Using Pre-Trained Convolutional Neural Networks to Detect Insufficient Solder Application in PCB Manufacturing

Table of Contents:

Figures:

Introduction:

The intent initial intent of this project was to try and understand if a Convolutional Neural Network trained on AOI(Automated Optical Inspection) images of PCBs(Printed Circuit Boards) will have high enough accuracy in detecting defects to remove the Human-In-The-Loop from the verification process. This initial project was further modified due to time constraints to performing a small segment of defect classification and acting as a proof of concept for further investigation.

Currently, an AOI station photographs every component on a PCB and compares to a master sample image. The matching rates to the master image must be very high leading to significant false-rejects that need to be further verified by human operators. If a Convolutional Neural Network is able to classify images as either a true-reject or a false-reject with sufficient accuracy it will cut down significantly on the amount of images for human verification.

Dataset:

Contrary to initial assumptions the majority of the time of this project was consumed by the data pre-processing/selection stage. The raw data for this project consisted of .jpeg images of individual components on PCB, not an overview of a PCB with a wide field of view over the entire board. The effects of having the images in the format of individual components and not larger images of the board is that the images of defects per defect mode per component were relatively low compared to the overall dataset size. The labelled image repository was on the scale of terabytes, however for a specific defect mode(ie; insufficient solder) on a specific component (ie; U2 - Microprocessor 2) for a specific program (ie; Subaru XXXX) there were relatively few images. This had several effects on the model as will be shown below.

This dataset was gathered from an AOI system like previously mentioned, this meant that all images were captured with remarkably similar anchor points and positioning relative to the object of interest. This did however lead to one drawback that will be discussed in the architecture section.

One difficulty that was seen with the chosen dataset is that the labels were applied by human operators on inspection of past components. In several test-cases run, after examining the bottleneck to the model learning and generalizing properly it was suspected that several images were mis-labelled by the operator. This led to an interesting dilemma. I do not possess the same level of experience as the operators making the classification, so I felt uncomfortable in overruling their judgement of a defect. Additionally, to myself, pruning a dataset to generalize better seemed ethically dubious. So when this would occur I would move to try a different defect mode on a different component.

For this project I made use of Google Colab’s online ML tool. This made the implementation of the structure of data much easier. This tool only requires the images to be separated into classes within respective directories. The structure of the data for this project was as shown below in Figure 1.

The chosen defect I attempted to detect was insufficient solder on the leads of the U2 processor for a specific program number. I looked at 6 months of manufacturing data out of the 2 year production cycle to date. This dataset consisted of roughly 120,000 images that were deemed NG by the AOI software, from this only a paltry 396 were then verified as defects by operators. The motivation behind building a model to discern the difference automatically becomes abundantly clear.

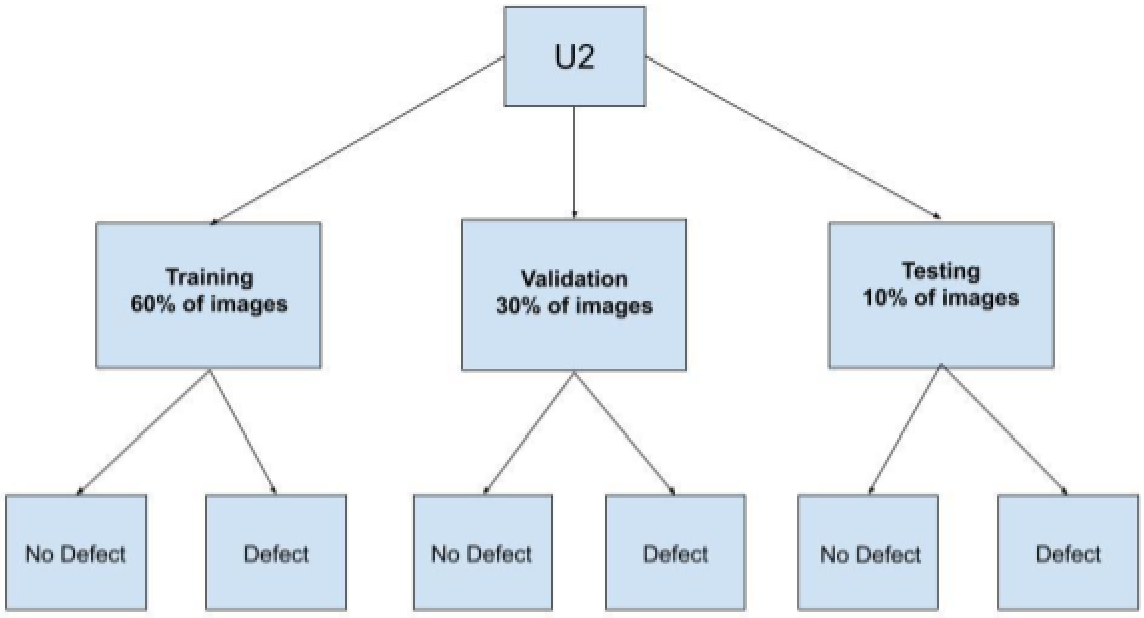

Figure 1: Layout of Data Directory

Data pre-processing:

