                 # image size
    batch=8                    # batch size
)
```

[ ] Python

5 epochs completed in 4.529 hours. (Vary from system to system.)
Optimizer stripped from runs\detect\train2\weights\last.pt, 6.2MB

This will create the directory: **runs\detect\train**

Here inside the train folder there present the custom trained model **best.pt** inside the weights folder & it also contains the arguments list in yaml format, evaluation plots, results & training & validation batches.

```
Validating runs\detect\train2\weights\best.pt...
Ultralytics 8.3.148  Python-3.12.2 torch-2.7.0+cpu CPU (11th Gen Intel Core(TM) i5-1135G7
2.40GHz)
Model summary (fused): 72 layers, 3,007,598 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
25%|██▌       | 13/52 [00:26<01:38,  2.53s/it]
WARNING NMS time limit 2.800s exceeded
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
29%|██▉       | 15/52 [00:36<02:22,  3.86s/it]
WARNING NMS time limit 2.800s exceeded
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
100%|██████████| 52/52 [02:44<00:00,  3.16s/it]
                   all        827      13197      0.625       0.37      0.401      0.251
             Capacitor        314       4672      0.696      0.147       0.22      0.104
                    IC        440       1330      0.693      0.811      0.822      0.617
                   LED        124        422      0.599      0.417      0.419      0.205
              Resistor        300       4919      0.749      0.156      0.196     0.0869
             connector        157        768      0.452       0.41      0.381      0.227
                 diode         85        251      0.547      0.299      0.321      0.176
              inductor         59        131      0.412        0.3      0.289      0.214
         potentiometer         34         42      0.668       0.19      0.327      0.231
                 relay         16         34      0.709      0.717      0.733      0.503
            transistor        116        628      0.722      0.252        0.3      0.148
Speed: 3.7ms preprocess, 147.4ms inference, 0.0ms loss, 28.8ms postprocess per image
Results saved to runs\detect\train2
```

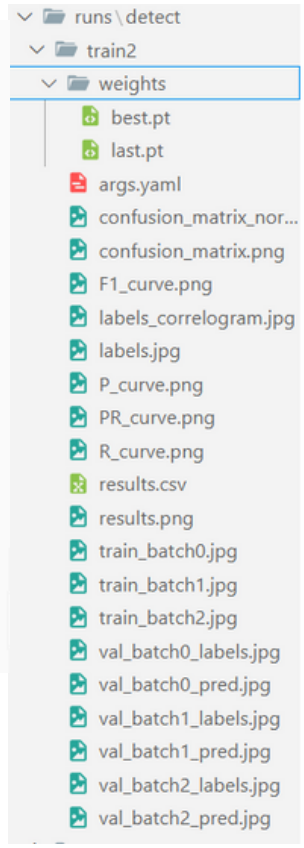


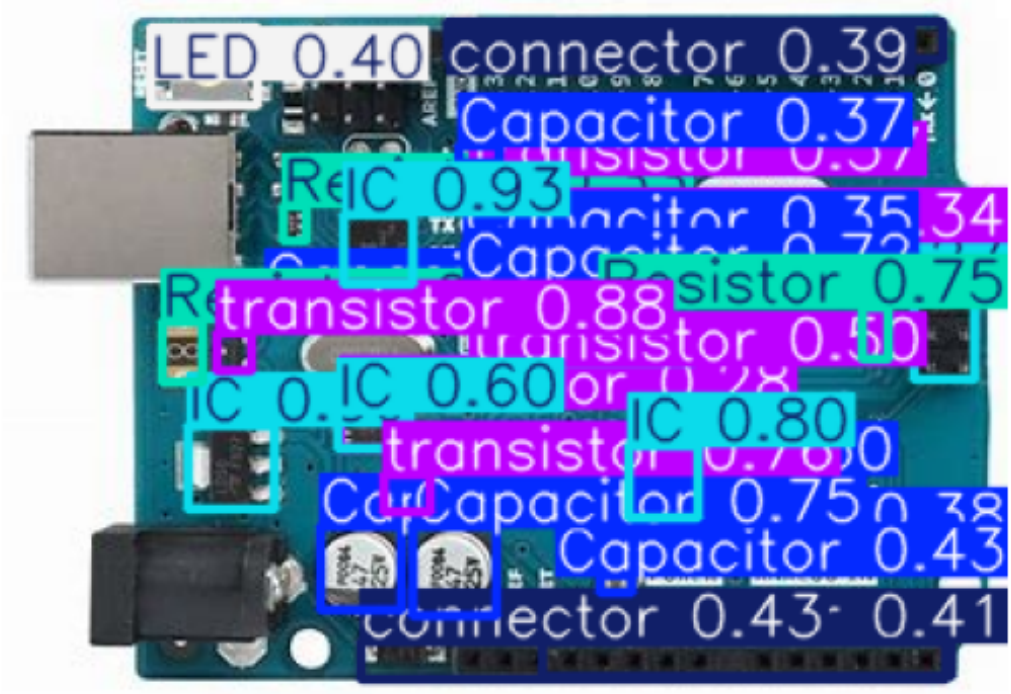

Fig. 6 Custom Model

## 8. Annotate Detected Components

Draw bounding boxes and labels on the image for each detected component using the deep learning model's output. We saved this as 'uno_comp.png' in our working directory.

