Image Based Anomaly Detection - PCB Defect Localization and Identification

PCB Anomaly Detection

Tejas Guha, Hassaan Mastoor, Mani Pabba

ENEE 439D 0101

05/20/2023

Table of Contents

Contributions

Mani Pabba
Mani primarily contributed to the YOLOv5 approach, demo model, and labeling utility.
Hassaan Mastoor
Hassaan primarily contributed to the development of the image processing and visualization of the dataset.
Tejas Guha
Tejas primarily contributed to the development of the autoencoder/decoder model along with its performance analysis.

University of Maryland Honor Pledge:

I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination.

Team Signatures:

Mani Pabba

Hassaan Mastoor

Tejas Guha

I. Executive Summary

PCBs are integral parts of small and large electronics devices. They provide electronic control for everything from modern car keys to aircrafts. However, with the prevalence of automatic fabrication methods, there is an increased likelihood of defects. These come in many forms but to name a few: missing hole, mouse bite, open circuit, short, and spurious copper). Typically these defects are determined after the PCB has malfunctioned and then an experienced individual takes a close look at the board to identify the defect. However, we have determined that there is a suitable method that can detect select defects through machine learning and image processing techniques.

This report explores PCB anomaly detection of realistic defects within PCB boards provided by the Aerospace corporation. An autoencoder/decoder network is explored to generate novel non-defect versions of images for defect localization and detection. This report includes an end-to-end processing chain, including preprocessing, model evaluation, and post processing to determine defects from PCB images.

The dataset being utilized in this project is a mixture of defective/non-defective pairs of PCBs provided by the Aerospace Corporation, along with pairs extracted from the *DeepPCB* dataset (Tang, 2019). The overarching goal of this project is to create a system that can precisely locate defects in an input image of a PCB. This project differs from its previous iterations in its goal to create a functional product for lower-quality images that mimic the conditions this project is intended to solve; images taken by a user with a standard smartphone camera. That is why we focused on incorporating images of PCBs taken using a phone camera as opposed to images taken by high quality industrial-grade cameras as previous iterations of this project did. Additionally, we specifically focused on detecting “short” defects in contrast to previous iterations. This was because this particular type of defect was the most common and of the most interest to our corporate mentor - the Aerospace Corporation.

II. Introduction

Printable circuit boards (PCB) are fundamental components to many electronic devices where the quality of the device can have a meaningful impact on performance. With increasing PCB component density and complexity, manual inspection for defects becomes more and more inaccurate and straining. This project focuses on using image processing and machine learning to automatically detect and classify defects in PCBs with the goal of being more accurate than manual inspection.

A naive approach would be to capture a PCB with no defects and compare it to a newly fabricated one. Computing the XOR of each pixel on image 1 with the corresponding pixel on

image 2 yields an image where the differences are apparent. Then a machine learning algorithm can take the output image and classify the defect by type. This was an initial idea but we decided to add to this to better fit our application.

We then experimented with semantic segmentation to localize defects and classify them using YOLOv5 on the Kaggle dataset. In this approach we used pre-drawn and labeled bounding boxes around defects and then passed this data into YOLOv5. The benefit of using YOLOv5 is that this approach does not depend on a template to extract differences, but relies on pure train data where defect properties are learned, not the differences between a test and its respective template.

Finally, we experimented with a denoising convolutional skip autoencoder where the input is an image of a defective PCB and the autoencoder attempts to reconstruct the same image without any defects. This is the approach we ultimately decided to pursue as it seemed to generate the most promising results in the early stages of development.

This project can be split into two core aspects, image pre-processing and localization where PCB images are standardized and split into features for classification, and classification where PCB features are actually identified by a specific defect type.

III. Goals and Design Overview

The original goals of this project were to be able to both localize and detect defects within a PCB image. Not only did we strive to classify defects, but also figure out the location of defects, which is especially challenging due to the sheer component density of most PCBs. Additionally, localization and classification was to be done not only on online synthetic datasets, but the dataset provided by the Aerospace corporation, a real-world dataset consisting of several FGPAs, ADCs (Analog to Digital Converters), and other types of PCBs.

We have shifted our focus onto localization since most defect types provided by Aerospace corporation are very similar in nature, either excess solder or shorts. We have chosen to ignore defects related to components having too little or no solder as they are virtually invisible in two-dimensional images from a bird's-eye perspective of a PCB. Localization is more of a key issue since actual classification for these types of defects can easily be done by the human eye. Additionally, there are few defect classes to classify defects for the data provided.

Our design consists of three main components, preprocessing, autoencoder/decoder, and post processing/evaluation.

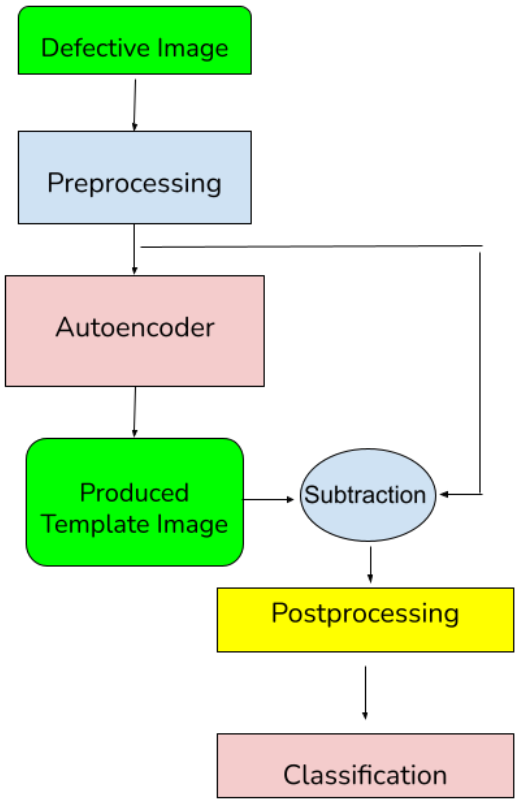


Figure 1: High Level System Design

Preprocessing

The first crucial step to implementing our algorithm would be to process the PCB images. Meaning, we needed to generalize the images to ensure the input to our defect detector had the least amount of erroneous noise as possible. After experimenting with many different procedures such as simply converting the images to gray scale and performing simple binarization, we opted for a method called Multi-Otsu Thresholding. The key difference between this method and simple binarization is that we can choose how many classes we want to separate the image into, allowing us to generalize our binarization technique to images with regions of multiple degrees of lighting. This produces an output image that is better representative of the PCB features we desire to emphasize. Before we could perform Multi-Otsu Thresholding, we needed to alter the color space of the image. First, we removed all of the green color space from the images as most of the PCBs in our dataset have green film over the top. Considering our application was focused on the pinholes and components, we reasoned that setting the green color space in RGB to zero did improve the quality of our output image. Next, we viewed the processed RGB image (where

G = 0) in the HSV color space. We saw that in the value color space (V) the image looked closest to highlighting the components we desired. Finally, multi-otsu thresholding was performed on this final image and we obtained a binarized version of the original PCB.

The result of this step is a list of processed 80x80 images which will be fed into the autoencoder.

Autoencoder/decoder

This is the core module of our design, this autoencoder model does the actual defect detection. This model is trained to generate a novel non-defective version of the inputted image.

Post Processing

The reconstructed version of the input-image is supposed to illustrate the same PCB board without any defects. The autoencoder reconstruction is produced by stitching together the 80x80 image outputs of the autoencoder. A difference image, produced by subtracting the input image with the autoencoder reconstruction, highlights the regions of the input image that are defective. This difference image goes through average filtering and basic threshold binarization which eliminates any noise in the difference image and only illustrates the predicted defective regions.

IV. Realistic Constraints

One economic constraint is cost in producing high quality datasets. One key issue we experienced was quality of images. With higher resolution image capture, such as the capture machines available at most PCB manufacturing plants, we would obtain better images with less variance due to environmental factors, such as light.

A manufacturability constraint we experienced was the variance in PCB boards. There are many different PCB layouts with fundamentally different components and as such defect types. This constraint makes it hard for the model to generalize all types of defects in detection.

Another constraint is the large variation in possible defects that may exist on a PCB board. For the scope of this project, we have limited the defects to be localized and classified to shorts between components in a PCB caused by excess solder. This particular type of defect is the most visible in the two-dimensional, lower quality images provided to us.

