Available online at www.HighTechJournal.org

ISSN: 2723-9535

HighTech and Innovation Journal

Vol. 3, No. 1, March, 2022

The Use of a Convolutional Neural Network in Detecting Soldering Faults from a Printed Circuit Board Assembly

Muhammad Bilal Akhtar [1*]

[1] *Department of Signal Processing, KTH Royal Institute of Technology, Stockholm, Sweden.*

Received 21 September 2021; Revised 15 December 2021; Accepted 24 December 2021; Published 01 March 2022

Abstract

Automatic Optical Inspection (AOI) is any method of detecting defects during a Printed Circuit Board (PCB) manufacturing process. Early AOI methods were based on classic image processing algorithms using a reference PCB. The traditional methods require very complex and inflexible preprocessing stages. With recent advances in the field of deep learning, especially Convolutional Neural Networks (CNN), automating various computer vision tasks has been established. Limited research has been carried out in the past on using CNN for AOI. The present systems are inflexible and require a lot of preprocessing steps or a complex illumination system to improve the accuracy. This paper studies the effectiveness of using CNN to detect soldering bridge faults in a PCB assembly. The paper presents a method for designing an optimized CNN architecture to detect soldering faults in a PCBA. The proposed CNN architecture is compared with the state-of-the-art object detection architecture, namely YOLO, with respect to detection accuracy, processing time, and memory requirement. The results of our experiments show that the proposed CNN architecture has a 3.0% better average precision, has 50% less number of parameters and infers in half the time as YOLO. The experimental results prove the effectiveness of using CNN in AOI by using images of a PCB assembly without any reference image, any complex preprocessing stage, or a complex illumination system.

Keywords: Automatic Optical Inspection; Deep Learning; Neural Network; Object Detection; Printed Circuit Board Assembly; YOLO.

1. Introduction

A Printed Circuit Board (PCB) is a mechanical structure that holds and connects electronic components. A PCB without electronic components installed is also called a bare PCB. Soldering is used to fix the electronic components in place on the PCB permanently by applying hot copper liquid onto a joint. After placing the electronic components onto the bare PCB it becomes a printed circuit board assembly (PCBA). With the development of technology, demand for electronic products to contain more features and be smaller in size has emerged. This demand has in turn caused the PCBA area to be smaller, more complex and denser. From enhanced complexity stems the need for accuracy. PCBA problems are often very costly to correct [1]. That is why, in a PCBA mass production process, the inspection of PCBA is considered an important task. For years, Manual Visual Inspection (MVI) has acted as the de facto test process for PCBA. This, coupled with an electrical test, such as an in-circuit or functional test, was deemed enough to detect major placement and soldering errors [2]. Manual modes of inspection had a low reliability rate and were often affected by visual fatigue [3, 4].

PCBA production process consists of three main steps. 1) Solder paste layering on the board's surface; 2) component positioning; and 3) solder joint shaping by reflowing the solder paste. At each step of the production process, different

* Corresponding author: bilal_akhtar88@yahoo.com

http://dx.doi.org/10.28991/HIJ-2022-03-01-01

defects could occur that could be detected by stage specific AOI. Inspection after the first stage is called "Solder Paste Inspection (SPI)". The inspection techniques applied after the second stage are known as automatic placement inspection (API) techniques, while the inspection carried out after the third stage is known as post soldering inspection (PSI). It is observed that in all the PCBA processes, 90% of the faults are only detectable during PSI [5]. Possible faults occurring at this stage are solder bridge (a form of short in which solder creates a short circuit between two pins not meant to be connected), cold solder (a form of open where solder has not melted to create an electrical connection between the pin and the board), and dry-joint (where solder has not been applied to a pin, and bare copper is visible), to name some.

To detect structural defects at an early stage in the PCBA manufacturing process is necessary to reduce the PCBA production cost. Many complex and high-cost techniques have been proposed in the industry, such as using X-ray, optical, ultrasonic, and thermal imaging [6]. Using classical image processing algorithms, Automated Optical Inspection (AOI), also known as Automated Visual Inspection (AVI) was proposed as a technique that improved diagnostic capabilities in terms of speed and tasks. Moganti et al. (1996) proposed a categorization of AOI algorithms based on the way information is treated, i.e., a referential approach and a non-referential approach [5]. The referential method compares the image to be inspected with a defect-free template, requires high alignment accuracy, and is sensitive to illumination. The non-referential approach works by checking if the image to be detected satisfies the general design rules, paving the way to losing irregular defects that do not satisfy the design rules. These image processing and classification algorithms take a lot of computational configuration and are usually defect specific. They can't be found useful across multiple PCBAs.

Because of its ability to self-learn and its promising potential for generalizability on object classification and detection tasks, CNN has been successful in replacing traditional computer vision algorithms. The deep network architecture of CNN [7] can detect discrimination features from all the input images on its own, so we do not need individuals to define image features. With improved computing machines, especially GPUs [8], the detection process has become so fast that on-line PCBA fault detection is possible using CNN. This paper outlines a method to design an optimal CNN architecture for soldering fault detection in a PCBA. It presents a novel CNN architecture that performs well in detecting soldering bridge faults on PCBAs from a single image without requiring any pre-processing step or a referential PCBA image. The dataset contains images of different PCBAs with soldering bridge faults. The dataset is small and imbalanced, so various data augmentation techniques were used.

The rest of the paper is structured as follows; Section 2 outlines the limitations of the previous research done in using CNN for AOI. Section 3 describes the methodology used to design the optimized CNN architecture. Section 4 presents the results of the optimized CNN architecture with the YOLO architecture. Section 5 concludes the paper and outlines the future work.

2. Literature Review

In the early days of the PCBA manufacturing industry, inspection tasks were performed by humans who were fatigued from perfunctory tasks. A comprehensive summary of the advancements in AOI systems over time has been given in Huang and Pan, (2015) [1], Moganti et al. (1996) [5], Taha et al. (2014) [9], Harlow (1982) [10], and Chin (1988) [11]. According to a report stated in Loh and Lu (1999) [12], solder joint defects correspond to 55% of the total faults in a PCBA. AOI can be broadly classified into three main categories, namely referential, non-referential, and hybrid methods [5]. Referential AOI systems compare the image of the PCB under test to a template image of the PCB that is free of any defects [13-15]. Referential methods include image subtraction, introduced by Lee (1978) [16], feature matching or template matching as used by Hara et al. (1983) [17], and comparing the compression codes [18]. Referential methods are susceptible to degraded performance due to image misalignment and variations in environmental conditions when capturing images. Non-referential methods remove the misalignment issues from the inspection process and are based on general design rule verification [19, 20]. Non-referential methods require complete knowledge of the PCBA design. Hybrid methods combine the positive effects of both referential and non-referential methods. Various forms of AI have been widely used in hybrid approaches, with different referential methods used as a preprocessing step for localizing the fault area in the image [21]. To control the variations in illumination conditions while capturing the image of the PCB, most AOI systems provide complex user-controlled illuminations [22, 23], e.g., three ring-shaped LEDs as shown in Figure 1.