```python
# Load your trained model
model = YOLO('runs/detect/train2/weights/best.pt')

# Inference on an image
results = model('uno.png') # Replace with your image path
results[0].show()
```

[ ] Python



**9. Evaluate Detection Accuracy**

Calculating metrices such as precision, recall, and F1-score to evaluate the detection and classification performance.

```
# Load the results.csv file from the specified path
df = pd.read_csv("runs/detect/train2/results.csv")

# Display the rows as cell output
df
```

[16] ✓ 0.1s Python

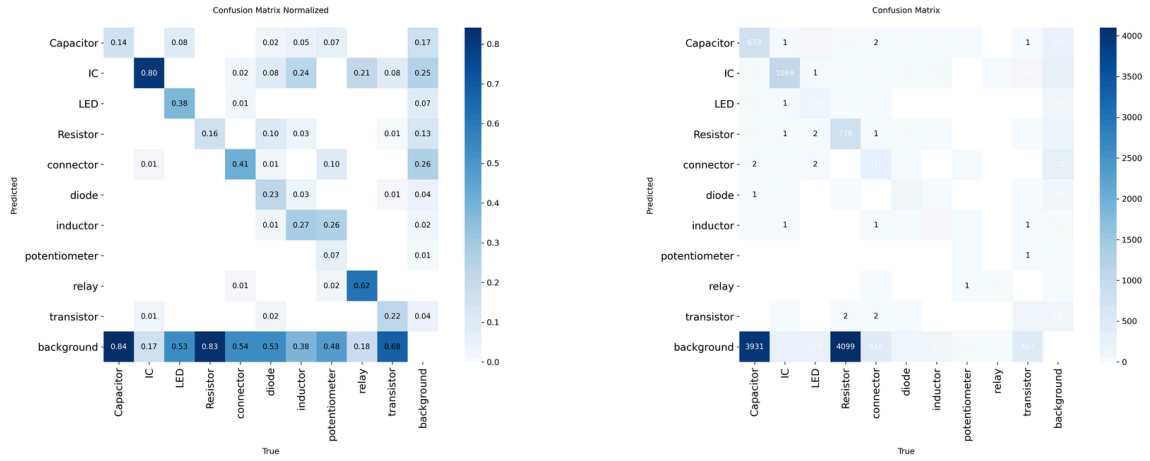
| epoch | time | train/box_loss | train/cls_loss | train/dfl_loss | metrics/precision(B) | metrics/recall(B) |
|---|---|---|---|---|---|---|
| 1 | 3891.63 | 1.54188 | 2.78041 | 1.29457 | 0.27581 | 0.18103 |
| 2 | 7519.81 | 1.47879 | 1.94560 | 1.24257 | 0.30420 | 0.22404 |
| 3 | 10243.10 | 1.45353 | 1.72829 | 1.20496 | 0.44012 | 0.28453 |
| 4 | 13169.30 | 1.38111 | 1.52445 | 1.17060 | 0.44111 | 0.34365 |
| 5 | 16303.80 | 1.35855 | 1.42148 | 1.15808 | 0.62279 | 0.36488 |

| metrics/mAP50(B) | metrics/mAP50-95(B) | val/box_loss | val/cls_loss | val/dfl_loss | lr/pg0 | lr/pg1 | lr/pg2 |
|---|---|---|---|---|---|---|---|
| 0.15472 | 0.10006 | 1.46581 | 2.29562 | 1.49823 | 0.000237 | 0.000237 | 0.000237 |
| 0.19883 | 0.12342 | 1.34908 | 2.07077 | 1.39822 | 0.000381 | 0.000381 | 0.000381 |
| 0.28242 | 0.17684 | 1.36057 | 1.63438 | 1.37423 | 0.000431 | 0.000431 | 0.000431 |
| 0.32860 | 0.19842 | 1.28480 | 1.54365 | 1.30495 | 0.000290 | 0.000290 | 0.000290 |
| 0.39776 | 0.24895 | 1.25604 | 1.35085 | 1.25368 | 0.000149 | 0.000149 | 0.000149 |