V. Engineering Standards

In terms of IEEE standards, our group had an organized google drive and Github repository containing all training data, collected data, and code.

One IEEE standard our group tried to approach was Standard P2894, Guide for an Architectural Framework for Explainable Artificial Intelligence. We put in effort to increase the transparency of out model, such displaying all substeps to our model, making it transparent what data we used to train out model, as well as justify what our model has "learned," such as how out model generally has a bias towards non-straight features to find defects.

Another IEEE standard tried to follow was Standard P3123, Standard for Artificial Intelligence and Machine Learning (AI/ML) Terminology and Data Formats. Our group emphasized maintaining our data set in such a way that it would be clearly documented which data corresponded to certain models.

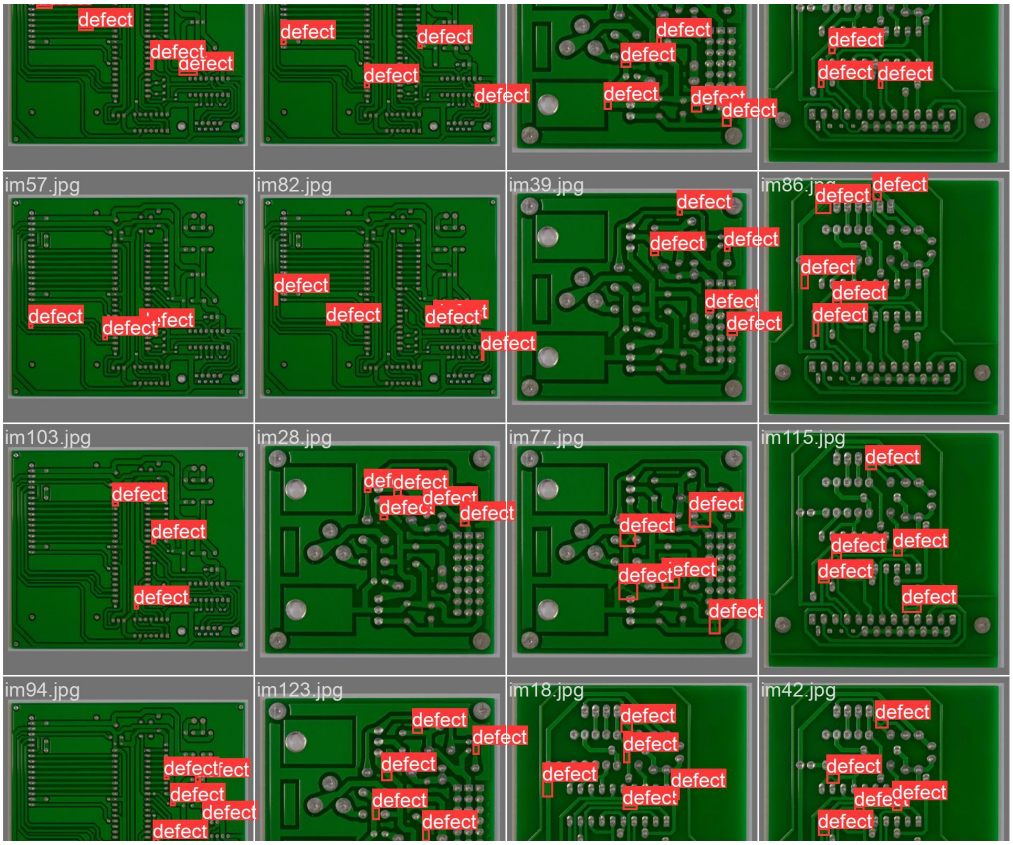
VI. Alternative Designs and Design Choices

Approach 1: Semantic Segmentation for Localization + Classification via YOLO + Style Transfer

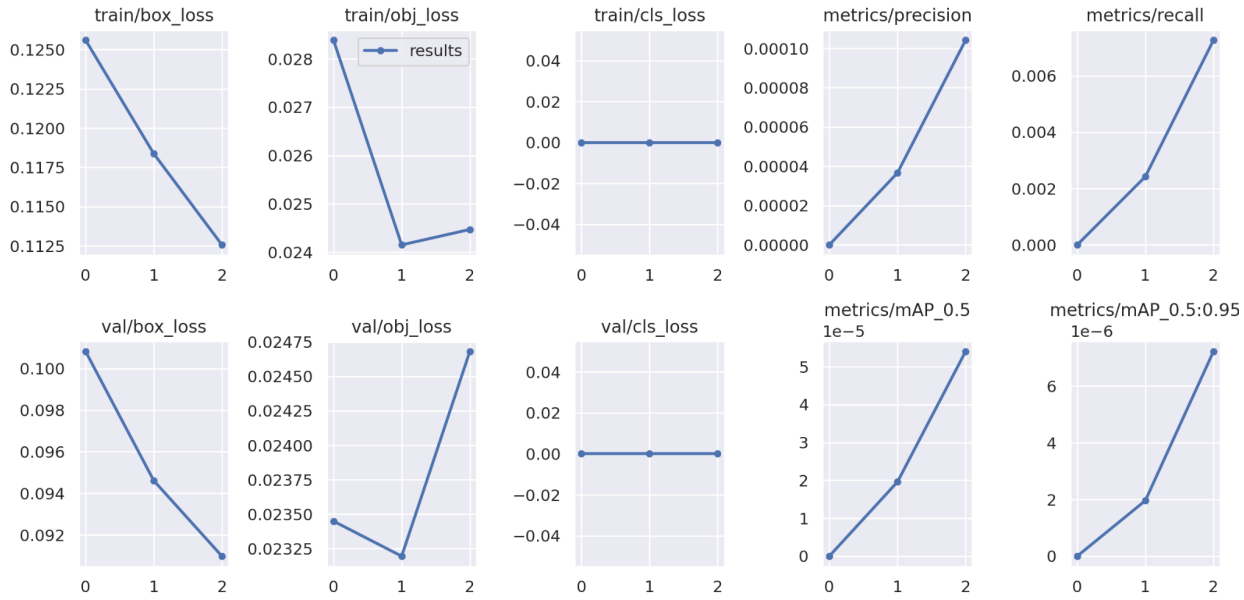
YOLO Training on Kaggle

With this approach, we use an algorithm capable of drawing bounding boxes with a classification (with classification accuracy) on defect features. We decided to use YOLOv5 as this approach does not depend on a template to extract differences, but relies on pure train data where defect properties are learned, not the differences between a test and its respective template.

This approach yielded in a bounding box loss of .1 and an object loss of .02 on the validation set. However, this model is only trained to detect the existence of a defect, not its specific classification as we want to generalize this to the Aerospace Corporation dataset.



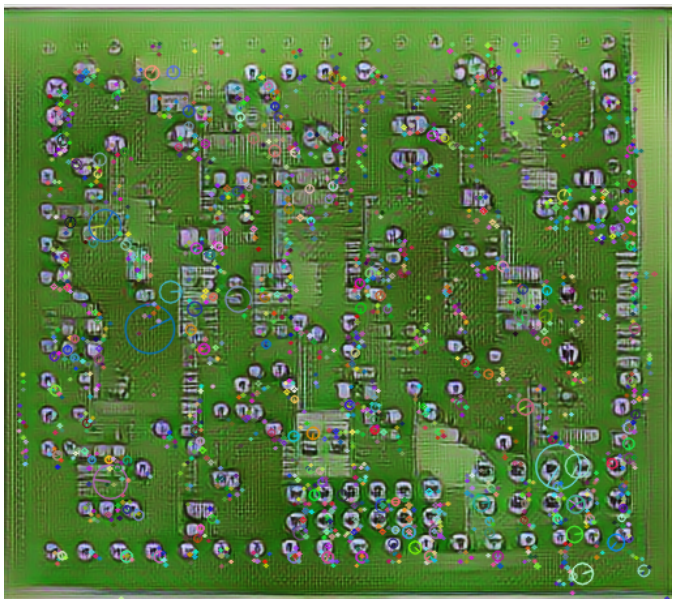
Labels on batch of validation images



Results graphs from training model

Generalization via Style Transfer

One big issue with this approach however is that the Aerospace Corporation data and Kaggle data images look very different and focus on different defect types. One approach we attempted was to use a style transfer model to transfer the aerospace data into an image that looks like the Kaggle data.



