Notes on "Printed Circuit Board Solder Joint Quality Inspection Based on Lightweight Classification Network"

Ahmad Choudhry

February 3, 2025
https://ietresearch.onlinelibrary.wiley.com/doi/epdf/10.1049/csy2.12101

1 Introduction

These notes explore the paper *"Printed circuit board solder joint quality inspection based on lightweight classification network"* by Zhang *et al.* We (i.e., our team) intend to apply these ideas to a **real-time soldering robot project**, where the robot can inspect each joint right after soldering, decide if it is defective, and attempt corrections. Our roadmap includes first creating a simple *binary* good/bad model, then building up to more advanced *multi-class defect detection* with real-time feedback.

2 Context and Motivation

Solder joints are critical connections on a PCB. Defective joints can cause intermittent or outright failures. Manually inspecting every joint can be slow and prone to human error, so **automatic visual inspection** is a desirable solution.

Zhang *et al.* propose a pipeline with two main parts:

1. A specialized **segmentation** method (the "Select Joint" approach) to locate each solder joint on the PCB.
2. A **lightweight classification network** ("MobileXT") that determines whether the extracted soldered joint is normal (i.e good) or belongs to a specific defect category.

3 Paper's Proposed Pipeline: Overview

- *Stage 1:* **Solder Joint Segmentation** using morphological and color-based thresholding in the HSV color space.
- *Stage 2:* **MobileXT Classification** which combines a **CNN** (for local features) and a **Transformer** component (for global context).

These two steps together form a system that can achieve very high accuracy while staying small enough to run efficiently on typical GPUs or even on embedded devices.

4 Stage 1: Solder Joint Segmentation ("Select Joint")

We now go step by step through their segmentation process.

4.1 High-Level Idea of Use-case For Our Project

We are given PCB image containing many solder joints from the Vision System. We want to extract sub-images ("patches"), each containing exactly one solder joint. Once we have those patches, we can feed them into a classification model instead of running detection on the entire large image which would increase computational complexity and time taken for response

4.2 Histogram Equalization

We first apply **histogram equalization** to make the image's overall brightness more uniform. Imagine we had an image that was very dark in some areas and bright in others. Histogram equalization redistributes pixel intensities (balancing contrast frequencies) so that we do not have extreme shadows or highlights. It can help the subsequent thresholding steps pick out solder more consistently.

4.3 BGR to HSV Conversion

Standard images come in **BGR** or **RGB** format. However, **HSV** (Hue, Saturation, Value) is more convenient for separating color tones.

- **Hue**: The *type* of color (e.g., red, green, or blue).
- **Saturation**: How intense or grayish the color appears. How vivid the colour appears. A low saturation is a washed-out pixel.
- **Value**: The brightness level. High value is bright, low value is dark.

Solder joints often have particular *Hue* and *Value* traits that differ from the rest of the PCB background. Hence, we convert our image into HSV.

4.4 Otsu's Thresholding on Hue Channel

What is Otsu's Method? Otsu's method is a technique for finding a **single** intensity threshold that separates an image into *foreground* and *background* by maximizing the difference between these two sets. See WikiPedia Article. While it may sound intimidating, the core idea is:

1. We guess a threshold T for the pixel intensities.
2. Pixels with intensity *less than or equal* to T form one group (Group 1), and pixels *greater than* T form another group (Group 2).
3. We measure how *distinct* these two groups are by using a formula called the *between-class variance*.
4. We adjust T to see where this between-class variance is maximized.

Mathematically, if our hue channel is considered a grayscale image I, and $I(i,j)$ is the intensity at pixel (i,j), we define:

$$\omega_1(T) = \frac{\text{number of pixels } (i,j) \text{ with } I(i,j) \leq T}{\text{total pixel count}}, \quad \omega_2(T) = 1 - \omega_1(T).$$

$$\mu_1(T) = \text{mean intensity of pixels } \leq T, \quad \mu_2(T) = \text{mean intensity of pixels } > T.$$

$$\mu_{\text{total}} = \text{mean intensity of all pixels.}$$

The **between-class variance** is:

$$\text{Var}_{\text{between}}(T) = \omega_1(T)\,[\mu_1(T) - \mu_{\text{total}}]^2 \; + \; \omega_2(T)\,[\mu_2(T) - \mu_{\text{total}}]^2.$$

Otsu's algorithm quickly tries all possible T (e.g., from 0 to 255 in an 8-bit image) and picks the one that maximizes $\text{Var}_{\text{between}}(T)$.

How Does This Help Us? The authors apply Otsu's threshold on the **Hue channel** to decide which pixels might correspond to solder (foreground) versus the PCB substrate or other areas (background). Because solder often has a specific *Hue* range, Otsu's method can *automatically* find a good cutoff for that hue distribution.

4.5 Low-Threshold Gating on Value Channel

Now, we also look at the **Value channel** (brightness). While Hue can separate color, the brightness might also be a clue. If we want to capture areas that are *shiny* or *reflective*, we might say:

$$\text{If } V \geq T_V, \text{ keep the pixel as potential solder.}$$

This is called a **gating** step because we are effectively gating or filtering out pixels that do not meet a brightness criterion. (The paper calls it "low threshold gating" if it is focusing on eliminating super-dark regions or ensuring we only keep bright-enough pixels.)