```
# Ordered list of result images for presentation from runs/detect/train2
image_files = [
    "confusion_matrix_normalized.png",
    "confusion_matrix.png",
    "F1_curve.png",
    "labels_correlogram.jpg",
    "labels.jpg",
    "P_curve.png",
    "PR_curve.png",
    "R_curve.png",
    "results.png"
]

base_path = "runs/detect/train2"

for fname in image_files:
    img_path = f"{base_path}/{fname}"
    img = cv2.imread(img_path)
    if img is not None:
        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        plt.figure(figsize=(8, 6))
        plt.imshow(img_rgb)
        plt.title(fname, fontsize=14)
        plt.axis('off')
        plt.show()
    else:
        print(f"Could not load {img_path}")
```

[17] ✓ 8.8s Python

### *A. Confusion Matrix:*




**Key Observations and Insights from This Matrix:**

**Good Performance:** The model performs well at classifying "IC" (80%), "Relay" (62%), and especially "Background" (84%).

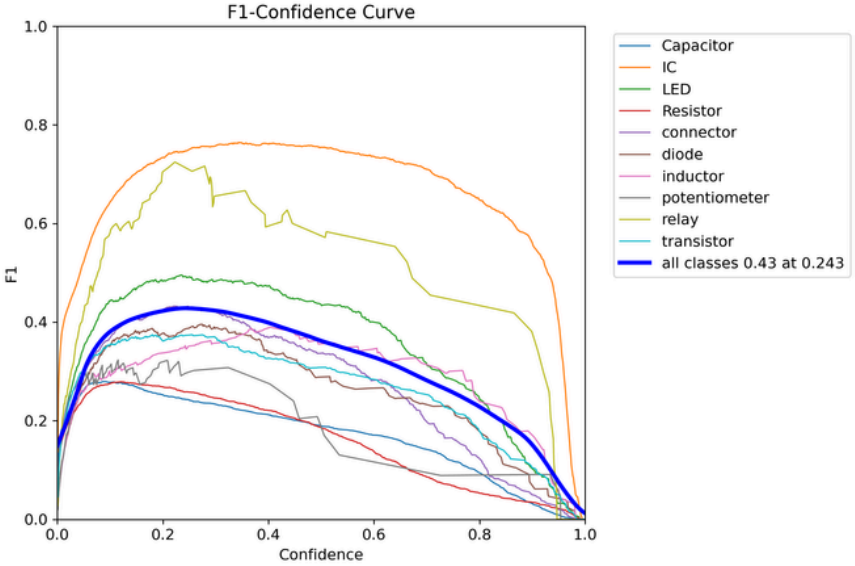
**Poor Performance:** The model struggles significantly with "Capacitor" (only 14% correct), "LED" (only 0.01% correct, which seems like a typo, perhaps it is 0.1), and "Resistor" (16% correct).

**Common Confusions:**

- Capacitors are often mistaken for background.
- ICs are confused with potentiometers and transistors.
- LEDs are frequently misclassified as ICs.
- Resistors are confused with connectors.
- Connectors are often labeled as background.
- Diodes are confused with inductors and potentiometers.
- Many classes are incorrectly identified as "background" (e.g., Capacitor, Connector, Transistor), suggesting the model might be overly biased towards the background class or struggles to detect objects when they are present.

**Class Imbalance (Potential Issue):** The strong performance on "Background" and the high number of misclassifications into "Background" for other classes might suggest a class imbalance in the training data, where "background" samples are much more prevalent, or the model has learned to classify objects as background when uncertain.

### *B. F1 Curve:*

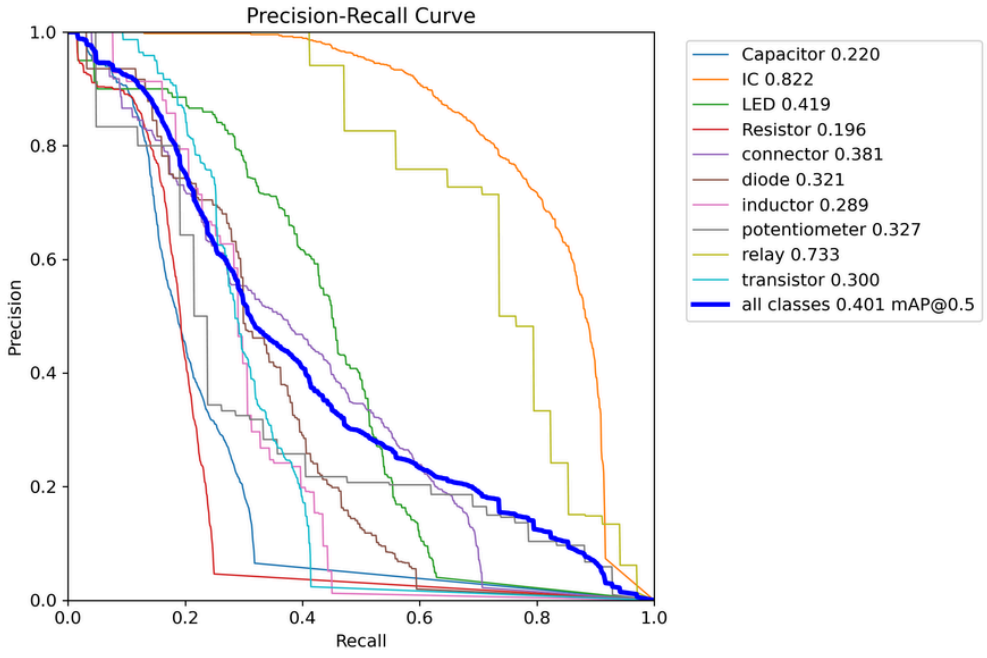


**Key Observations and Insights from This Specific F1-Confidence Curve:**

1. **Optimal Overall Threshold:** The model achieves its best average F1-score (0.43) when the confidence threshold is set to approximately 0.243. This means that if you want to optimize the overall balance of precision and recall, you should set your model's decision threshold around 0.243.
2. **Varying Performance Across Classes:**