Aerospace PCB with Style transfer applied

However, even with this approach, we were not able to classify any defects on the style transferred images using the YOLO model. We think this is due to the vastly different defect types in the Aerospace data and the Kaggle data.

Data Sources:

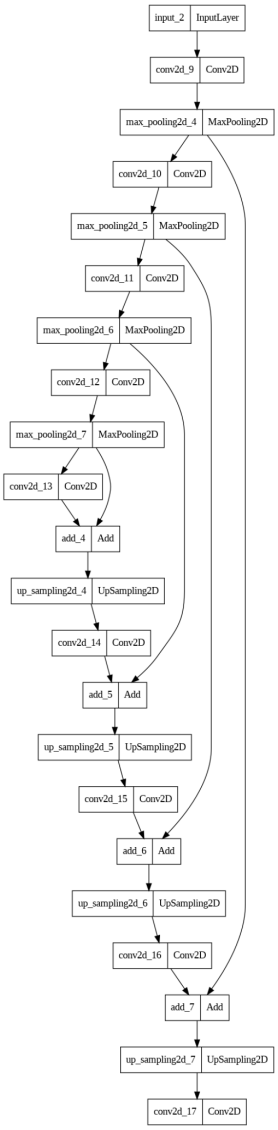
VII. PCB Synthetic dataset with 1386 images with 6 types of defects
https://www.kaggle.com/datasets/akhatova/pcb-defects

VIII. Aerospace Corporation dataset

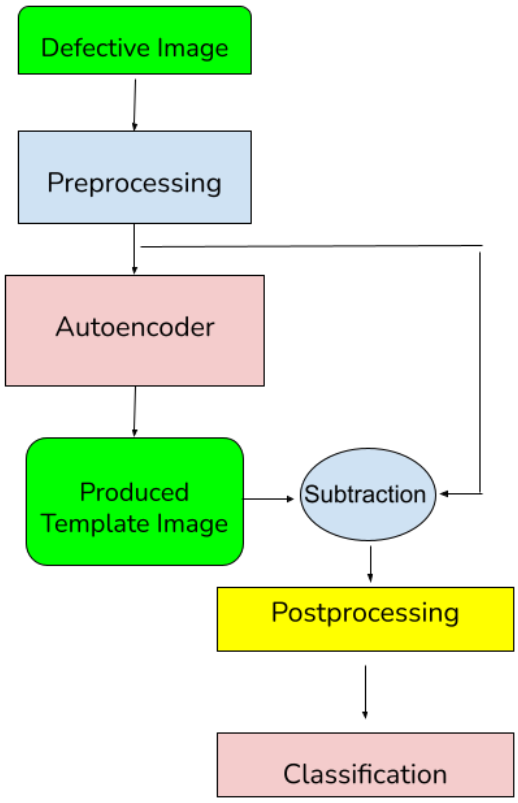
Approach 2: Denoising Convolutional Skip Autoencoder for PCB Reconstruction

In this approach, we use an autoencoder directly on the Aerospace Corporation dataset to generate a non-defect image from a defect image. Performing a bitwise XOR operation on the image would then yield the locations of defects. The initial version of this approach attempted to train two autoencoders where one autoencoder was trained on PCB images with defects and the other without them. Then, the outputs from each of the autoencoders would be compared to for

defect localization. However, this approach was determined to be pointless considering both autoencoders learned the identity function. Consulting scientific literature, a common approach to training autoencoders for anomaly detection is utilizing a defective image as the input and attempting to reconstruct a paired nondefective image (Khallian, Kim). To avoid overfitting produced by training with a limited amount of PCB layouts, we added salt and pepper noise to the defective training images. This approach was found to be promising. Images were trained on 80x80 regions of corresponding defective and non-defective PCBs using a synthetically derived dataset (DeepPCB) and images taken from by the authors on the PCBs provided by the Aerospace Corporation. The architecture of the autoencoder involves a 4-layer encoder and decoder with maximum pooling to decompose a binarized 80x80 image into a latent space of 5x5x4. Skip connections were added to the decoding layers to prevent gradient decay in the deeper layers of the autoencoder as it was initially observed when the encoder was built and used in literature (Kim).



Denoising Convolutional Skip Autoencoder for PCB Reconstruction



High Level Model Architecture

Data Sources:

1) Aerospace Corporation dataset
2) DeepPCB (dataset with defect and non-defect image pairs)

Approach 3: Defect Localization using XOR + Classification via CNN

This approach is an extension of the previous approach. One of the primary issues with the approach above is that at times it is quite difficult to identify the differences between the autoencoder reconstructed version of a defective PCB and the defective PCB itself even after a XOR operation. This is because the XOR always picks up errors from the autoencoder reconstruction which are often present on the outlines of each of the components within the PCB itself. Thus, we propose the utilization of a simple CNN that is able to classify a defect from the XOR between the autoencoder reconstruction and an input image. The number of layers and types of layers in the CNN would be adjusted based on the need for classification. We could

evaluate the accuracy of different neural network architectures and determine which one produces the best accuracy.

Data Sources:

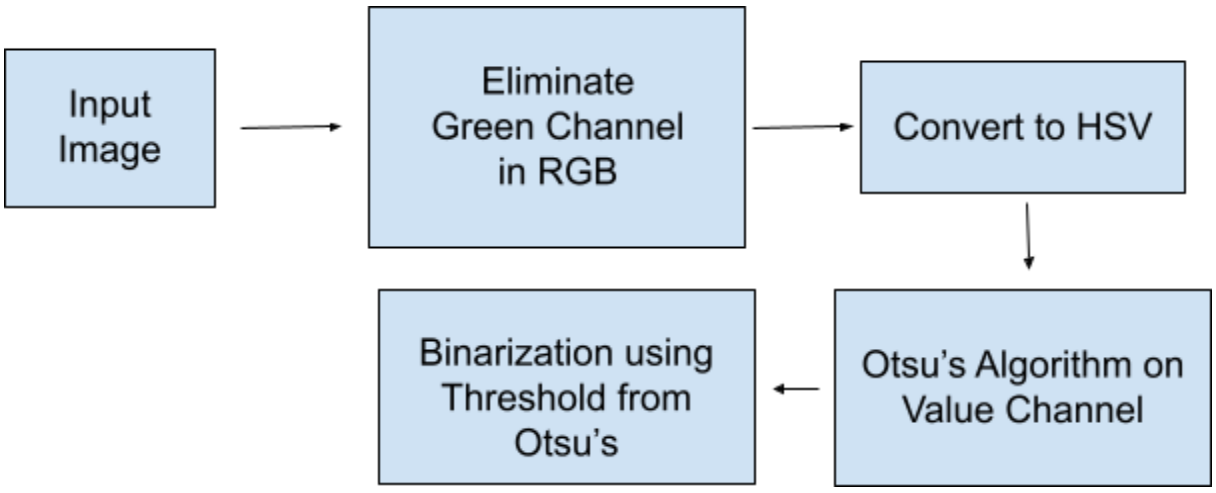
1) This paper outlines the use cases for XOR in regards to image processing https://www.iosrjournals.org/iosr-jce/papers/Vol17-issue2/Version-5/B017250715.pdf
2) Deep PCB with 1,500 image pairs with templates and aligned test images – useful if a template approach is taken where the differences are inputted into the model https://github.com/tangsanli5201/DeepPCB

VII. Technical Analysis for Systems and Subsystems

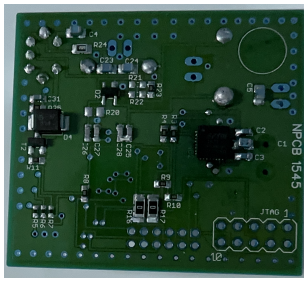
Preprocessing

The overall preprocessing chain consists of the following steps:

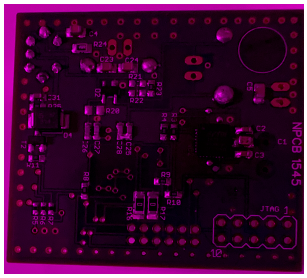
1. RGB based processing
2. HSV conversion and processing
3. Binary thresholding.