The biggest potential barrier to AOI is its inflexibility and reliance on system configuration. CNN is a self-learning process that has potential for generalizability. However, most of the work done on AOI using CNN has been very preliminary and in its initial phase. Previous applications of CNN have been mainly focused on bare PCBs using a reference image, a computation-intensive preprocessing step, a complex illumination system, or a combination of these [24-28].

Acciani et al. (2006) [29] proposed a general architecture for applying very shallow neural networks in AOI based on hand-crafted features. Fanni et al. (2000) [30] used the energy components from the Fast Fourier Transform (FFT) and Haar Transform (HT) as the input feature set. Classic machine learning algorithms with input from selected feature

sets [31-33]. CNN [34, 35] has achieved outstanding results in image classification and detection tasks [36, 37]. A CNN takes in the whole image and learns the features necessary for classification and detection, whereas previous classifiers take a set of manually selected features. A huge amount of data is needed to train a CNN that generalizes well. For developing CNN based AOI, researchers feel a great void in the availability of publicly accessible, huge and diverse datasets. Tang et al. (2019) [26] and Huang and Wei (2019) [25], the authors present a publicly available dataset that contains examples of defects on a bare PCB only. These datasets cannot be used to design a CNN for post-soldering AOI systems.

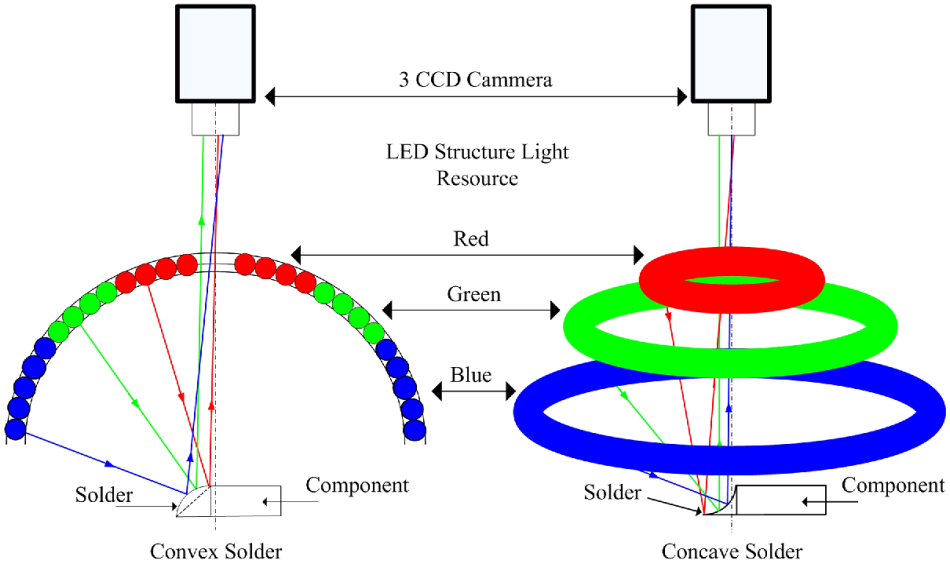

Figure 1. Three Ring LEDs structure taken from Wu and Zhang (2014) [27]

Tang et al. (2019) [26] proposes a template image based object detection CNN that treats the faults in a PCB as objects. Currently various CNN algorithms exist for object detection that try to balance the accuracy of the CNN with architecture efficiency. These object detection CNNs are broadly divided into two stage detectors or single stage detectors.

R-CNN (Regions with CNN features) [38] is a famous two stage detector that uses selective search [39], in stage one, to generate region proposals, also called regions of interest (ROI). In a later version, Fast R-CNN [40], the whole image passes through a CNN once instead of applying CNN on each ROI individually. In Faster R-CNN [41] the region proposal algorithm is integrated into the CNN. Even after all the advances R-CNN is very slow but performs very well in terms of prediction accuracy making R-CNN impossible to infer in real-time. Overfeat combines the classification and localization tasks into single object detection CNN [42] that is faster but less accurate than R-CNN. YOLO (You Only Look Once) [43] is simple single stage object detection CNN that divides the image into a grid of fixed size and for each grid cell it detects the bounding box coordinates through regression and the class probabilities for a fixed number of anchor boxes. YOLO was able to generalize well, corroborated by its ability to predict objects from hand painted images. It is the fastest object detection algorithm even though it drags down the performance in accuracy. In a better version YOLOv2 [44], the authors have used batch normalization for faster convergence during training; a custom feature extraction network that makes it faster; a convolutional anchor box predictions instead of fully connected layer that has shown to increase the prediction accuracy at the expense of increased false detection i.e. detecting an object in a grid cell that was not present in reality, and a pass-through layer to use fine-grained features from an earlier layer leading to increased accuracy performance than earlier version of YOLO. YOLOv3 [45] was further improved by incorporating feature pyramid representation for multiscale detection and increase in the number of feature extraction layers with residual connections that improved its accuracy significantly. Figure 2 represents the working principal of YOLO.

Lin et al. (2018) demonstrates for the first-time application of famous object detection CNN, YOLOv2, for detecting capacitors on a PCBA image [46]. Adibhatla et al. (2020) designed a deep learning algorithm based on the YOLO approach for detecting defects on a bare PCB [47]. Khare et al. (2020) [48] has used YOLOv3 to detect missing components from a PCBA using dataset from [49] that labels each IC component on PCBA image.

To the best of the author's knowledge at the time of writing this paper this is the first work on the effectiveness of using CNN for AOI of a PCBA from 2 dimensional colored image of the PCBA without requiring a referential image, or any pre-processing step, or a complex illumination system. In our experiment we base the CNN design on the grid cell division principal used in YOLO. Each input PCBA image is divided into 14×14 grid. The output is a binary value for each grid cell. An output value of 1 suggests presence of soldering bridge fault in that grid cell.