   - **IC (Orange Line):** This class stands out with the highest F1-score, peaking around 0.75 at a relatively low confidence threshold (around 0.2-0.3). This suggests the model is very good at identifying ICs.
   - **Relay (Light Blue Line):** This class also shows good performance, with an F1-score peaking around 0.6-0.7.
   - **LED (Green Line):** The LED class also shows a relatively high F1-score, peaking around 0.6-0.7.
   - **Lower Performing Classes:** Many other classes (e.g., Capacitor, Resistor, Diode, Inductor, Potentiometer, Transistor, Connector) have significantly lower F1-scores, generally peaking below 0.4.
     - **Capacitor (Blue Line):** This class has a very low F1-score that drops off quickly, indicating poor performance and sensitivity to the confidence threshold.
     - **Resistor (Red Line):** Similar to Capacitor, it shows poor performance.
3. **Sensitivity to Threshold:**
   - For classes like "IC" and "Relay", the F1-score remains relatively high over a broader range of confidence thresholds before dropping off. This implies that the model's predictions for these classes are more stable.
   - For classes with lower F1-scores (e.g., Capacitor, Resistor), the curves are often steeper or consistently low, indicating that their F1-score is highly sensitive to the threshold or consistently poor regardless of the threshold.
4. **Trade-offs at Different Thresholds:**

- If you choose a very low confidence threshold (e.g., close to 0.0), the model will make many predictions, potentially increasing recall but also introducing many false positives (lowering precision and thus F1).
- If you choose a very high confidence threshold (e.g., close to 1.0), the model will only make predictions when it's very certain, potentially increasing precision but missing many true positives (lowering recall and thus F1). This is evident by all curves converging to 0 F1-score at a confidence of 1.0.

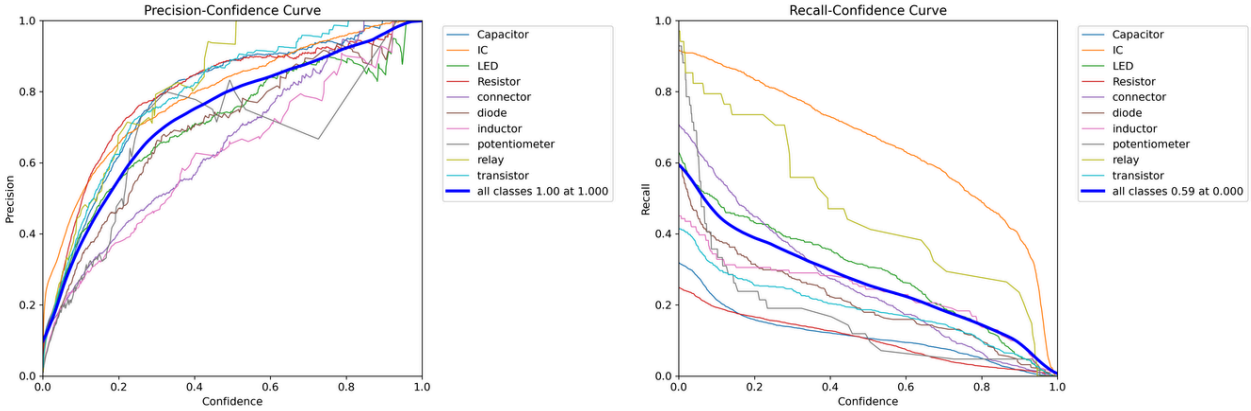
***C. Precision-Recall Curve:***



**Key Observations and Insights from The PR Curve:**

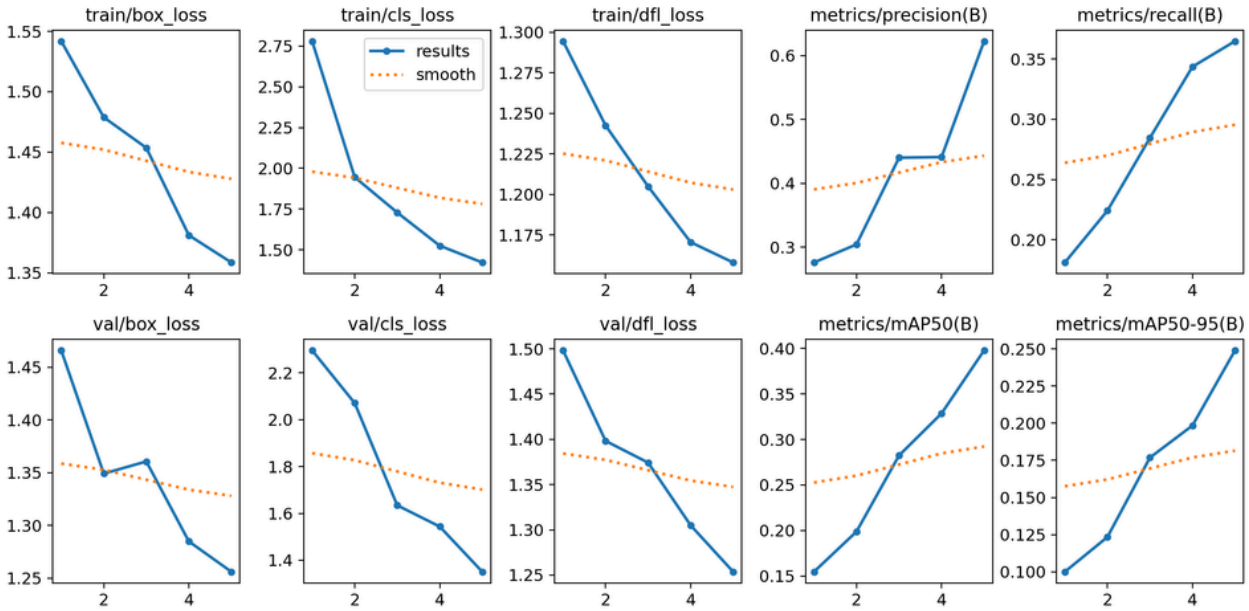
1. **Strong Classes:** The model performs exceptionally well for "IC" and "Relay", maintaining high precision even as recall increases. This suggests the model has learned robust features for these classes.
2. **Weak Classes:** The model struggles significantly with "Capacitor" and "Resistor", among others. Their PR curves are very close to the bottom-left corner, indicating that the model either has low precision (many false positives) or low recall (missing many true positives), or both, for these categories.