First we convert into the RGB color space and zero out the green channel. We do this since the background of most PCBs used are green. This helps us remove irrelevant features/background.

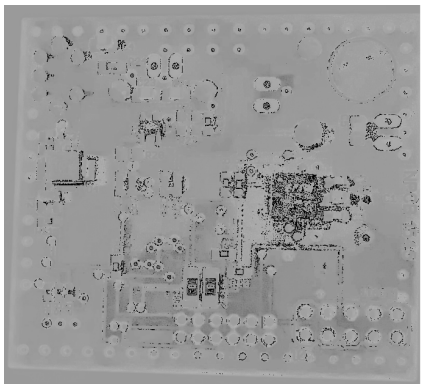


Original Image

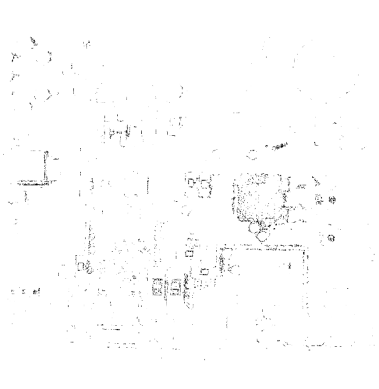


Green Channel Removed Image

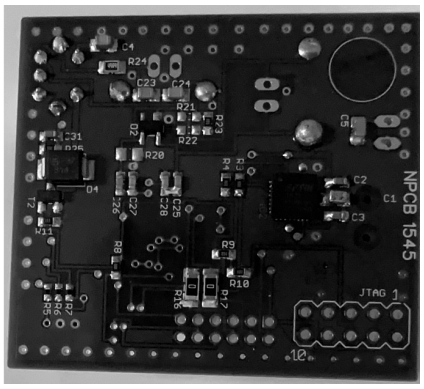
We then converted the RGB image to the HSV color space and used the V (value) channel for binarization. This was because the V channel directly corresponds to the intensity of a color and does not relate to the lighting of the color in the image nor the shade of color like the H and S channels do.



Hue Channel

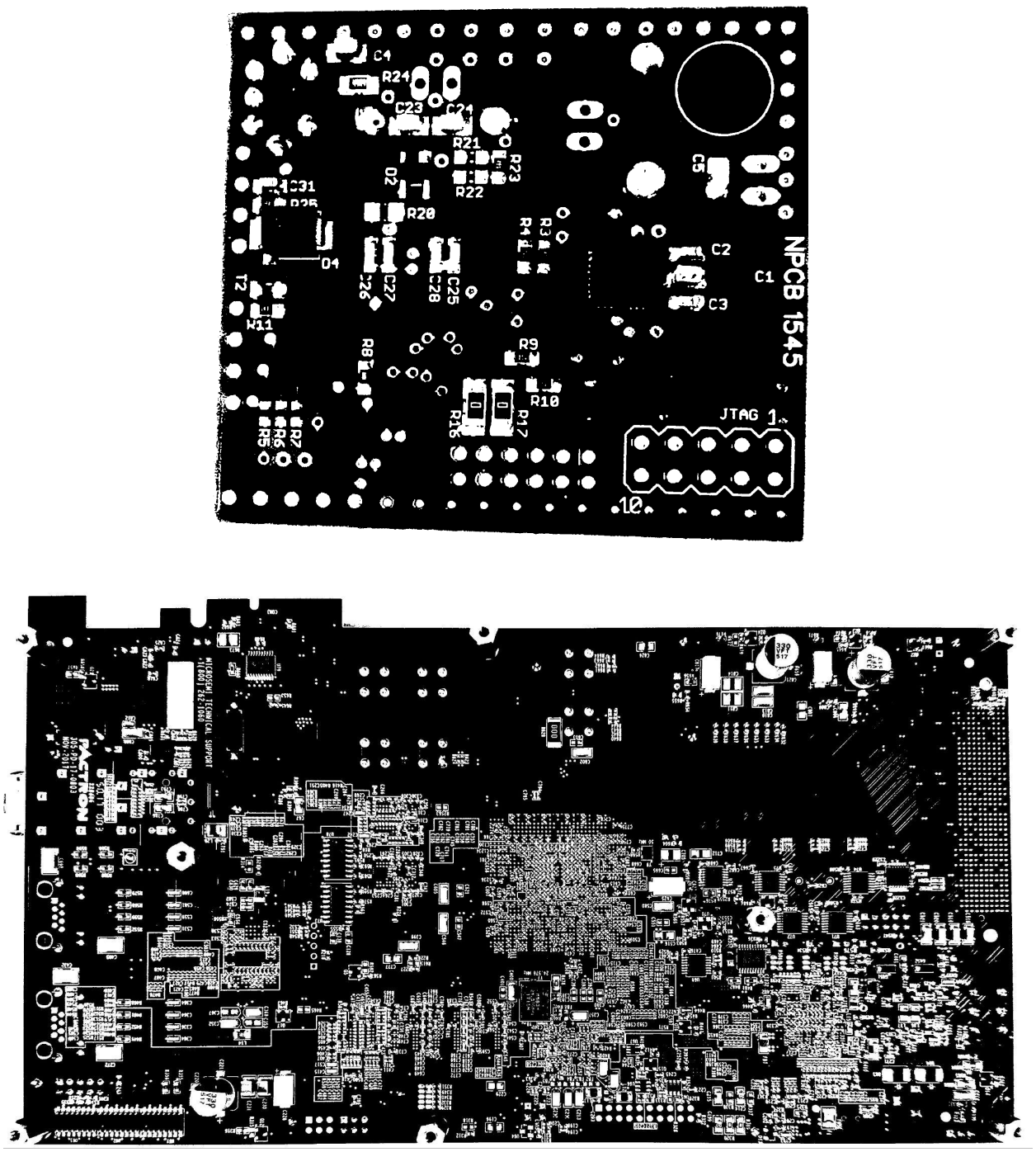


Saturation Channel



Value Channel

We then used Otsu's multi-thresholding algorithm for binarization. We opted to utilize the algorithm for 3-classes because examining the histogram of the value channel for various PCB images revealed that was the minimum number and optimal number of classes needed for binarization. Otsu's algorithm is a variance-based one that seeks to minimize the within-class variance of the ranges it chooses to represent a particular class (foreground and background).



Examples of Preprocessed Images Following Binarization

$$\sigma^2(t) = \omega_{bg}(t)\sigma^2{}_{bg}(t) + \omega_{fg}(t)\sigma^2{}_{fg}(t)$$

Within-class weighted variance formula used (Krishnan)

The class ranges for the foreground and background are computed by iteratively finding the two values (lower and upper thresholds) that minimize the optimization problem above. Values from 0 to the lower threshold, between the two thresholds, and the upper threshold to 255 represent the possible ranges that may be classified as the foreground or the background. This dynamic thresholding approach, versus a fixed constant, gives us a more robust binarization that can work in various lighting conditions.

Lastly, binarization was found to be an essential preprocessing step because it simplified the noisy features of a PCB image into either 0 or 1. It was found that without binarization it became difficult for the reconstruction autoencoder to converge properly or learn defects - the magnitude of noise present in the Aerospace noises made it difficult for the autoencoder to learn defects and binarization helped reduce that noise.

Autoencoder/decoder

We use an autoencoder which takes an 80x80 subimage and outputs an 80x80 version of the input without defects. We compress down to a latent space which ideally only stores key features for detection and then decode into a full 80x80 image. To prevent gradient decay due to the large number of layers in the autoencoder, we added skip connections in the autoencoder.