Why Combine Hue and Value? Just thresholding Hue alone might include certain reflections that are not solder, or it might exclude some solder in weird lighting. By also checking Value, we strengthen our confidence that the pixel is truly solder material.

4.6 Merging Masks & Median Filtering

After we create two separate binary masks (one from the Hue threshold, one from the Value threshold), we **merge** them with a logical AND or OR (depending on how the authors configure it). For instance:

$$\text{CombinedMask}(i,j) = \text{MaskHue}(i,j) \ \wedge \ \text{MaskValue}(i,j).$$

But thresholding alone can produce small "specks" of noise. Hence, the authors do a **median filter**:

$$\text{FilteredMask}(i,j) = \text{median}\left\{\text{CombinedMask}(u,v) \ \mid \ (u,v) \text{ in a small neighborhood around } (i,j)\right\}.$$

This step basically *smooths* the binary image so isolated 1-pixel or 2-pixel outliers vanish.

4.7 Contour Detection & Bounding Boxes

Finally, we detect *contours* in the cleaned mask. Each connected blob presumably corresponds to a solder joint. Using `findContours` (OpenCV) or a similar function, we can get the bounding rectangle for each blob:

$$(x_{\min},\, y_{\min},\, \text{width},\, \text{height}).$$

We discard outliers (extremely large or small boxes) and keep the rest. These boxes tell us where to *crop* the original image, producing small patches focused on each solder joint.

5 Stage 2: MobileXT Classification

Now that each solder joint is isolated, we feed that *cropped patch* (resized to something like 256×256) into a **classification network**. The network used by the Research Paper is **MobileXT**, a small but powerful hybrid that merges CNN and Transformer elements.

5.1 Why a Hybrid CNN-Transformer?

- **CNNs**: Good at local pattern detection. Convolution filters pick up edges, corners, small shapes.
- **Transformers**: Use *attention* to capture *global relationships* (e.g., how different parts of the image relate to each other, even if they are far apart).

MobileXT tries to keep the parameter count low (in the range of 1–5 million parameters), so it can be run quickly and still achieve high classification accuracy.

5.2 MobileXT Overview

MobileXT stands for *Mobile* (small, efficient) + X (cross-covariance) + T (Transformer). This section in the original paper has the following components:

5.2.1 CNN Backbone

They begin with a **convolutional stem** (like a 3×3 conv with stride 2) and then multiple *inverted residual* blocks (inspired by MobileNet). See Figure 3 in the original paper. A 3x3 conv with stride 2 means a convolutional layer that uses a 3x3 filter (kernel) to slide across the input data, but with a stride of 2, which means the filter jumps ahead by two pixels at each step instead of one, effectively reducing the size of the output feature map by downsampling the input data. The inverted residual blocks do the following:

1. **Channel expansion**: Multiply the channels by some factor.
2. **Depthwise convolution**: Apply a separate convolution filter for each channel (reducing parameter count).
3. **Channel projection**: Bring the channels back down to a smaller number.
4. **Skip connection**: Add the original input if shapes match.

You can look more into MobileXT CNN architecture online.

5.2.2 Cross-Covariance Attention (XCA)

The **Transformer** portion uses an *attention* mechanism that tries to identify which parts of the image are relevant to other parts. However, the authors do not use a standard *self-attention* (which can have a high computational cost of about $O(n^2 d)$). Instead, they use **XCA (Cross-Covariance Attention)**, which we will expand on in Section **??**.

5.2.3 Fusion Layer

After we get local (CNN) features and global (XCA) features, MobileXT merges them (e.g., by concatenation or addition) and then applies another 3×3 convolution or pointwise convolution to *fuse* everything into a single coherent feature map. Finally, it does **global average pooling** and a **fully connected layer** to output classification logits (the final defect categories).

6 Dataset and Experiments

6.1 Data Collection

The authors took 93 PCBs, used their *Select Joint* approach to crop out **1,804** solder-joint patches, and labeled each patch in **7** categories: *Normal, Miss, Bridging, Tip, Littletin, Polytin, Rosin.*

They split the dataset 80%/10%/10% for train/val/test. They then trained MobileXT (and other comparison models like ResNet, MobileNet, ViT) with standard training parameters (batch size=128, momentum=0.9, weight decay=0.005, etc.).

6.2 Results

They found that **MobileXT-XXS** (the smallest version) achieved about **99.1% accuracy** on the test set, outperforming bigger networks but with only **1.27 million** parameters. They also used *Grad-CAM* to visualize what regions the model focuses on, showing tight attention on actual solder shapes.

7 Applying These Ideas to Real Time PCB Soldering System

We want to build a system that:

1. Inspects solder joints (and eventually corrects them) in real time, or near-real time.
2. Starts with a simpler **binary classification** (good vs. bad) to get a baseline. We could aim for around 70% accuracy initially.
3. Expands to more categories (e.g., bridging, missing solder, insufficient solder, etc.) once we collect more data.