3. **Overall Model Performance:** The overall mAP of 0.401 is decent but indicates room for improvement, especially for the weaker classes. The overall curve shows that as recall increases, precision drops steadily, which is expected, but the rate of drop is quite significant, particularly after recall values of around 0.3-0.4.
4. **Actionable Insights:**

- For "IC" and "Relay", the model is performing quite well.
- For classes with low AP (e.g., Capacitor, Resistor), efforts should focus on:
  - **More training data:** Specifically, more diverse and representative examples of these components.
  - **Improved feature engineering:** Designing features that better distinguish these classes.
  - **Addressing class imbalance:** If these are minority classes, techniques like oversampling, undersampling, or using focal loss could be beneficial.
  - **Analyzing misclassifications:** Looking at where the model makes errors (using the confusion matrix) can provide clues.




**Relationship between the curves and overall model performance:**

- **Precision-Confidence and Recall-Confidence** show the components that make up the F1-Confidence curve. The F1-score is a balance of precision and recall.
- **IC and Relay** are strong performers across all metrics (high precision, high recall, high F1, high AP).
- **Capacitor and Resistor** are consistently weak performers (low precision, low recall, low F1, low AP), indicating significant challenges for the model in detecting and correctly classifying these components.
- The "**all classes**" lines on both confidence curves clearly illustrate the trade-off. To get high precision (Precision-Confidence curve approaching 1.0), you need a high confidence threshold, but this will drastically reduce your recall (Recall-Confidence curve approaching 0.0). Conversely, to get high recall (Recall-Confidence curve starting high), you need a low confidence threshold, but this will reduce your precision.




**Overall Result Analysis:**

Based on these plots, the model training is progressing very well:

- **Losses are consistently decreasing** on both training and validation sets, indicating that the model is learning effectively and not simply memorizing the training data (no signs of overfitting yet in these 5 epochs).
- **Performance metrics (Precision, Recall, mAP)** are all significantly increasing on the validation set, which means the model is getting better at its core task of accurately detecting and classifying objects.

**10. Scalability for Real-Time Inspection**

Discuss and demonstrate code optimizations or batching for real-time or near real-time PCB inspection scenarios.