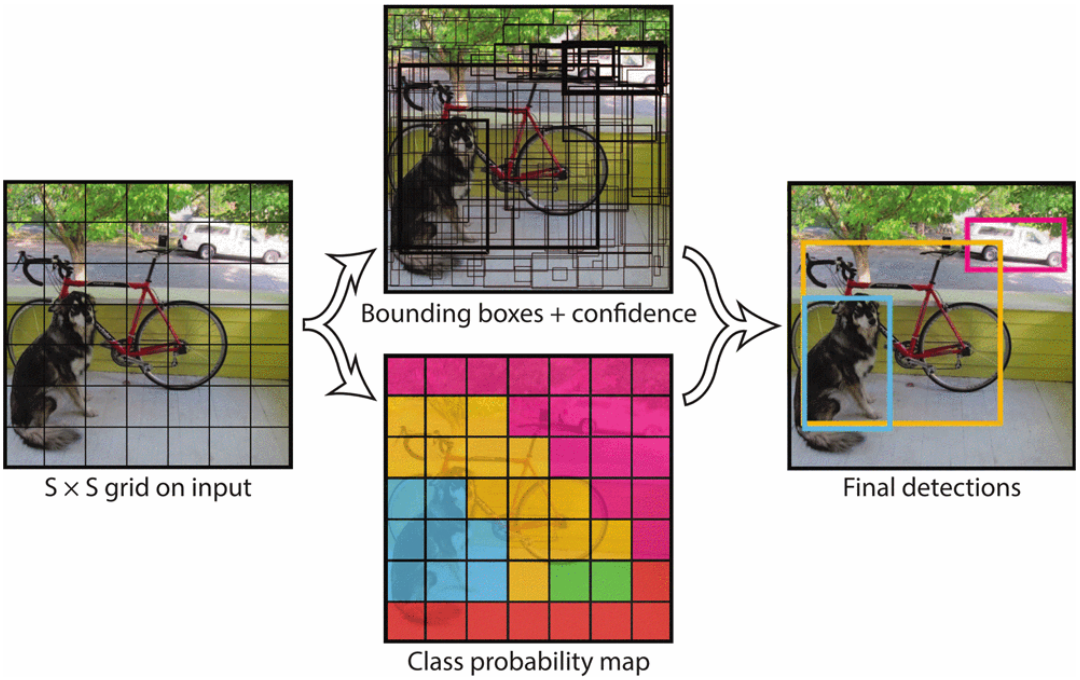

Figure 2. Unified Detection using YOLO taken from Redmon et al. (2016) [43]

3. Research Methodology

There is no one-hit formula to design an optimum CNN model, therefore, we had to rely on experiments to find an optimum CNN model to detect soldering bridge faults.

3.1. Dataset

Due to unavailability of open source dataset of soldering faults on a PCBA the dataset was collected manually. It includes 2D RGB images of 64 different PCBAs with soldering bridge fault manually introduced at different places. The total number of soldering bridge faults in the dataset is 359. The images in the dataset are resized to the size of 1024 ×1024. A corresponding annotation file was generated that contains bounding box information for all the possible defects in the image. Figure 3 shows a hypothetical image of PCBA and its corresponding annotation file. The dimensions of RGB input image to the CNN are chosen to be 448 × 448, inspired by YOLO. Resizing the image of whole PCBA to this size would incur loss of crucial information as the size of soldering faults are very small compared to the whole image size. The available dataset was not enough to train a CNN that generalizes well. Data augmentation is used to create a larger dataset for training. To avoid information loss and keeping in view the concept of generalization, we randomly cropped images of the complete PCBA panels in the dataset with the dimensions ranging from 448 × 448 to 512×512. The augmented dataset contains mutually exclusive 2000 images created by cropping images of the PCBA panels and randomly applying rotation and flipping on each cropped image. A grid of size 14 × 14 was used.

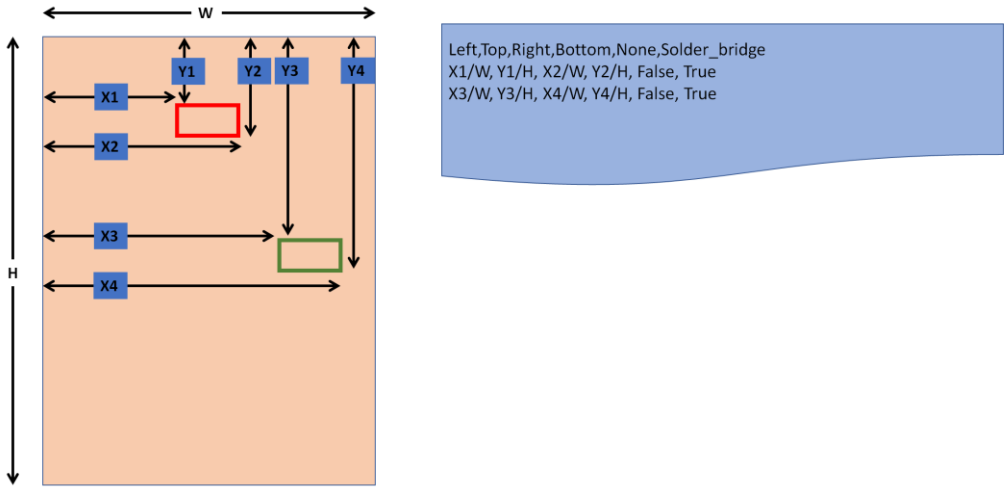

Figure 3. Method for writing annotation file (on the right side) for a PCBA image file (on the left side) with dimensions W x H and 2 soldering bridge faults represented in red and green bounding boxes

3.2. CNN Design

The performance of a CNN gets better when the network gets deeper [50] at the expense of increased resources utilization and increase in the number of learnable parameters. The choice of hyperparameters also plays a significant role in improving the performance of a CNN. We designed the CNN for soldering bridge detection in PCBA from scratch using the design optimization principles of inception module, bottleneck layer and residual block. In the inception module [51] filters of different sizes are used in each layer and the results are stacked. This allows the model to choose optimal filter size for itself. Conventionally the number of channels increases as we go deeper in a CNN model. This gives the deeper layers a larger receptive field. Lin et al. (2013) [52] a network in network layer is introduced as a 1 × 1 convolutional layer called a bottleneck layer. The bottleneck layer has the same summarising effect as pooling layer except that pooling layer shrinks the width and height while the bottleneck layer shrinks the number of channels which in turn reduces the overall number of parameters. A ground-breaking CNN architecture optimization principal was the introduction of residual block in He et al. (2016) [53]. Residual block effectively diminishes the vanishing gradient problem with increasing depth of the network without adding to the computational complexity of the architecture. Figure 4 shows the basic structure used for designing the CNN. The number of hidden layers is chosen from [25] that describes CNN architecture for detecting faults in bare PCB. The five max pooling layers downsample the input image to an output of size 14×14. The output is a binary number for each grid cell. It is 1 if there is soldering bridge fault in that grid cell and 0 otherwise. The YOLO architecture described in [45] is used as benchmark for comparing the performance of the optimally designed CNN architecture. The metrics used for comparison are: detection accuracy, inference time, and number of learnable parameters in CNN. To determine the detection accuracy of an object detection algorithm, Average Precision (AP) is a popular metric that ranges between 0 and 1. AP is determined for an individual class. Mean average precision (mAP) is the mean value of average precision for all the classes in a dataset. As we have only soldering bridge fault in our dataset we use AP to determine the detection accuracy of the models. A higher value of AP signifies higher detection accuracy of a model.