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 80, 80, 1)]	0	[]
conv2d_9 (Conv2D)	(None, 80, 80, 32)	832	['input_2[0][0]']
max_pooling2d_4 (MaxPooling2D)	(None, 40, 40, 32)	0	['conv2d_9[0][0]']
conv2d_10 (Conv2D)	(None, 40, 40, 16)	12816	['max_pooling2d_4[0][0]']
max_pooling2d_5 (MaxPooling2D)	(None, 20, 20, 16)	0	['conv2d_10[0][0]']
conv2d_11 (Conv2D)	(None, 20, 20, 8)	3208	['max_pooling2d_5[0][0]']
max_pooling2d_6 (MaxPooling2D)	(None, 10, 10, 8)	0	['conv2d_11[0][0]']
conv2d_12 (Conv2D)	(None, 10, 10, 4)	804	['max_pooling2d_6[0][0]']
max_pooling2d_7 (MaxPooling2D)	(None, 5, 5, 4)	0	['conv2d_12[0][0]']
conv2d_13 (Conv2D)	(None, 5, 5, 4)	404	['max_pooling2d_7[0][0]']
add_4 (Add)	(None, 5, 5, 4)	0	['conv2d_13[0][0]', 'max_pooling2d_7[0][0]']
up_sampling2d_4 (UpSampling2D)	(None, 10, 10, 4)	0	['add_4[0][0]']
conv2d_14 (Conv2D)	(None, 10, 10, 8)	808	['up_sampling2d_4[0][0]']

```
add_5 (Add)                    (None, 10, 10, 8)    0          ['conv2d_14[0][0]',
                                                                'max_pooling2d_6[0][0]']

up_sampling2d_5 (UpSampling2D)  (None, 20, 20, 8)   0          ['add_5[0][0]']

conv2d_15 (Conv2D)             (None, 20, 20, 16)   3216       ['up_sampling2d_5[0][0]']

add_6 (Add)                    (None, 20, 20, 16)   0          ['conv2d_15[0][0]',
                                                                'max_pooling2d_5[0][0]']

up_sampling2d_6 (UpSampling2D)  (None, 40, 40, 16)  0          ['add_6[0][0]']

conv2d_16 (Conv2D)             (None, 40, 40, 32)   12832      ['up_sampling2d_6[0][0]']

add_7 (Add)                    (None, 40, 40, 32)   0          ['conv2d_16[0][0]',
                                                                'max_pooling2d_4[0][0]']

up_sampling2d_7 (UpSampling2D)  (None, 80, 80, 32)  0          ['add_7[0][0]']

conv2d_17 (Conv2D)             (None, 80, 80, 1)    33         ['up_sampling2d_7[0][0]']

==================================================================================================
Total params: 34,953
```

Exact architecture for encoder decoder system

The autoencoder was trained using a MSE (minimum squared error) loss function with 1550 images from the Aerospace PCBs and 2000 images from the DeepPCB dataset. There are only 31 80x80 defect images from the Aerospace PCBs but in order to expand the training data we repeated multiple images with different types of salt-and-pepper noise. The noise to signal ratio of the training images was 0.3.

Aerospace Defect

Aerospace Defect Image with Noise

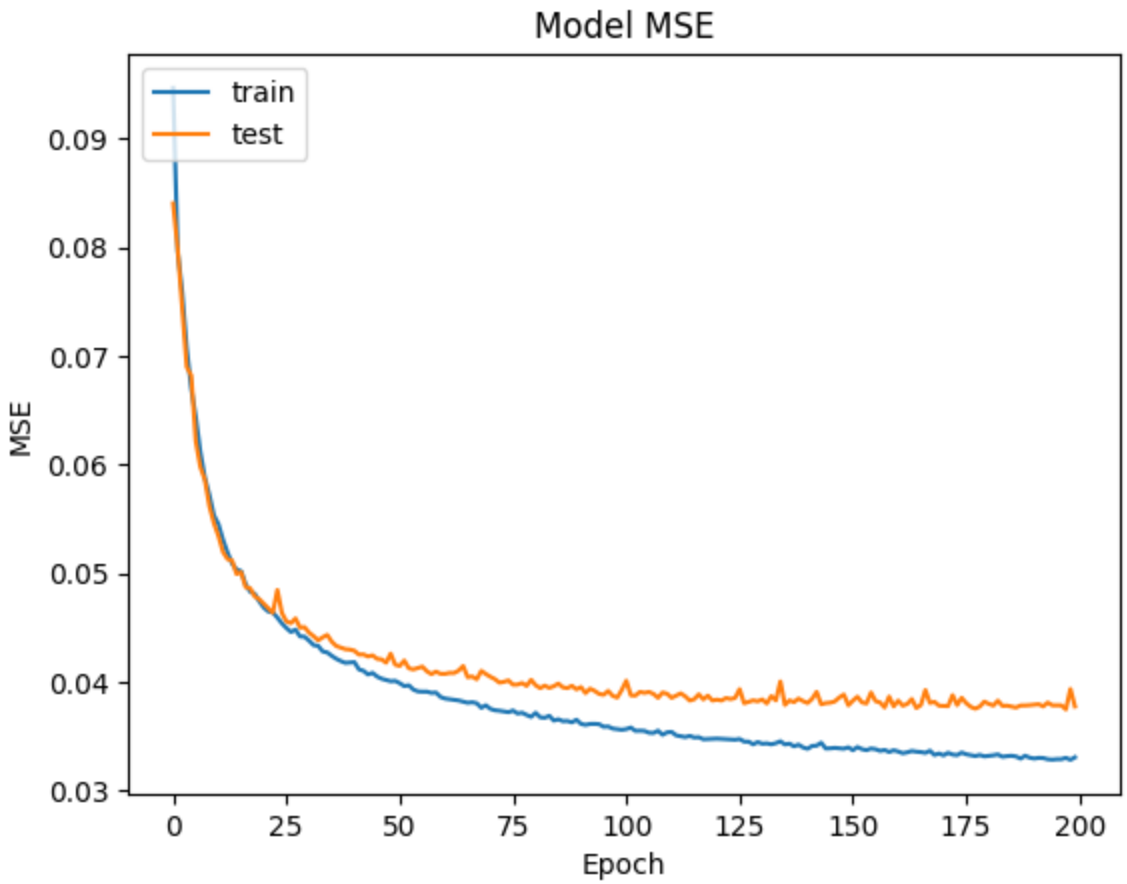
Aerospace No-defect

As can be observed above, the Aerospace defect/no-defect pairs often have variations in their alignments. This is problematic because the autoencoder may learn extraneous information as a result which inhibits its ability to perform well in its function of no-defect reconstruction. Since there are no precise bounding box annotations of the defects in the PCBs provided by the Aerospace Corporation, we attempted to solve this by developing our own image cropping program. This image cropping program takes two images of PCBs - one which is defective and one which isn't - and asks for an user to click on regions of the defective PCB that are defective.

The cropping program works to align two images of the PCBs and crops 80x80 images surrounding the user-selected defect. Obviously, this process is not perfect as the alignment does not always create identical defect/no-defect pairs. Our discrepancy in performance of the image cropping algorithm could be resolved by taking pictures of the PCBs in a more controlled manner where the images are spatially aligned when taken. Additionally, one could investigate if there are any means of improving the alignment process in the image-cropping program itself which could create better training data.

DeepPCB Defect with Noise

DeepPCB No-Defect

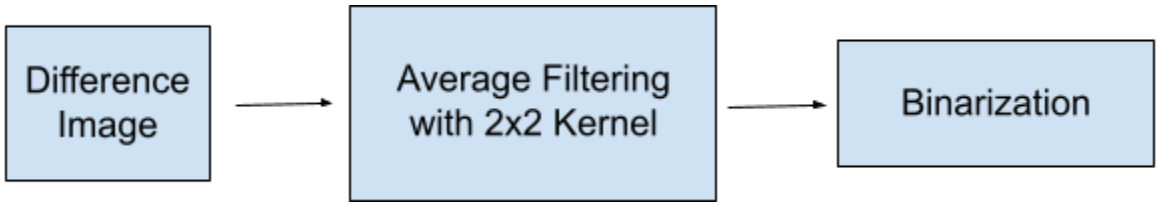


MSE of Autoencoder training on Combined Aerospace + DeepPCB Dataset

We chose the Mean-Squared Error (MSE) as our loss function because it was the intuitive choice as the purpose of the autoencoder is to minimize the difference between its reconstruction of a defect PCB and a non-defect version of the same PCB. The results above are after 200 epochs of training and it is quite obvious that the model has converged. Other loss functions such as “binary cross entropy” were considered especially since the pixels were either 0 or 1 but it was determined that MSE worked better empirically.

Postprocessing

The previous layer yields a list of 80x80 images. First, we stitch the images back together in the shape of the original input image. Next, we perform an average filter on the reconstructed output of the autoencoder using a 2x2 convolution kernel. The output is then binarized such that all values in the difference that are below 0.5 are labeled as a defect (a value of 1 in the differenced image) or nondefective (a value of 0 in the differenced image).