```python
# Real-time detection using YOLOv8

# Load your trained model
model = YOLO('runs/detect/train2/weights/best.pt')
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    results = model(frame)[0]
    for box in results.boxes:
        x1, y1, x2, y2 = map(int, box.xyxy[0])
        label = model.names[int(box.cls[0])]
        cv2.rectangle(frame, (x1, y1), (x2, y2), (0,255,0), 2)
        cv2.putText(frame, label, (x1, y1-10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255,0,0), 2

    cv2.imshow("YOLOv8 PCB Detection", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

[ ] Python

## *For Streamlit Deployment:*

Create:
- "***requirements.txt***" file for listing Dependencies
- "***app.py***" file for Deployment Code

Install dependencies:
***pip install -r requirements.txt***

Run using following command:
***python -m streamlit run app.py***

Or simply
***streamlit run app.py***

**requirements.txt**

```
streamlit
opencv-python
numpy
matplotlib
ultralytics
pandas
```

## ***app.py***

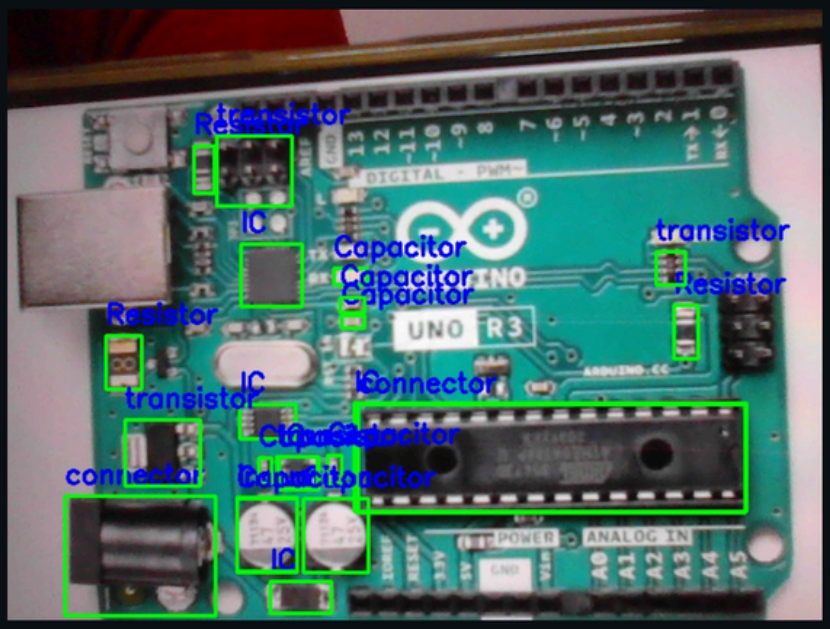
```python
import streamlit as st
import cv2
from ultralytics import YOLO
import numpy as np

st.title("YOLOv8 PCB Component Detection")

# Load model once
@st.cache_resource
def load_model():
    return YOLO('runs/detect/train2/weights/best.pt')

model = load_model()

# Webcam or image upload
option = st.radio("Choose input source:", ("Webcam", "Image Upload"))

if option == "Webcam":
    run = st.checkbox('Start Webcam')
    FRAME_WINDOW = st.image([])
    cap = None

    if run:
        cap = cv2.VideoCapture(0)
