Table of Contents

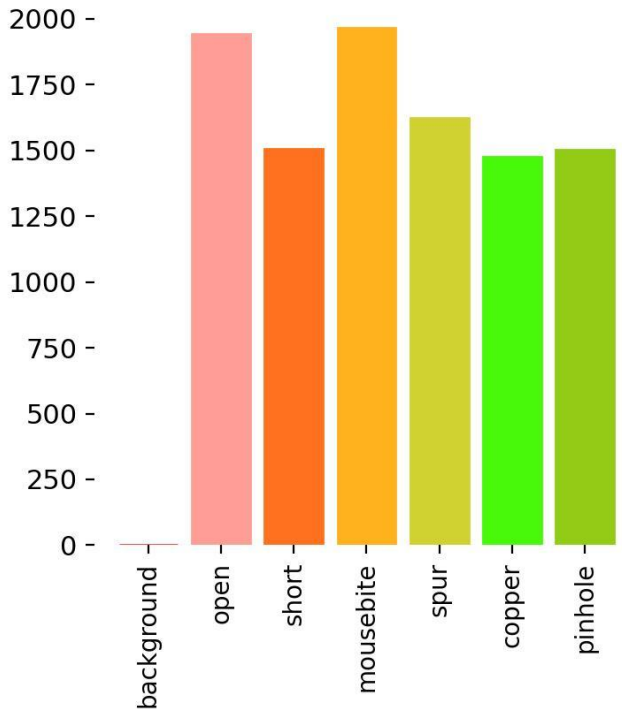
To IITD-AIA-FSM,

Understanding the dataset and Exploratory Data Analysis (EDA)

About the Dataset

The collection comprises 1,500 picture pairs, each consisting of a template image devoid of defects and a tested image that has been aligned and annotated with the positions of the six most prevalent PCB defects: open, short, mouse bite, spur, pinhole, and spurious copper.
There are a total of 10013 labels in the dataset whose distribution is as follows:

1. Background - 0 (not used in entire dataset)
2. Open - 1942
3. Short - 1506
4. Mouse bite - 1965
5. Spur - 1625
6. Pinhole - 1474
7. Spurious copper - 1501



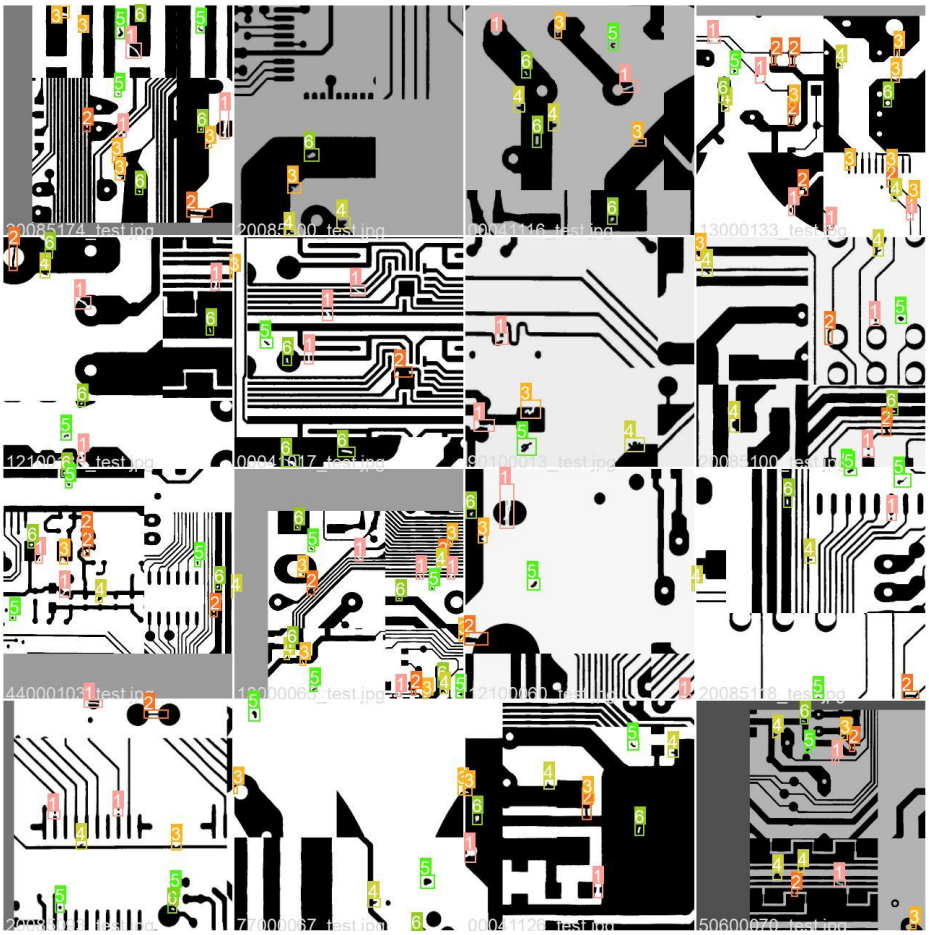
Distribution Histogram of the categorical data (Categories vs instances of occurrence)