The reason we decided to perform an average filtering on the differenced image in the first place is to reduce the proportion of defects that are errors in the autoencoder's reconstruction of an input image. Typically, these errors show up as outlines of individual components in an input image of a PCB. The autoencoder produces a blurrier version of an input image of a PCB which causes the edges of the input image to show up when taking the difference of the autoencoder reconstruction with the input. Performing average filtering and then thresholding helps reduce these errors and makes it clearer where the system predicts the defects in an input image to be.

Differenced Image before Post-processing

Differenced Image after Post-processing

Effect of Post-processing on Differenced Image between Autoencoder reconstruction and input image

White indicates predicted defect

VIII. Design Validation for System and Subsystems

To validate the preprocessing subsystem, we visually inspected the images to make sure that only key components were in the processed images, including defect features.

To validate the performance of our autoencoder, we opted to measure the precision and recall rate of the autoencoder when reconstructing a test input image. This was done by first determining the segmentation mask that defines the defect in a test defect and non-defect PCB pair. Simple image differencing between a test PCB image and its template was done to achieve this.

Precision is a measurement of the false positive rate of a classification system; in this case it was determined by counting the number of overlapping defect pixels between the post-processed differenced image and the segmentation mask and then dividing by the number of pixels that were identified as a defect in the post-processed differenced image. $Precision = a/b$ where a is the number of pixels that are both defects in the segmentation mask and the post-processed difference image and b is the number of the pixels labeled as defects in the post-processed difference image.

Recall is a measurement of the false negative rate of a classification system; in this case it was determined by the following equation: $Recall = a/(a + c)$ where a is the number of pixels that are both defects in the segmentation mask and c is the number of pixels that were labeled as defects in the segmentation mask but not in the post-processed difference image.

In situations where classifying a defective part of a PCB as non-defective part is costly, the recall rate should be as large as possible. In situations where classifying a non-defective part of a PCB as defective is costly, the precision rate should be as large as possible.

Steps for Analyzing System Performance

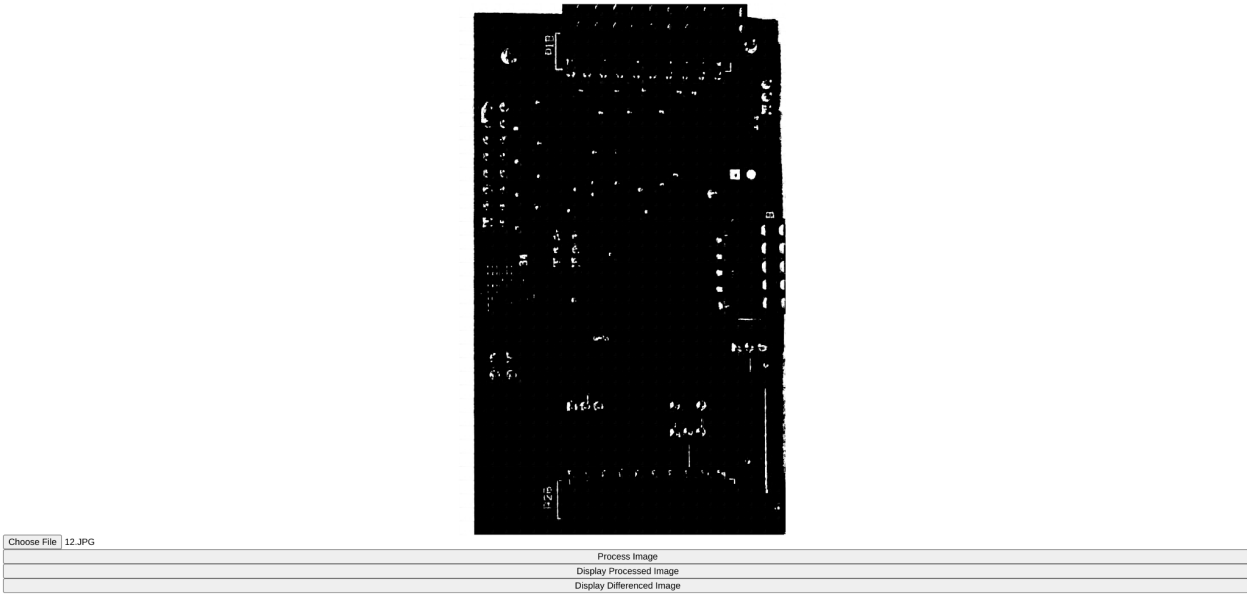
From top row to bottom row, left to right*: Test Image, Template Image, Defect Segmentation Mask, Post-processed Differenced Image, True-Positive of Prediction (a), False-Positive of Prediction (b - a), False-Negative of Prediction (c).*

Recall *for this image is 0.404 and* ***Precision*** *is 0.833.*

IX. Test Plan and Overall Performance Achieved

Test Plan

We plan to test the overall system by visually inspecting the defect accuracy of the system on the images we gathered from the PCBs sent by the Aerospace corporation, along with PCB images from the *DeepPCB* dataset. To help with this/have a final working product, we put together a website where users can upload images and see where defects are located. This was implemented in Javascript/HTML/CSS and the actual model evaluation is done on a backend API implemented using Flask.

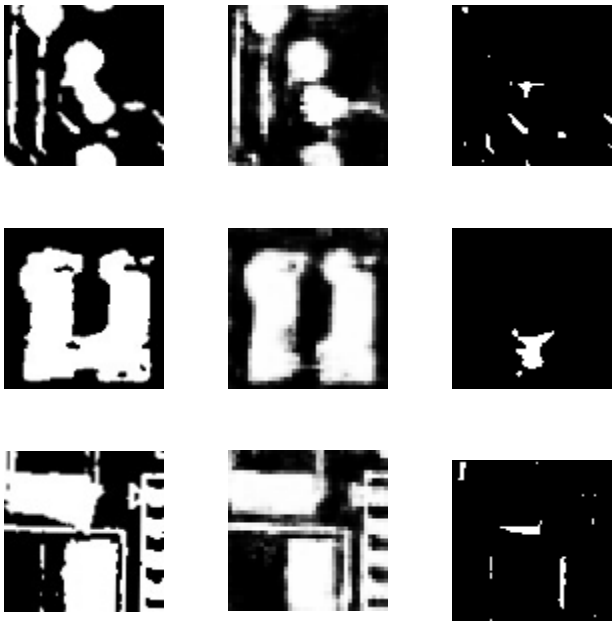


Website preview – AN image can be uploaded and processed results can be seen. The actual image processing / model evaluation is set up through an API (via Flask).

Additionally, we plan on measuring the performance of our system by determining the recall and precision of our system in two datasets that we have not trained on: a set of 640x640 images and a set of 80x80 images of PCBs from *DeepPCB*. The reasoning why we decided against quantifying recall and precision on the images we took of the Aerospace PCBs is due to the smaller amount of data and the lack of alignment between template and defective images. The spatial variability between the template and test images would corrupt measurements of recall and precision and give results that do not accurately reflect the performance of the system.

Performance

Evaluating the performance of the system highlights its limited capability. While the autoencoder reconstruction performs well reconstructing images it was trained on correctly, it fails to successfully reconstruct images it was not trained on.