        while run:
            ret, frame = cap.read()
            if not ret:
                st.warning("Failed to grab frame")
                break
            results = model(frame)[0]
            for box in results.boxes:
                x1, y1, x2, y2 = map(int, box.xyxy[0])
                label = model.names[int(box.cls[0])]
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0,255,0), 2)
                cv2.putText(frame, label, (x1, y1-10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255,0,0), 2)
            FRAME_WINDOW.image(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
        if cap:
            cap.release()
else:
    uploaded_file = st.file_uploader("Choose an image...", type=["jpg", "jpeg", "png"])
    if uploaded_file is not None:
        file_bytes = np.asarray(bytearray(uploaded_file.read()), dtype=np.uint8)
        img = cv2.imdecode(file_bytes, 1)
        results = model(img)[0]
        for box in results.boxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            label = model.names[int(box.cls[0])]
            cv2.rectangle(img, (x1, y1), (x2, y2), (0,255,0), 2)
            cv2.putText(img, label, (x1, y1-10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255,0,0), 2)
        st.image(cv2.cvtColor(img, cv2.COLOR_BGR2RGB), caption="Detected Components", use_column_width=True)
```

# YOLOv8 PCB Component Detection

Choose input source:

- (•) Webcam
- ( ) Image Upload

[x] Start Webcam



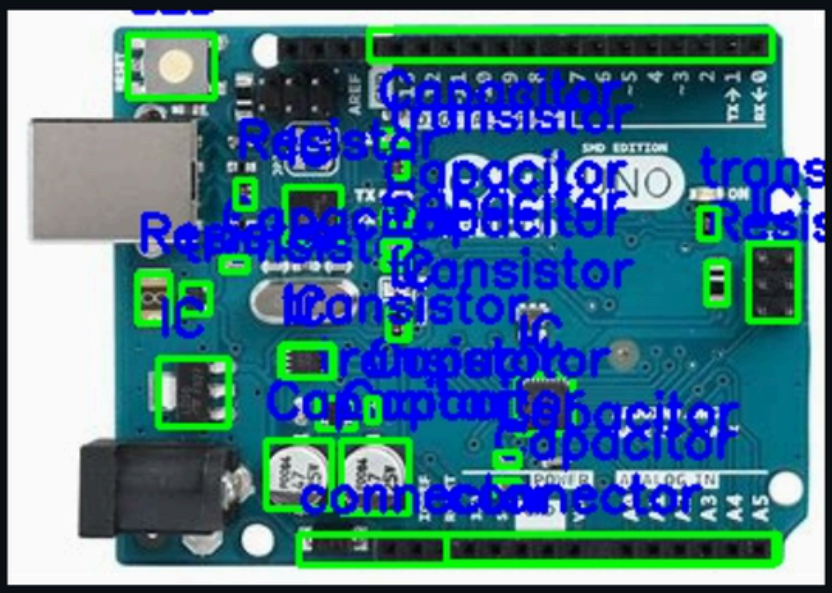

Choose input source:

- ( ) Webcam
- (•) Image Upload

Choose an image...