Initial efforts were made to run the model with difference images, ie; finding the differences between an image and a master image and having the model learn what differences constitute a good part or a NG part. This proved to be ineffective and didn't allow for generalization as well as feeding the model the original image. Several different image pre-processing methods were tried as well before the images were passed to the model. The one that proved most effective however was a simple colour to gray transform. This seemed to allow for greater contrast between the solder joint and the background PCB given the lighting conditions as can be seen in Figures 2 and 3.

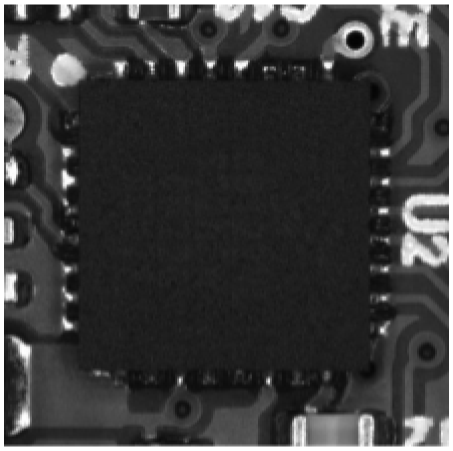
Figure 2: Sample Good Image

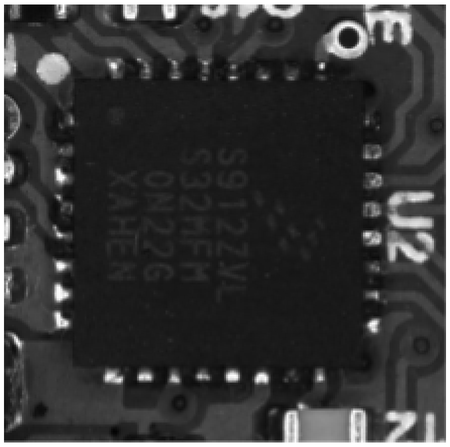
Figure 3: Sample NG Image

Architecture:

For this model a simple CNN architecture was chosen. Due to the limited number of images available within the NG category discussed earlier, a transfer learning regime was necessary. Several different pre-trained networks were tested and the best results came from the VGG16 Network with ImageNet weights.

As discussed earlier, the poignant factor to the method of image capturing is that all the images are captured in a very similar manner. For this reason, all test images and validation images that would be given to the model would also look similar. It was found that typical methods of dealing with limited samples such as data augmentation through image translation and rotation and other such transformations move the image away from what it will always be given and do nothing to improve the accuracy. One technique that was used to mitigate the effects of highly unbalanced classes is (significant) undersampling of the majority class. Of the roughly 120,000 possible confirmed good images I sampled a set of 700 images to contrast the 400 confirmed NG samples. This yielded the highest accuracy in terms of dataset split.

For this particular model, the data doesn't have much of a similarity to the ImageNet images so the weights for the VGG16 network were intended to be used as a feature extractor. As such the last few layers of the network were left as trainable on the dataset as outlined in Figure 4 below.

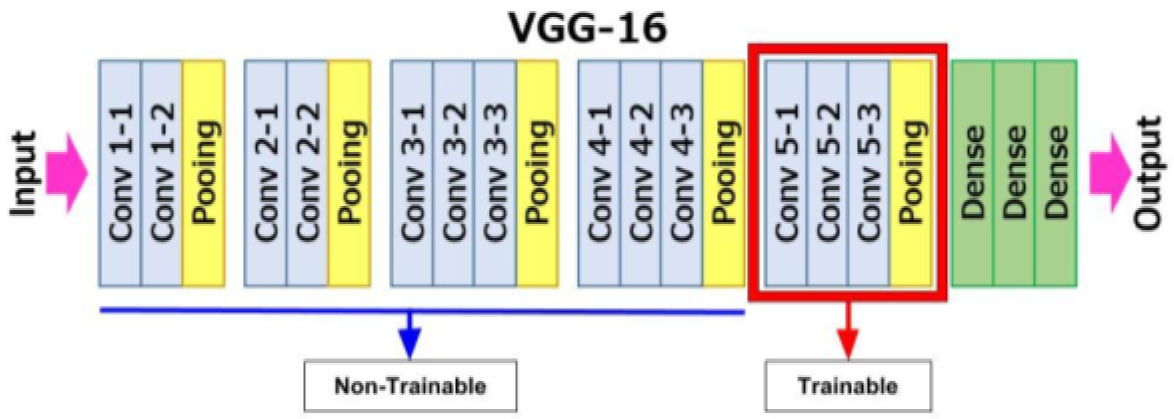

Figure 4: VGG16 Network Architecture

The typical features of a CNN were included after the input of the pre-trained ImageNet weights, being sure to include a drop-out layer with weighting of 0.5 to try and reduce overfitting, especially due to the small dataset, to try and mitigate simply memorizing the data. ELU activation functions were used to try and maximize the usefulness of the neuron within the negative featurespace. As this is only a binary classification problem a final sigmoid activation functions was used. If this model were to be extended beyond a binary classification problem a "softmax" or other activation function would be selected.