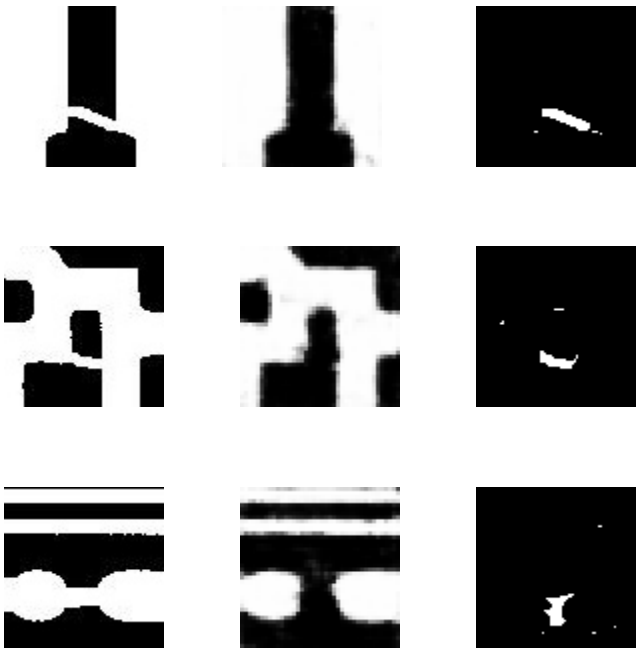


Aerospace PCB Defect Images and their Reconstructions

Above are three pairs of training images with defects, their reconstructions from the autoencoder, and the difference of the reconstruction from the defect image (from left to right) from the Aerospace PCBs. Note that the autoencoder performs well in its reconstruction of these 80x80 images. However, performing a difference operation between the training image and the autoencoder reveals that other artifacts in the images are also often present creating many false positives that can make it difficult for an user to correctly access which part of the image/PCB is the defect. These false positives are from errors in the autoencoder's reconstruction of non-defective regions of the PCB. These aberrations are exacerbated when reconstructing images with a high density of electrical components.

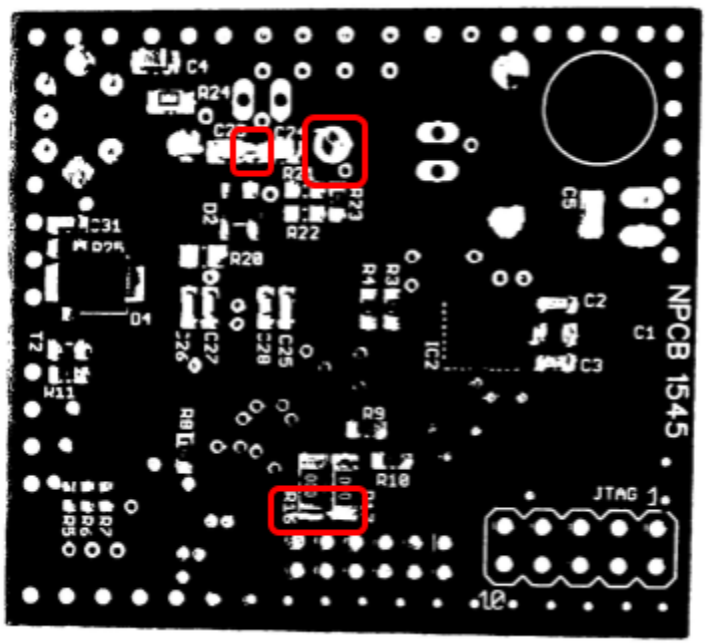
Reconstruction of a Training Image with a Defect Spanning Multiple Components

The image above is an example of a busier image of a PCB that the autoencoder would have to reconstruct in a realistic environment. The defect present is an extraneous piece of solder that connects multiple components beneath a chip. As is easily observed the reconstruction is quite a bit blurry when compared to the actual image with many components (such as the individual pins of the chip seemingly melded together). The difference reveals a confusing array of false positives with a central defect in the middle. A well-made CNN that scans the differences from the autoencoder reconstruction and the actual image may be able to distinguish the actual defect apart from the false positives. However, this reveals a central drawback of the autoencoder - since it was largely trained on images with only part of a single component it fails to properly reconstruct images that may contain multiple components.

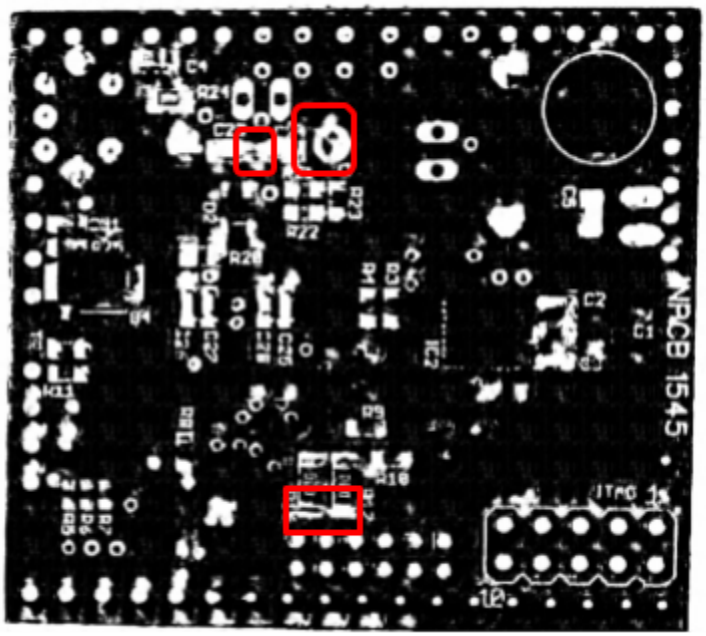


DeepPCB Defect Images and their Reconstructions

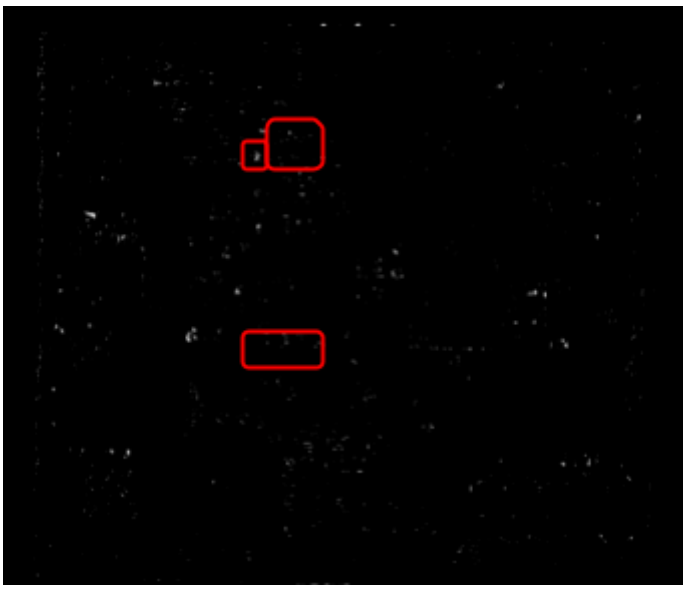
Above are the reconstructions of DeepPCB defect images and their differences with the original image. Note that unlike the Aerospace defect images, the differences lack very few false positives and the defect is extremely clear. This reveals that the autoencoder does a better job reconstructing the DeepPCB images than the Aerospace ones. This is probably because the images trained on from DeepPCB are extremely zoomed-in and only contain parts of components with much less noise than Aerospace. While binarization of the Aerospace images attempts to reduce the amount of noise, there are still irregularities present that are completely nonexistent in the DeepPCB dataset. These differences help explain the difference in performance of the autoencoder when reconstructing images from both sets.



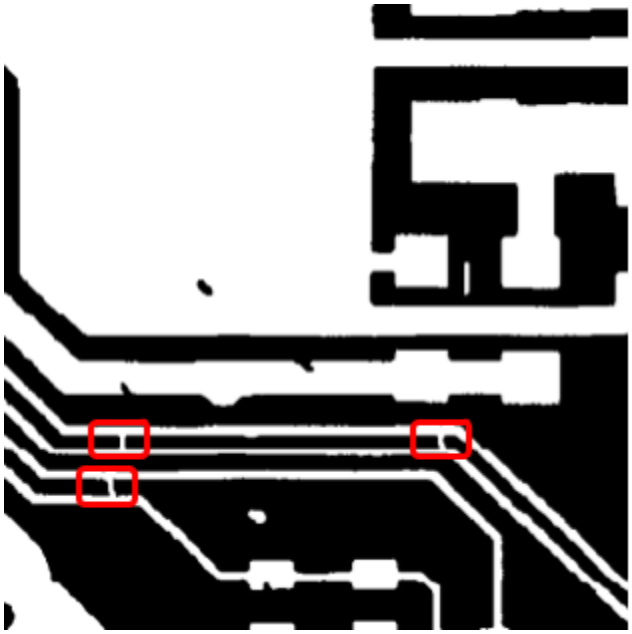
Defective PCB (Aerospace)