About the image

The linear scan CCD used to capture each image in this collection has a resolution of about 48 pixels per millimetre. The original size of the template and the tested image is around 16k x 16k pixels. They are then divided into several 640 × 640 pixels sub-images using a cropping process, then aligned using template matching methods. However, the 1500 defective PCB images and their annotation files will be primary resources for training our Deep Learning Model.

About the image annotations

We utilize an axis-aligned bounding box with a class ID for each flaw in the tested photos. Each annotated image owns an annotation file with the same filename, e.g.00041000_test.jpg, 00041000_temp.jpg and 00041000.txt are the tested image, template image and the corresponding annotation file. Each defect on the tested image is annotated as the format: x1, y1, x2, y2, type, where (x1,y1) and (x2,y2) are the top-left and the bottom-right corner of the bounding box of the defect and type is an integer ID that follows the matches:
0-background (not used), 1-open, 2-short, 3-mouse-bite, 4-spur, 5-copper, 6-pin-hole.



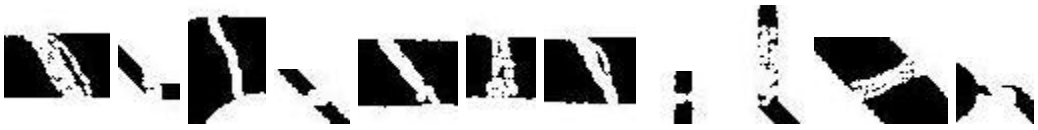
Encoded annotations of the training data (representation ID as mentioned above)

Cropping and extracting the defects

I have created an image cropping tool in order to crop the defects from the dataset

```
notation_file_path =
r"Dataset\PCBData\group{f_name}\{f_name}_not//".format(f_name=12000)
notation_file_dirs = os.listdir(notation_file_path)

saving_folder =
r"Dataset\PCBData\group{f_name}\{f_name}_defects//".format(f_name=12000)
saving_folder_dirs = os.listdir(saving_folder)

image_file_path =
r"Dataset\PCBData\group{f_name}\{f_name}//".format(f_name=12000)
image_file_dirs = os.listdir(image_file_path)

for imag in image_file_dirs:
    if os.path.isfile(image_file_path+imag):
        im = Image.open(image_file_path+imag)
        f,e = os.path.splitext(image_file_path+imag)
        if f.endswith("test"):
            f = f.split("//")[-1].split("_")[0]
            text_file = open(notation_file_path+f+".txt")
            text_lines = text_file.readlines()
            count = 0
            for line in text_lines:
                nf,ne = os.path.splitext(saving_folder)
                x1,y1,x2,y2,anno = [int(s) for s in line.strip().split()]
                cropim = im.crop((x1,y1,x2,y2))
                count = count+1

print(nf+str(anno)+"_"+f+"_"+str(annotate(anno))+"_"+str(count))

cropim.save(nf+str(anno)+"_"+f+"_"+str(annotate(anno))+"_"+str(count)+".jpeg"
, "jpeg")
```

The output for open circuit defect extraction from the first group are as follows:



Similar insights can be observed when iterating through other groups of dataset. This helps in understanding the shape and size of the bounding boxes from the annotation files.