Let me outline a plan, step by step:

7.1 Step 1: Data Collection and Labeling

- **Gather Raw PCB Images:** We set up a consistent camera rig that captures the board under stable lighting.
- **Apply Morphological Segmentation:** We can replicate the paper's approach (histogram equalization, HSV thresholding, median filtering, contour detection) to isolate each solder joint.
- **Label Joints:** For now, we can label them as *good* or *defective* if we only want a binary classification. Eventually, we can add more detail (the exact defect type).

7.2 Step 2: Baseline Classification Model

- If CoCo does not recognize the solders, we can train a small lightweight CNN (like *MobileNetV2* or *MobileNetV3*) on the cropped patches. See online 'Medium' Articles on MobileNet and open source codes.
- If we have limited images, we should use data augmentation (random flips, rotations, slight brightness changes) to improve generalization.
- Our target might be ∼70% to 80% accuracy on a test set for *good vs. bad.* This is enough to demonstrate feasibility.

7.3 Step 3: Scaling Up to More Classes or More Precision

Once we are comfortable with the pipeline, we can:

- Introduce *5-7 defect categories* (similar to the paper's: bridging, missing solder, tip, etc.).
- Potentially adopt or adapt *MobileXT* from the research paper, which might yield higher accuracy and better real-time performance.
- Expand the dataset with more boards, more lighting conditions, or different camera angles to ensure the model is robust.

7.4 Step 4: Real-Time Feedback and Correction

Our ultimate goal might be to *automate rework or correction*:

- If the system spots a bridging defect, it can prompt a soldering iron or hot air station to reflow or remove extra solder.
- If it detects insufficient solder, it can quickly dispense a bit more solder and reflow.

Achieving this in *1 second or less* would require:

- Keeping the classification model **lightweight** so inference is fast.
- Possibly implementing parallel processing or hardware acceleration (e.g., using an embedded GPU like Jetson Nano or similar).
- Ensuring that the mechanical soldering apparatus can respond quickly and precisely in hardware.

7.5 Potential Challenges

- **Data Scarcity:** We might not find a big public dataset for through-hole or surface-mount solder joints. We have to capture our own images.
- **Lighting Variation:** Reflective metallic surfaces can cause glare and inconsistent appearances. Good lighting control is key.
- **PCB Variation:** Different PCB finishes (HASL, ENIG, OSP, etc.) or solder types (lead-free vs. leaded) might require re-tuning thresholds or additional training data.

8 Further Explanation of the Math

If you are unaware of some of these Image Processing topics (they are discussed in CE 4TN4 which you are encouraged to take), here is some more depth:

8.1 Morphological Thresholding in More Detail

Thresholding is a basic method in image processing to decide which pixels belong to a region of interest (foreground) and which do not (background). For example, we might say:

$$\text{Mask}(i,j) = \begin{cases} 1, & \text{if } H_{i,j} \le T_H \text{ and } V_{i,j} \ge T_V, \\ 0, & \text{otherwise,} \end{cases}$$

where $H_{i,j}$ is the Hue value at pixel (i,j), $V_{i,j}$ is the Value (brightness) at pixel (i,j). The thresholds T_H, T_V come from Otsu's method or manual tuning.

Why Morphology? After thresholding, we apply *morphological* operations like **median filtering** to clean up noise. A median filter basically replaces each pixel with the *median* of its surrounding neighborhood. If there is a small black or white "speck," it tends to vanish, improving segmentation quality.

Contour Detection: After we have a binary mask, **contour detection** using OpenCV will find connected blobs. Each blob is presumably *one solder joint*. A bounding rectangle around that blob lets us crop the relevant area from the full image.

9 Conclusions and Our Roadmap

The research paper essentially performs:

- **Morphological Segmentation** (Stage 1) helps isolate each solder joint quickly, reducing the subsequent classification problem to small patches.
- **Lightweight Hybrid Network** (MobileXT, Stage 2) yields **high accuracy** (over 99% in their experiments) while staying compact enough for real-time usage. Uses XCA, inverted residual convulotional strides and transfomer blocks in their architecture.

For our **PCB soldering robot project**, here is a proposed plan:

1. **Collect Our Own Dataset**: We will photograph boards, apply the morphological approach, label each joint. For a start, we might do just *Good vs. Bad* classification.

2. **Train a Baseline Model**: Something like a small CNN. Aim for around 70%–80% accuracy to confirm the pipeline works.

3. **Advance to Multi-Defect Classification**: We can adopt or adapt *MobileXT* for bridging, missing solder, etc., as we grow our labeled dataset.

4. **Real-Time Correction**: Integrate this into the robot's control loop so that it *immediately* tries to fix defects by adding or removing solder.

We expect challenges (e.g., limited open-source data for *through-hole* solder joints, lighting variations, and different PCB finishes). But the pipeline proposed by Zhang *et al.* provides a clear blueprint for a robust, high-accuracy system that remains practical to deploy. We anticipate that by following and expanding on these methods, we can achieve an automated solder-joint inspection and correction system in the near future.