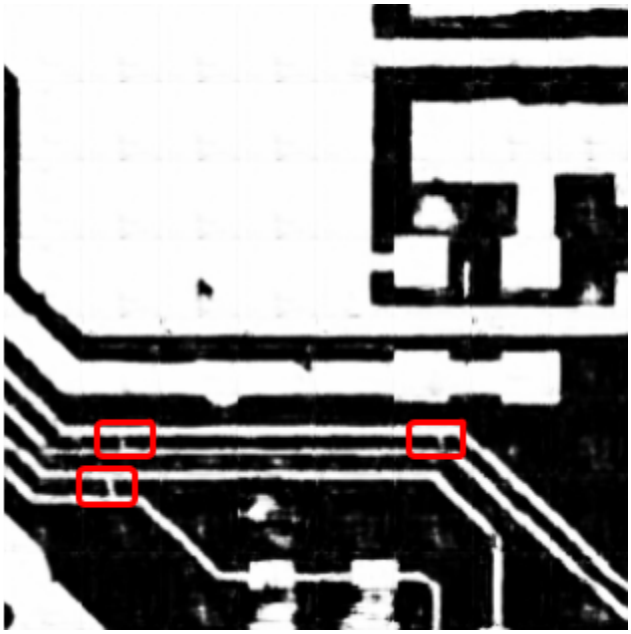
Autoencoder Reconstructed PCB (Aerospace)



Difference between Reconstruction and Original Image
Note that the red boxes denote the regions with the defects



Defective PCB (DeepPCB)



Autoencoder Reconstructed PCB (DeepPCB)

Difference between Reconstruction and Original Image
Note that the red boxes denote the regions with the defects

Above is the output when a whole defective PCB image is passed into our system. Note we are solely focusing on shorts and the red boxes are annotating all the short defects present in the PCB. Our system is able to successfully highlight a short (out of the three) in the Aerospace image however with many other false defects showing up that make it a bit difficult for an average user to decipher where the defect is. The system performs better with the DeepPCB image where it is able to highlight two out of the three shorts that exist in the image. It is important to note that DeepPCB has a lot less false positives than Aerospace, but has a fairly large false positive towards the bottom right of the PCB. We believe one of the reasons our system does not always detect shorts is because when the image is cut into regions a defect may be split across several of the regions. Thus, the defect becomes less obvious to the autoencoder in each region and has a higher probability of being ignored compared to the entire defect being present in one region. There are two solutions to this issue: when cutting the input image into smaller regions we can produce multiple sets of cuts for the same image with the same region size (e.g. in one cutting iteration we start with an offset of 0 pixels in each row and in another cutting iteration we start with an offset of 10 pixels in each row etc.) or with multiple region sizes and then pass each cutting set into the autoencoder and compare the reconstructed PCBs for each set of cuts to each other so that only similarities are kept in the final reconstruction; another solution is creating another machine learning algorithm that is able to segment components of a PCB board and then run the reconstruction on each of the segmented regions of the PCB.

The quantitative performance of our system on 3,060 80x80 DeepPCB images of shorts and open defects (these images the autoencoder was not trained on) can be summarized with the following: *Average Precision = 0.563* and *Average Recall = 0.517*. It is important to note that the average recall does not necessarily mean about half of the defects are defected. This is because the predicted segmentation mask of a defect may only occupy half the area of the actual defect segmentation mask. The area in which the system typically annotates as a defect is usually smaller than the total area of the image the defect occupies. This can be observed to a larger extent when examining the system's performance with 303 640x640 DeepPCB images of shorts where the *Average Precision = 0.163* and *Average Recall = 0.054*. The significantly smaller recall value is surmised to be a result of utilizing small regions (80x80) for template reconstruction. Visually speaking, when looking at a short the autoencoder typically eliminates the connection of the short between the solder joints it is connecting but the short itself still remains in the template reconstruction. Thus, the recall remains quite small as only a small area of the defect has been detected and successfully reconstructed. As corroborated by the performance data from the 80x80 and 640x640 image sets and noted above too, the amount of false positives increases as the complexity of the PCB regions being examined does. A scheme that employs larger regions for autoencoder reconstructions and one that does component segmentation may be useful for boosting the performance of the models in terms of recall and precision.

Project Planning and Management

1. Image pre-processing **(4/1/23)**
    a. Define area of PCB
    b. Binary Mask to standardize all images
2. Image Enhancement **(4/18/23)**
    a. Sharpen PCB images to emphasize possible errors
    b. Filter out noise that could hinder predictions
3. Encoder/Decoder Model **(5/5/23)**
4. Post Processing + Demo **(5/9/23)**
5. Analysis of Results **(5/12/23)**

These four steps are the main milestones we need to accomplish in order to be successful. We have allocated the most amount of time to image pre-processing as we believe that this will be the most complicated and time consuming task. Additionally, the thoroughness of our image pre-processing will have a direct correlation to the success of our final algorithm to detect PCB defects.

X. Conclusions

In conclusion, we implemented an autoencoder/decoder model to generate non-defective versions of inputs to identify defect features in the form of excess solder joints and shorts between components. To improve accuracy, we included a preprocessing module which highlights key features and thresholds all input images. Additionally we expanded our training dataset by incorporating images from *DeepPCB* which helped our model perform better when faced with new types of images. We then proceeded to test the performance of our non-defect reconstruction autoencoder on a variety of PCBs where we learned the limitations of our system. Specifically, the autoencoder often degrades the overall quality of an input image which can lead to excess false positive detection of defects as it introduces differences in regions that do not have defects. The probability of false positives in a region increases as the number of components to reconstruct increases. A future step to overcome the limitation of our system and improve its performance is introduce a preprocessing step where individual components in an input PCB is segmented; the hope is that maybe the autoencoder can then reconstruct each of the individual components instead of arbitrary regions of the an input PCB. This preprocessing step may improve performance as it is less likely for defects to become split among multiple regions making it more likely for an autoencoder to identify and reconstruct defects correctly. Additionally, this step would mitigate the false positive rate by preventing the autoencoder from reconstructing regions of a PCB with multiple components and instead focus on one component at a time.

Overall, we achieved good results with synthetic datasets such as DeepPCB and decent accuracy on the Aerospace corporation dataset. However, we did face key issues in this dataset as discussed earlier.

XI. References

Adibhatla, V.A.; Chih, H.-C.; Hsu, C.-C.; Cheng, J.; Abbod, M.F.; Shieh, J.-S. Defect Detection in Printed Circuit Boards Using You-Only-Look-Once Convolutional Neural Networks. Electronics 2020, 9, 1547. https://doi.org/10.3390/electronics9091547

Huang, W., Wei, P., Zhang, M. and Liu, H. (2020), HRIPCB: a challenging dataset for PCB defects detection and classification. The Journal of Engineering, 2020: 303-309. https://doi.org/10.1049/joe.2019.1183

Khalilian, Saeed, et al. "PCB Defect Detection Using Denoising Convolutional Autoencoders." *ArXiv.org*, 28 Aug. 2020, https://doi.org/10.48550/arXiv.2008.12589.

Kim, J., Ko, J., Choi, H., & Kim, H. (2021). Printed Circuit Board Defect Detection Using Deep Learning via A Skip-Connected Convolutional Autoencoder. Sensors (Basel, Switzerland), 21(15), 4968. https://doi.org/10.3390/s21154968

Krishnan, Muthu T. "Otsu's Method for Image Thresholding Explained and Implemented." *Muthukrishnan*, 13 Mar. 2020, muthu.co/otsus-method-for-image-thresholding-explained-and-implemented/.

Singh, Anurag and Namrata Dhanda. "DIP Using Image Encryption and XOR Operation Affine Transform." (2015). https://www.iosrjournals.org/iosr-jce/papers/Vol17-issue2/Version-5/B017250715.pdf

Tang, S, et al., “Online PCB Defect Detector on a New PCB Defect Dataset.” (2019). https://doi.org/10.48550/arXiv.1902.06197

XII. Appendices

- Otsu’s Algorithm
- Encoder/Decoder structure
- CNN structure
- Data collection utility
- Demo
- Data Sources
    - Deep PCB: https://github.com/tangsanli5201/DeepPCB
    - Peking University Dataset: https://www.kaggle.com/datasets/akhatova/pcb-defects
    - Aerospace Corporation Dataset: https://drive.google.com/drive/folders/1ez_sybKgIYs-KzZfYoRsF7CZZjDDekDT?usp=share_link