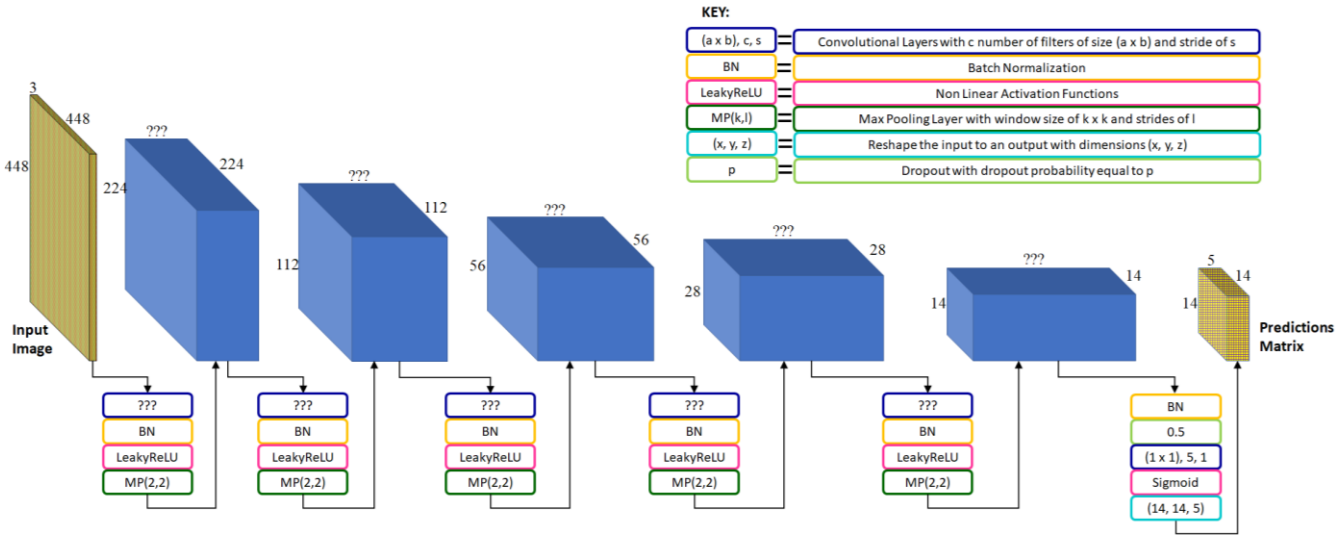

Figure 4. Basic structure used for designing CNN to detect soldering faults

Conventional convolutional layers are referred to as plain convolutional layers that do not contain an inception module or a residual block or a bottleneck layer. We start the experiment with basic CNN architecture employing plain convolutional layers and then try various combinations of convolutional layers added on to the basic structure i.e. going deeper, bottleneck layers, inception modules and residual blocks. The models were trained using the augmented dataset. The models were designed based on the following methodology:

Model 1 – Plain model based on description in Figure 4.

Model 2 – Adding CONV layers in low level and mid-level features extraction layers of Model 1.

Model 3 – Adding CONV layers in high level features extraction layers of Model 1.

Model 4 – Adding residual block and bottleneck layer to the model giving the best performance results from Model 1-3.

Model 5 – Adding Inception module in low level feature extractor layers.

Model 6 – Adding CONV layers to the most efficient model chosen from Model 1-5.

3.3. Hyperparameters

We use filters of size 3×3 and stride value of 1 in all the convolutional layers except for the Inception layer which is a combination of filters of different sizes. We used max pooling layer with window size of 2 and stride value of 2. Following YOLO, we used Leaky ReLU, with slope for negative input values equal to 0.1, as the activation function in all the hidden layers and used sigmoid activation function in the last layer to return the prediction values in between 0 and 1. Adam optimizer is used for training. As our output values are binary hence, we used binary cross entropy loss function. A loss function determines how far away the predicted output of the model is from the ground truth during the training process.

The learning rate was initially chosen as a small value of 1×10^{-5} for the first 25 epochs to induce stability in the training process. For the next 50 epochs its value was raised to 1×10^{-3} for a faster convergence of the model. For the remaining epochs we used a learning rate of 5×10^{-6}. The model was evaluated tested after every 25 epochs using average precision (AP) of soldering bridge faults as the metric, with the threshold value of 0.5. Training was stopped when the AP started to drop. Beyond this point the model start to overfit the training data, a phenomenon where the model learns detail and noise in the training data such that it negatively starts to impact the performance of the model on new and previously unseen data, in other words it starts to lose generalization. Regularization is any supplementary technique that makes the model generalize well and prevents the model from overfitting. A simple technique to choose a model that generalizes well is to terminate training when the loss on validation dataset starts to decrease. This technique is called early stopping. Batch normalization is proven to improve convergence and generalization in training neural networks in Luo et al. (2018) [54]. We added batch normalization followed by leaky ReLU activation function for regularization. Another very common, simple and extremely effective regularization technique used is dropout [55]. When using dropout on a layer in CNN each neuron is ignored during a training step with a probability p, where p is a hyperparameter called dropout rate. We have used dropout layer before the prediction layer with dropout rate equal to 0.5.

4. Results

This section compares the performance of the 6 CNN architectures described in Appendix I with YOLO architecture based on the three important metrics, namely the accuracy of the model measured through AP for soldering bridge fault, inference time (all the time measurements are taken on the same machine) and the number of learnable parameters in the model. Model4 and Model5 are based on Model1, while Model6 alters the architecture of Model4. In the comparison tables ↑ means an increase in performance compared to YOLO and ↓ means a decrease in performance as compared to YOLO. Table 1 describes the number of learnable parameters for the models used in the experiment. Table 2 shows the results of soldering bridge fault detection AP for the models used (the higher the better.) Recall value gives the total number of soldering bridge faults detected in the test images out of the total number of true soldering bridge faults in the test images. Table 3 shows the results of time taken by the models for inferencing a single panel image. The inferencing time experiment was repeated 5 times for each model and Table 3 shows the average value.