Image Preprocessing

It is essential to pre-process the image in order to delete any kind of noise or unwanted detailing in the dataset. This helps in better evaluation as well as conducting mathematical operations or image functions using OpenCV to the images. However, in our case, we require our Machine learning (or deep learning) algorithm to learn every single detail about the data that has been

fed to it. If we process the data into something that is not distinguishable for the algorithm, it will not learn all the relevant features that are necessary to output a valid and confident prediction. In other words, here preprocessing the data for smoothening the surface, or reducing size, or even applying gaussian filter (let's say) will act as an unwanted noise, instead of helping us for better results.
So, what can be done?

Preparing images for training

Current folder structure:

```
└──PCBData
    ├──group00041
    │   ├──00041
    │   ├──00041_not
    ├──group12000
    │   ├──12000
    │   └──12000_not
    ├──group12100
    │   ├──12100
    │   └──12100_not
    ├──group12300
    │   ├──12300
    │   └──12300_not
    ├──group13000
    │   ├──13000
    │   └──13000_not
    ├──group20085
    │   ├──20085
    │   └──20085_not
    ├──group44000
    │   ├──44000
    │   └──44000_not
    ├──group50600
    │   ├──50600
    │   └──50600_not
    ├──group77000
    │   ├──77000
    │   └──77000_not
    ├──group90100
    │   ├──90100
    │   └──90100_not
    └──group92000
        ├──92000
        └──92000_not
```

The desired folder structure which is an entire repository of all the training images and validation-cross-validation images:

```
└──Dataset
      └──PCBData
          ├──images
          ├──labels
          └──validation
```

Where `/images` contain 1200 test images, `/labels` contain 1500 annotation files, and `/validation` contains 300 test images which are not taken for training the dataset. That means, `/labels` will have all the annotation files corresponding to images present in `/images` and `/validation` directories.

Processing the folder structure

This has been obtained by writing the following python script:

```
import glob
import shutil
import os

folder_names = [00041, 12000, 12100, 12300, 13000, 20085, 44000, 50600, 77000,
90100, 92000]

for folder_name in folder_names:
    src_dir = f"Dataset/PCBData/group{folder_name}/{folder_name}"
    dst_dir = f"Dataset/PCBData/images"
    for jpgfile in glob.iglob(os.path.join(src_dir, "*_test.jpg")):
        shutil.copy(jpgfile, dst_dir)
    for txtfile in glob.iglob(os.path.join(src_dir, "*.txt")):
        shutil.copy(txtfile, dst_dir)
```

Creating the `/validation` directory is not mandatory, however, is often recommended to avoid testing the model on the trained dataset itself whilst training.

Why to change the folder structure?

In order to answer this question, let us understand how were the folders grouped in the first place. The images were grouped according to the contradiction in the novel module, termed Group Pyramid Pooling by the author. This is assumed to be categorized into the number of pixels that are being operated while feature scaling the neural network model.
However, this distinction is redundant for us because the model architecture we are going to use will be a lot deeper than the one the author of DeepPCB has used, hence we will be able to extract finer features enabling us to carry out experiments regarding the efficiency and effectiveness of our model.

Therefore, it will be much more efficient for us to just make two folders namely `/images` and `/labels`. This will help our server/local computer to recognize where all the images and their corresponding labels are located onto which we need to train our model. And just like that, we have prepared our dataset which can now be trained further.