Recall was added as a metric of the accuracy of the model because this is truly the measure that matters in this particular use case. We are trying to understand the amount of defects it can correctly classify as defects. There is very little harm in classifying a good part as a defect so the precision is not warranted in this case.

Results:

The model discussed above achieved an accuracy of ~98% fairly quickly (10 epochs) with a recall of 98% within the same time frame. To ensure the model was not simply memorizing the data overtime the test set which was never seen by the model in training or validation was predicted and achieved an accuracy and recall of 96% in classifying any defects present. This is still a relatively small dataset and more samples should be tested in order to confirm the accuracy of the model. After about 15 epochs however the model did memorize the training data as can be witnessed by the 100% accuracy and recall.

The model also seems to generalize remarkably quick, showing >80% accuracy and recall after the first epoch as outlined in Figure 5 below. I believe this lends itself to the fact that the images are all very similar save a few small differences. Using the pre-trained network as feature extractors work well to discern where the differences are, and because there are not many differences are able to correctly classify what leads to an image being labelled as NG.

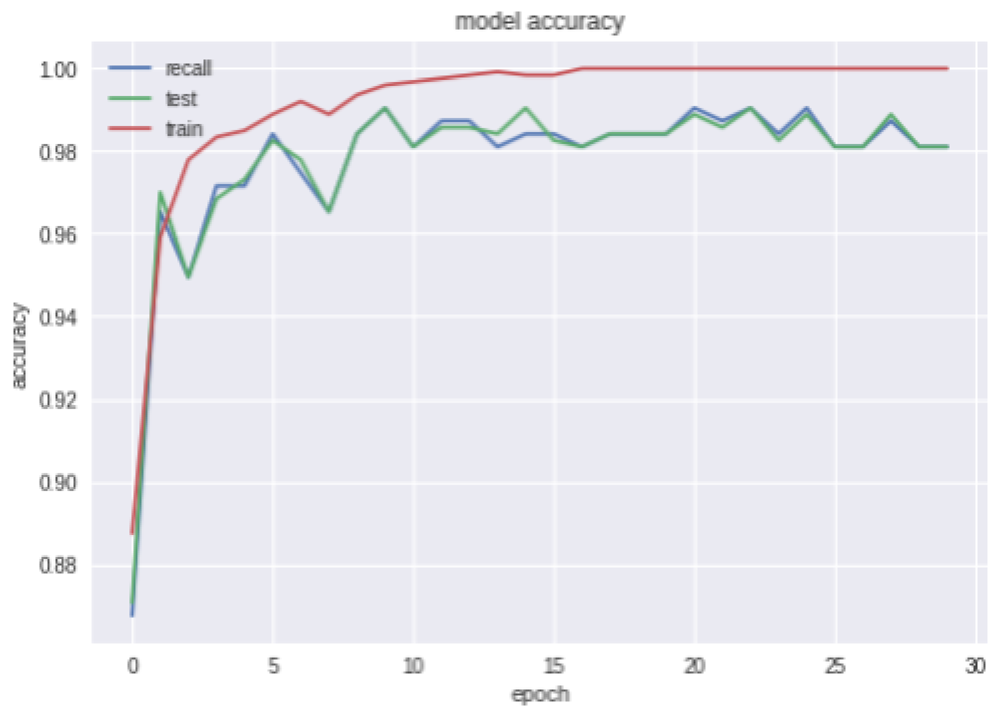

Figure 5: Accuracy Results

Next Steps:

The architecture that was selected would not be the architecture I would continue this project with or redo it with, given the chance. The CNN architecture chosen was able to learn and with some accuracy classify defects as such, but for very specific, non-transferable cases. Additionally, the data, I believe would lend itself very well to an object detection and classification model over a simple binary classification model. The data is currently formatted such that the defect locations are mapped. This would also significantly increase the instances of data available for defect modes like insufficient solder where there might be 26+ instances of a component lead in a single image to learn correct solder quality.

I would endeavour to be able to apply an object detection model to these images, trained on the mapped defect location, to be able to learn to detect things like insufficient solder on any lead not simply for one component on one program.