Table 1. Comparison of results for the number of learnable parameters

Model	Number of Parameters	Comparison
YOLO	56,630,623	=
Model1	**6,300,390**	↑
Model2	7,624,262	↑
Model3	71,607,014	↓
Model4	9,102,054	↑
Model5	13,401,702	↑
Model6	14,968,422	↑

Table 2. Comparison of results for AP of soldering bridge fault detection

Model	AP of soldering bridge fault detection	Recall	Comparison
YOLO	0.80489	142 / 167 ≈ 85%	=
Model1	0.78792	135 / 167 ≈ 81%	↓
Model2	0.65828 (Experiment 1) 0.73503 (Experiment 2)	125 / 167 ≈ 75% 127 / 167 ≈ 76%	↓ ↓
Model3	0.82036	140 / 167 ≈ 84%	↑
Model4	0.83383	141 / 167 ≈ 84%	↑
Model5	**0.83733**	**143 / 167 ≈ 86%**	↑
Model6	0.82435	139 / 167 ≈ 83%	↑

Table 3. Inference times for the different models

Model	Inference time (in seconds)	Comparison
YOLO	14.54	=
Model1	**4.40**	↑
Model2	10.78	↑
Model3	9.32	↑
Model4	4.60	↑
Model5	21.01	↓
Model6	23.94	↓

From the results, it can be inferred that Model1 performs equally well in detecting soldering bridge faults as the YOLO model, with a slight decrease (≈2%) in the AP of soldering bridge fault detection but a significant savings in memory (≈88%) and inference time. These results corroborate the claim that an optimal CNN architecture exists that can perform better than the state-of-the-art YOLO architecture in detecting soldering faults.

Results for Model2 and Model3 signify the importance of adding CONV layers in improving the accuracy performance of a model. As a rule of thumb, increasing the number of CONV layers increases the number of features learned, which in turn improves the accuracy of the CNN architecture, but only up to a certain number of layers [52]. Model3, which adds CONV layers to the high-level feature extraction part of Model1, shows ≈4% increase in AP at the cost of a significant increase in memory requirements and a higher inference time when compared to Model1. Model2, which adds CONV layers to the low-level feature extraction parts of Model1, showed anomalous behavior with a significant decrease in accuracy performance (≈16%), an increase in the inference time (equivalent to the increased inference time of Model3), and a slight increase in memory requirement. For verification, we repeated the training of Model2, resulting in a decreased accuracy once again. These results also suggest that adding CONV layers to the high-level feature extraction part of a CNN has a lower chance of overfitting. Comparing Model3 to the YOLO architecture suggests that YOLO also has better accuracy than Model1 because it uses more CONV layers in the mid-level and high-level feature extraction parts. Another implication from these results is that inference time does not only depend upon the number of learnable parameters in a model, as the inference time for Model2 and Model3 is almost similar, whereas, the number of learnable parameters in Model2 is 10 times higher than in Model3. This can be dependent on many parameters, including the depth, filter size, value of the stride, type of operations, and many more.

To understand the phenomenon of overfitting due to an increase in the number of CONV layers in a model, we can use the example of a model that classifies an image as a cow or not a cow. After a certain number of layers, adding more layers to the model will let it learn non-important features, leading to poor generalization of the model, e.g., learning to extract a bell from images of cows with bells around their necks in the training dataset, or a green background if images labeled as cows are captured in meadows.

For further experiments, we therefore chose Model1, which gives the best compromise between detection accuracy, memory requirement, and inference time. Model4 onwards is based upon improvements in the architecture of Model1. Model4 that incorporates only skip connections to Model1 displays a significant improvement in the performance of Model1 as the recall value has improved from 135 to 141 and the AP value also shows an improvement of 6% and 3.5% compared to Model1 and YOLO, respectively. The inventors of the residual block attribute the improvement in accuracy to the ability of the model to learn identity mappings that bypass the nonlinearities of a CONV layer. The performance improvement in Model4 suggests the effectiveness of the residual block not only in increasing the accuracy without increasing the learnable parameters, as Model4 has almost 84% less learnable parameters than YOLO. Model4 and Model1 have almost the same inference time.

Model5 indicates the impact of applying inception modules in the low-level and mid-level feature extraction layers, and it proves to be beneficial in improving the accuracy of Model4 slightly at the cost of increasing the number of learnable parameters and inference time significantly. In Model4 and Model5 we also used the bottleneck layer. Results of Model5 and Model6 show a slight improvement in the accuracy performance at the expense of a significant increase in the number of learnable parameters and significantly slower inference time. The average accuracy of manually detecting the soldering faults with the aid of a magnifying glass is almost 90% [5] and the fault detection accuracy given by the recall value for the optimal CNN is 84%. This suggests that CNN can be powerful in achieving human level accuracy given it is trained on a larger dataset with an equal number of diverse examples.

Figure 5 shows a sample of the prediction result with grid lines drawn for understanding that the image is divided into 14×14 grid. The prediction returns a binary for each grid cell. In the future, with more data, we can work on drawing bounding boxes around the fault only.

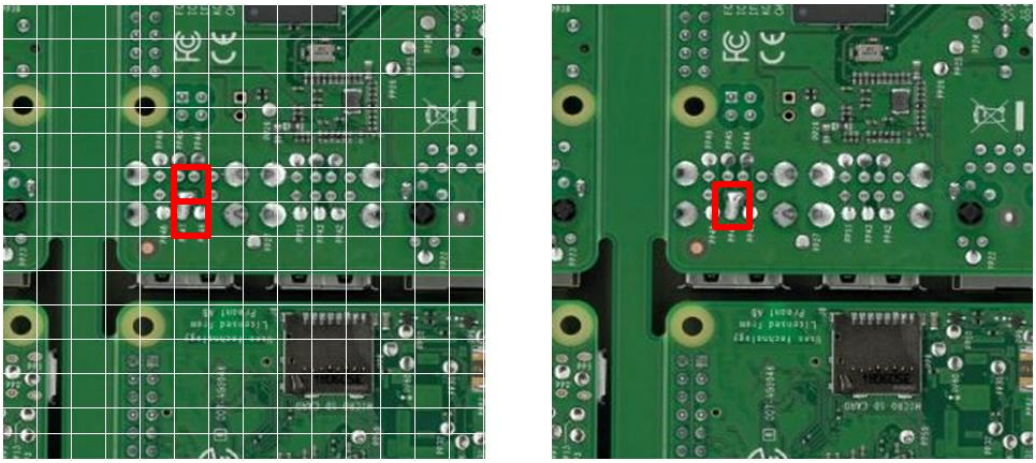

Figure 5. Image on the left: Prediction by Model4, Image on the right: Ground truth

5. Conclusion

It has been shown that CNN based AOI can be used to replace manual inspection to detect soldering faults on PCBAs. The problem was treated as an object detection task. Fast inferencing plays a vital role in AOI, and for this reason, we based our custom CNN on YOLO, which is a state-of-the-art fast object detection CNN. The experiments show that using state-of-the-art object detection CNNs in AOI can perform well in accuracy detection but does not prove to be resource efficient. Hence, transfer learning does not always provide an efficient solution for carrying out a CNN based AOI task. It was also shown that the accuracy performance of a custom CNN can be improved using optimization blocks without compromising its resource efficiency. The use of a bottleneck layer was effective in constraining the memory utilization while achieving high accuracy. Use of residual blocks had the most significant impact on accuracy improvement without any increase in resource utilization. It was seen that YOLO provides a simple technique for designing fast object detection CNN that generalizes well. This technique of dividing the image into a grid can be the basis of a custom CNN design for other types of fault detection in a PCBA.