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG
Browse files

uno.png 194.5KB




Detected Components

## *Outcomes:*

- A functional prototype that can detect and classify basic PCB components in real time.
- Immediate visual feedback with annotated bounding boxes and labels for each detected component.
- A modular codebase that can be extended with advanced features, such as more advanced & large scaled machine learning-based classification or support for additional component types.
- The project talks about 3 different methodologies to detect the PCB components in real time:
  a. **Template Matching**
  b. **NumPy Logic**
  c. **Deep Learning Model**
- This is in the increasing order of accuracy where Template Matching is the least reliable method, NumPy Logic is constraints to Environmental Conditions (e.g. Lighting & Distance from Detector Camera) & Deep Learning Model is the most reliable with higher training epoch & large dataset training.

## *Significance & Future Outlook:*

This project demonstrates the feasibility of using computer vision for automated PCB inspection. It provides a foundation for further development, including integration with highly trained deep learning models for higher accuracy and deployment in industrial settings for large-scale quality control.

## *Conclusion:*

By leveraging computer vision techniques, this project automates and improve the reliability of PCB component inspection, paving the way for smarter and more efficient electronics manufacturing processes.

## *References:*


1. ***Dataset Used:***
   **roboflow:**
   **workspace:** rahul-lmk51
   **project:** mydataset-ohm6o-zlegu
   **version:** 1
   **license:** CC BY 4.0
   **url:** https://universe.roboflow.com/rahul-lmk51/mydataset-ohm6o-zlegu/dataset/1
2. YOLOv8 ( Source: https://docs.ultralytics.com/models/yolov8/ )


| Model | Filenames | Task | Inference | Validation | Training | Export |
|---|---|---|---|---|---|---|
| YOLOv8 | `yolov8n.pt` `yolov8s.pt` `yolov8m.pt` `yolov8l.pt` `yolov8x.pt` | Detection | ✅ | ✅ | ✅ | ✅ |
| YOLOv8-seg | `yolov8n-seg.pt` `yolov8s-seg.pt` `yolov8m-seg.pt` `yolov8l-seg.pt` `yolov8x-seg.pt` | Instance Segmentation | ✅ | ✅ | ✅ | ✅ |
| YOLOv8-pose | `yolov8n-pose.pt` `yolov8s-pose.pt` `yolov8m-pose.pt` `yolov8l-pose.pt` `yolov8x-pose.pt` `yolov8x-pose-p6.pt` | Pose/Keypoints | ✅ | ✅ | ✅ | ✅ |
| YOLOv8-obb | `yolov8n-obb.pt` `yolov8s-obb.pt` `yolov8m-obb.pt` `yolov8l-obb.pt` `yolov8x-obb.pt` | Oriented Detection | ✅ | ✅ | ✅ | ✅ |
| YOLOv8-cls | `yolov8n-cls.pt` `yolov8s-cls.pt` `yolov8m-cls.pt` `yolov8l-cls.pt` `yolov8x-cls.pt` | Classification | ✅ | ✅ | ✅ | ✅ |

Performance

Detection (COCO) | Detection (Open Images V7) | Segmentation (COCO) | Classification (ImageNet) | Pose (COCO) | OBB (DOTAv1)

See Detection Docs for usage examples with these models trained on COCO, which include 80 pre-trained classes.

| Model | size (pixels) | mAP$^{val}$ 50-95 | Speed CPU ONNX (ms) | Speed A100 TensorRT (ms) | params (M) | FLOPs (B) |
|---|---|---|---|---|---|---|
| YOLOv8n | 640 | 37.3 | 80.4 | 0.99 | 3.2 | 8.7 |
| YOLOv8s | 640 | 44.9 | 128.4 | 1.20 | 11.2 | 28.6 |
| YOLOv8m | 640 | 50.2 | 234.7 | 1.83 | 25.9 | 78.9 |
| YOLOv8l | 640 | 52.9 | 375.2 | 2.39 | 43.7 | 165.2 |
| YOLOv8x | 640 | 53.9 | 479.1 | 3.53 | 68.2 | 257.8 |