The author believes that the performance and generalizability of the CNN model can be improved by collecting more data with a diversity of examples and classes. This research paper sets a solid foundation that CNNs can provide a simple, highly flexible, and fast AOI system for fault detection in PCBAs. In future work, the optimized architectural design principles described in this study can be used to detect multiple types of faults in PCBAs. It requires there to be an open source dataset for different types of faults in PCBAs.

6. Declarations

6.1. Data Availability Statement

The data presented in this study are available in article.

6.2. Funding and Acknowledgements

The author would like to thank Sony Mobile Communications AB, Lund, Sweden for their contribution to this study.

6.3. Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

7. References

[1] Huang, S. H., & Pan, Y. C. (2015). Automated visual inspection in the semiconductor industry: A survey. Computers in Industry, 66, 1–10. doi:10.1016/j.compind.2014.10.006.

[2] Richter, J., Streitferdt, D., & Rozova, E. (2017). On the development of intelligent optical inspections. 2017 IEEE 7th Annual Computing and Communication Workshop and Conference, CCWC 2017, 1-6. doi:10.1109/CCWC.2017.7868455.

[3] Day, R. H. (1972). Visual spatial illusions: A general explanation. Science, 175(4028), 1335–1340. doi:10.1126/science.175.4028.1335.

[4] Coren, S., & Girgus, J. S. (1973). Visual spatial illusions: Many explanations. Science, 179(4072), 503–504. doi:10.1126/science.179.4072.503.

[5] Moganti, M., Ercal, F., Dagli, C. H., & Tsunekawa, S. (1996). Automatic PCB inspection algorithms: A survey. Computer Vision and Image Understanding, 63(2), 287–313. doi:10.1006/cviu.1996.0020.

[6] Janczki, M., Becker, kos, Jakab, L., Grf, R., & Takcs, T. (2013). Automatic Optical Inspection of Soldering. Materials Science - Advanced Topics. doi:10.5772/51699.

[7] Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics, 36(4), 193–202. doi:10.1007/BF00344251.

[8] Raina, R., Madhavan, A., & Ng, A. Y. (2009). Large-scale deep unsupervised learning using graphics processors. Proceedings of the 26th International Conference on Machine Learning, ICML 2009, 873–880.

[9] Taha, E. M., Emary, E., & Moustafa, K. (2014). Automatic Optical Inspection for PCB Manufacturing: a Survey. International Journal of Scientific & Engineering Research, 5(7), 1095–1102.

[10] Harlow, C. A. (1982). Automated Visual Inspection: A Survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-4(6), 557–573. doi:10.1109/TPAMI.1982.4767309.

[11] Chin, R. T. (1988). Automated visual inspection: 1981 to 1987. Computer Vision, Graphics and Image Processing, 41(3), 346–381. doi:10.1016/0734-189X(88)90108-9.

[12] Loh, H. H., & Lu, M. S. (1999). Printed circuit board inspection using image analysis. IEEE Transactions on Industry Applications, 35(2), 426–432. doi:10.1109/28.753638.

[13] Pal Singh Chauhan, A., & Bhardwaj, S. C. (2011). Detection of bare PCB defects by image subtraction method using Machine Vision. Proceedings of the World Congress on Engineering 2011, WCE 2011, 2, 1597–1601.

[14] Ozturk, S. (2017). Detection of PCB Soldering Defects using Template Based Image Processing Method. International Journal of Intelligent Systems and Applications in Engineering, 4(5), 269–273. doi:10.18201/ijisae.2017534388.

[15] Lin, S. C., Chou, C. H., & Su, C. H. (2007). A development of visual inspection system for surface mounted devices on printed circuit board. IECON Proceedings (Industrial Electronics Conference), 2440–2445. doi:10.1109/IECON.2007.4459975.

[16] Lee, D. T. (1978). A computerized cutomatic inspection system for complex printed thick film patterns. Technical Symposium East, 3, 172–177.

[17] Hara, Y., Akiyama, N., & Karasaki, K. (1983). Automatic Inspection System for Printed Circuit Boards. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-5(6), 623–630. doi:10.1109/TPAMI.1983.4767453.

[18] Hong, J. joong, Park, K. ja, & Kim, K. gu. (1998). Parallel processing machine vision system for bare PCB inspection. IECON Proceedings (Industrial Electronics Conference), 3, 1346–1350. doi:10.1109/iecon.1998.722846.

[19] Sun, Y. N., & Tsai, C. T. (1992). A new model-based approach for industrial visual inspection. Pattern Recognition, 25(11), 1327–1336. doi:10.1016/0031-3203(92)90145-9.

[20] Mandeville, J. R. (1985). Novel Method for Analysis of Printed Circuit Images. IBM Journal of Research and Development, 29(1), 73–86. doi:10.1147/rd.291.0073.

[21] Belbachir, A. N., Lera, M., Fanni, A., & Montisci, A. (2005). An automatic optical inspection system for the diagnosis of printed circuits based on neural networks. Conference Record - IAS Annual Meeting (IEEE Industry Applications Society), 1, 680–684. doi:10.1109/IAS.2005.1518381.

[22] Kobayashi, S., Tanimura, Y., & Yotsuya, T. (1990). Identifying solder surface orientation from color highlight images. IECON Proceedings (Industrial Electronics Conference), 1, 821–825. doi:10.1109/iecon.1990.149246.

[23] Klein Wassink, R. J. (1984). Soldering in Electronics. In Soldering in Electron. Electrochemical Pub. doi:10.1016/0026-2714(90)90700-w.

[24] Metzner, M., Fiebag, D., Mayr, A., & Franke, J. (2019). Automated optical inspection of soldering connections in power electronics production using convolutional neural networks. 2019 9th International Electric Drives Production Conference, EDPC 2019 - Proceedings. doi:10.1109/EDPC48408.2019.9011820.

[25] Huang, W., & Wei, P. (2019). A PCB Dataset for Defects Detection and Classification. ArXiv, 1901 08204. Available online: http://arxiv.org/abs/1901.08204 (accessed on May 2021).

[26] Tang, S., He, F., Huang, X., & Yang, J. (2019). Online PCB Defect Detector on a New PCB Defect Dataset. ArXiv. Available online: http://arxiv.org/abs/1902.06197 (accessed on May 2021).

[27] Wu, F., & Zhang, X. (2014). An inspection and classification method for chip solder joints using color grads and Boolean rules. Robotics and Computer-Integrated Manufacturing, 30(5), 517–526. doi:10.1016/j.rcim.2014.03.003.

[28] Fupei Wu, & Xianmin Zhang. (2011). Feature-Extraction-Based Inspection Algorithm for IC Solder Joints. IEEE Transactions on Components, Packaging and Manufacturing Technology, 1(5), 689–694. doi:10.1109/tcpmt.2011.2118208

[29] Acciani, G., Brunetti, G., & Fornarelli, G. (2006). Application of neural networks in optical inspection and classification of solder joints in surface mount technology. IEEE Transactions on Industrial Informatics, 2(3), 200–209. doi:10.1109/TII.2006.877265.

[30] Fanni, A., Lera, M., Marongiu, E., & Montisci, A. (2001). Neural Network Diagnosis for Visual Inspection in Printed Circuit Boards. 11th International Workshop on Principles of Diagnosis, 47–54.

[31] Ko, K. W., & Cho, H. S. (2000). Solder joints inspection using a neural network and fuzzy rule-based classification method. IEEE Transactions on Electronics Packaging Manufacturing, 23(2), 78. doi:10.1109/6104.846932.

[32] Xie, H., Zhang, X., Kuang, Y., & Ouyang, G. (2011). Solder joint inspection method for chip component using improved adaboost and decision tree. IEEE Transactions on Components, Packaging and Manufacturing Technology, 1(12), 2018–2027. doi:10.1109/TCPMT.2011.2168531.

[33] Wu, H., Zhang, X., Xie, H., Kuang, Y., & Ouyang, G. (2013). Classification of solder joint using feature selection based on bayes and support vector machine. IEEE Transactions on Components, Packaging and Manufacturing Technology, 3(3), 516–522. doi:10.1109/TCPMT.2012.2231902.

[34] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 84–90. doi:10.1145/3065386.

[35] Sze, V., Chen, Y. H., Yang, T. J., & Emer, J. S. (2017). Efficient Processing of Deep Neural Networks: A Tutorial and Survey. Proceedings of the IEEE, 105(12), 2295–2329. doi:10.1109/JPROC.2017.2761740.

[36] Zhao, Z. Q., Zheng, P., Xu, S. T., & Wu, X. (2019). Object Detection with Deep Learning: A Review. IEEE Transactions on Neural Networks and Learning Systems, 30(11), 3212–3232. doi:10.1109/TNNLS.2018.2876865.

[37] Agarwal, S., Terrail, J. O. Du, & Jurie, F. (2018). Recent Advances in Object Detection in the Age of Deep Convolutional Neural Networks. ArXiv. Available online: https://arxiv.org/abs/1809.03193v2. (accessed on May 2021).

[38] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 580–587. doi:10.1109/CVPR.2014.81.

[39] Uijlings, J. R. R., Van De Sande, K. E. A., Gevers, T., & Smeulders, A. W. M. (2013). Selective search for object recognition. International Journal of Computer Vision, 104(2), 154–171. doi:10.1007/s11263-013-0620-5.

[40] Girshick, R., (2015). Fast R-CNN. ArXiv. Available online: https://arxiv.org/abs/1504.08083 (accessed on May 2021).

[41] Ren, S., He, K., Girshick, R., Sun, J., & Faster, R. (2019). Towards real-time object detection with region proposal networks, 2016. ArXiv. Available online: https://arxiv.org/abs/1506.01497 (accessed on June 2021).

[42] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., & LeCun, Y. (2014). Overfeat: Integrated recognition, localization and detection using convolutional networks. 2nd International Conference on Learning Representations, ICLR 2014 - Banff, AB, Canada, 1312-6229.

[43] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2016-December, 779–788. doi:10.1109/CVPR.2016.91.

[44] Redmon, J., Farhadi, A., & YOLO9000. (2017). Better, Faster, Stronger. IEEE Conference on Computer Vision and Pattern Recognition CVPR, Honolulu, HI, USA, 517–6525.

[45] Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. ArXiv. Available online: http://arxiv.org/abs/1804.02767 (accessed on May 2021).

[46] Lin, Y. L., Chiang, Y. M., & Hsu, H. C. (2018). Capacitor Detection in PCB Using YOLO Algorithm. International Conference on System Science and Engineering, ICSSE 2018. doi:10.1109/ICSSE.2018.8520170.

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

[48] Khare, T., Bahel, V., & Phadke, A. C. (2020). PCB-Fire: Automated Classification and Fault Detection in PCB. MPCIT 2020 - Proceedings: IEEE 3rd International Conference on "Multimedia Processing, Communication and Information Technology," 123–128. doi:10.1109/MPCIT51588.2020.9350324.

[49] Pramerdorfer, C., & Kampel, M. (2015). A dataset for computer-vision-based PCB analysis. Proceedings of the 14th IAPR International Conference on Machine Vision Applications, Tokyo, Japan, 378–381. doi:10.1109/MVA.2015.7153209.

[50] Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. 3rd International Conference on Learning Representations, ICLR 2015 - San Diego, CA, USA.

[51] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, Boston, Massachusetts, USA, 1-9.

[52] Lin, M., Chen, Q., & Yan, S. (2013). Network in network. ArXiv. Available online: https://arxiv.org/abs/1312.4400 (accessed on December 2021).

[53] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2016-December, Cancun, Mexico, 770–778. doi:10.1109/CVPR.2016.90.

[54] Luo, P., Wang, X., Shao, W., & Peng, Z. (2018). Towards understanding regularization in batch normalization. ArXiv. Available online: https://arxiv.org/abs/1809.00846 (accessed on December 2021).

[55] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15, 1929–1958.

Appendix I

The architectures of the models used are given below. Convolutional layer type is composed of convolution layer followed by batch normalization and Leaky ReLU activation layer.

Table A1. YOLO Architecture

	Type	Filters	Size / Stride	Output
	Convolutional	32	3 × 3 / 1	448 × 448
	Convolutional	64	3 × 3 / 2	224 × 224
	Convolutional	32	1 × 1 / 1	224 × 224
1 ×	Convolutional	64	3 × 3 / 1	224 × 224
	Residual			
	Convolutional	128	3 × 3 / 2	112 × 112
	Convolutional	64	1 × 1 / 1	112 × 112
2 ×	Convolutional	128	3 × 3 / 1	112 × 112
	Residual			
	Convolutional	256	3 × 3 / 2	56 × 56
	Convolutional	128	1 × 1 / 1	56 × 56
8 ×	Convolutional	256	3 × 3 / 1	56 × 56
	Residual			
	Convolutional	512	3 × 3 / 2	28 × 28
	Convolutional	256	1 × 1 / 1	28 × 28
8 ×	Convolutional	512	3 × 3 / 1	28 × 28
	Residual			
	Convolutional	1024	3 × 3 / 2	14 × 14
	Convolutional	512	1 × 1 / 1	14 × 14
4 ×	Convolutional	1024	3 × 3 / 1	14 × 14
	Residual			
	Convolutional	512	1 × 1 / 1	14 × 14
	Convolutional	1024	3 × 3 / 1	14 × 14
	Convolutional	512	1 × 1 / 1	14 × 14
	Convolutional	1024	3 × 3 / 1	14 × 14
	Convolutional	512	1 × 1 / 1	14 × 14
	Convolutional	1024	3 × 3 / 1	14 × 14
	Convolutional	255	1 × 1 / 1	14 × 14
	Convolution	6	1 × 1 / 1	14 × 14
	Sigmoid			

Table A2. Model1 Architecture

	Type	Filters	Size / Stride	Output
	Convolutional	32	3 × 3 / 1	448 × 448
	Max Pooling		2 × 2 / 2	224 × 224
	Convolutional	64	3 × 3 / 1	224 × 224
	Max Pooling		2 × 2 / 2	112 × 112
	Convolutional	128	3 × 3 / 1	112 × 112
	Max Pooling		2 × 2 / 2	56 × 56
	Convolutional	256	3 × 3 / 1	56 × 56
	Max Pooling		2 × 2 / 2	28 × 28
	Convolutional	512	3 × 3 / 1	28 × 28
	Max Pooling		2 × 2 / 2	14 × 14
	Convolutional	1024	3 × 3 / 1	14 × 14
	Dropout		0.5	14 × 14
	Convolutional	6	1 × 1 / 1	14 × 14
	Sigmoid			

Table A3. Model2 Architecture

	Type	Filters	Size / Stride	Output
3 ×	Convolutional	64	3 × 3 / 1	448 × 448
	Convolutional	32	1 × 1 / 1	448 × 448
	Max Pooling		2 × 2 / 2	224 × 224
3 ×	Convolutional	128	3 × 3 / 1	224 × 224
	Convolutional	64	1 × 1 / 1	224 × 224
	Max Pooling		2 × 2 / 2	112 × 112
4 ×	Convolutional	256	3 × 3 / 1	112 × 112
	Convolutional	128	1 × 1 / 1	112 × 112
	Max Pooling		2 × 2 / 2	56 × 56
	Convolutional	256	3 × 3 / 1	56 × 56
	Max Pooling		2 × 2 / 2	28 × 28
	Convolutional	512	3 × 3 / 1	28 × 28
	Max Pooling		2 × 2 / 2	14 × 14
	Convolutional	1024	3 × 3 / 1	14 × 14
	Dropout		0.5	14 × 14
	Convolutional	6	1 × 1 / 1	14 × 14
	Sigmoid			

Table A4. Model3 Architecture

	Type	Filters	Size / Stride	Output
	Convolutional	32	3 × 3 / 1	448 × 448
	Max Pooling		2 × 2 / 2	224 × 224
	Convolutional	64	3 × 3 / 1	224 × 224
	Max Pooling		2 × 2 / 2	112 × 112
	Convolutional	128	3 × 3 / 1	112 × 112
	Max Pooling		2 × 2 / 2	56 × 56
4 ×	Convolutional	512	3 × 3 / 1	56 × 56
	Convolutional	256	1 × 1 / 1	56 × 56
	Max Pooling		2 × 2 / 2	28 × 28
3 ×	Convolutional	1024	3 × 3 / 1	28 × 28
	Convolutional	512	1 × 1 / 1	28 × 28
	Max Pooling		2 × 2 / 2	14 × 14
3 ×	Convolutional	2048	3 × 3 / 1	14 × 14
	Convolutional	1024	1 × 1 / 1	14 × 14
	Max Pooling		2 × 2 / 2	14 × 14
	Dropout		0.5	14 × 14
	Convolutional	6	1 × 1 / 1	14 × 14
	Sigmoid			

Table A5. Model4 Architecture

Type	Filters	Size / Stride	Output
Convolutional	32	3 × 3 / 1	448 × 448
Max Pooling		2 × 2 / 2	224 × 224
Convolutional	64	3 × 3 / 1	224 × 224
Residual			
Max Pooling		2 × 2 / 2	112 × 112
Convolutional	128	3 × 3 / 1	112 × 112
Residual			
Max Pooling		2 × 2 / 2	56 × 56

	Convolutional	256	3 × 3 / 1	56 × 56
	Residual			
	Max Pooling		2 × 2 / 2	28 × 28
	Convolutional	512	3 × 3 / 1	28 × 28
	Residual			
	Max Pooling		2 × 2 / 2	14 × 14
	Convolutional	1024	3 × 3 / 1	14 × 14
	Residual			
	Convolutional	2048	1 × 1 / 1	14 × 14
	Dropout		0.5	14 × 14
	Convolutional	6	1 × 1 / 1	14 × 14
	Sigmoid			

Table A6. Model5 Architecture

	Type	Filters	Size / Stride	Output
	Convolutional	32	3 × 3 / 1	448 × 448
	Inception			448 × 448
	Max Pooling		2 × 2 / 2	224 × 224
	Convolutional	64	3 × 3 / 1	224 × 224
	Inception			224 × 224
2 ×	Convolutional	32	1 × 1 / 1	224 × 224
	Residual			
	Max Pooling		2 × 2 / 2	112 × 112
	Convolutional	128	3 × 3 / 1	112 × 112
	Inception			112 × 112
	Convolutional	64	1 × 1 / 1	112 × 112
4 ×	Convolutional	128	3 × 3 / 1	112 × 112
	Residual			
	Max Pooling		2 × 2 / 2	56 × 56
	Convolutional	256	3 × 3 / 1	56 × 56
	Inception			56 × 56
	Convolutional	128	1 × 1 / 1	56 × 56
4 ×	Convolutional	256	3 × 3 / 1	56 × 56
	Residual			
	Max Pooling		2 × 2 / 2	28 × 28
	Convolutional	512	3 × 3 / 1	28 × 28
	Residual			
	Convolutional	256	1 × 1 / 1	28 × 28
	Convolutional	512	3 × 3 / 1	28 × 28
	Max Pooling		2 × 2 / 2	14 × 14
	Convolutional	1024	3 × 3 / 1	14 × 14
	Residual			
	Convolutional	2048	1 × 1 / 1	14 × 14
	Dropout		0.5	14 × 14
	Convolutional	6	1 × 1 / 1	14 × 14
	Sigmoid			