electronics

MDPI

Article

Enhancement of Product-Inspection Accuracy Using Convolutional Neural Network and Laplacian Filter to Automate Industrial Manufacturing Processes

Hyojae Jun [1] and Im Y. Jung [2,*]

[1] School of Electronics Engineering, Kyungpook National University, Daegu 41566, Republic of Korea
[2] School of Electronic and Electrical Engineering, Kyungpook National University, Daegu 41566, Republic of Korea
* Correspondence: iyjung@ee.knu.ac.kr

Abstract: The automation of the manufacturing process of printed circuit boards (PCBs) requires accurate PCB inspections, which in turn require clear images that accurately represent the product PCBs. However, if low-quality images are captured during the involved image-capturing process, accurate PCB inspections cannot be guaranteed. Therefore, this study proposes a method to effectively detect defective images for PCB inspection. This method involves using a convolutional neural network (CNN) and a Laplacian filter to achieve a higher accuracy of the classification of the obtained images as normal and defective images than that obtained using existing methods, with the results showing an improvement of 11.87%. Notably, the classification accuracy obtained using both a CNN and Laplacian filter is higher than that obtained using only CNNs. Furthermore, applying the proposed method to images of computer components other than PCBs results in a 5.2% increase in classification accuracy compared with only using CNNs.

Keywords: printed circuit board; product inspection; Laplacian filter; convolutional neural network

Citation: Jun, H.; Jung, I.Y. Enhancement of Product-Inspection Accuracy Using Convolutional Neural Network and Laplacian Filter to Automate Industrial Manufacturing Processes. *Electronics* **2023**, *12*, 3795. https://doi.org/10.3390/electronics12183795

Academic Editor: Juan M. Corchado, Byung-Gyu Kim, Carlos A. Iglesias, In Lee, Fuji Ren and Rashid Mehmood

Received: 1 August 2023
Revised: 5 September 2023
Accepted: 6 September 2023
Published: 7 September 2023

1. Introduction

The key aspect of the fourth industrial revolution (i.e., Industry 4.0) involves the automation of industrial processes to attain high product quality, precision, and reliability [1]. Figure 1a shows the sequence involved in the automation of industrial processes, and Figure 1b illustrates the process of inspecting manufactured goods using photographs captured during the inspection stage.

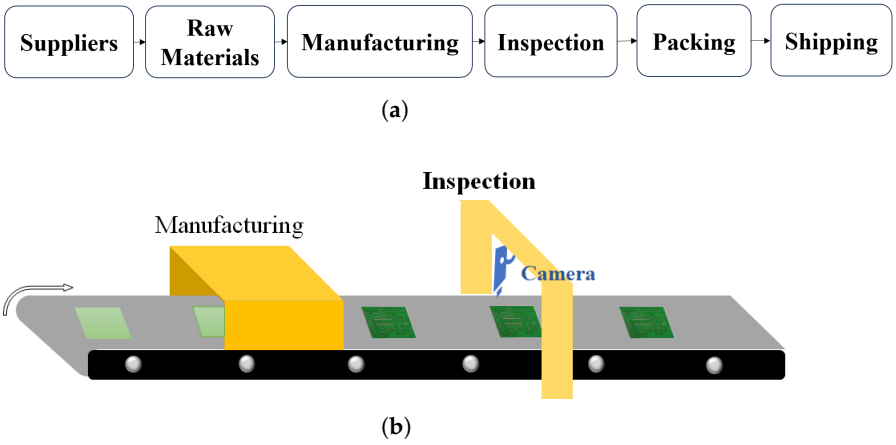


Figure 1. Industrial process automation. (**a**) Automation of industrial manufacturing processes. (**b**) Automated inspection process.

The products that are inspected using cameras during the inspection stage include printed circuit boards (PCBs); PCBs form an essential part in the manufacturing process of electronics [2]. PCBs are used in nearly all electronic devices, including laptops, digital watches, and smartphones. Moreover, PCBs function as a base to electrically connect different components within electronic devices. Figure 2 presents an example of the PCB images captured using cameras during the inspection stage; the presented image was obtained from a publicly available PCB-image dataset (The Open Lab on Human Robot Interaction, Peking University) [3].

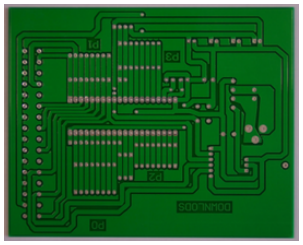
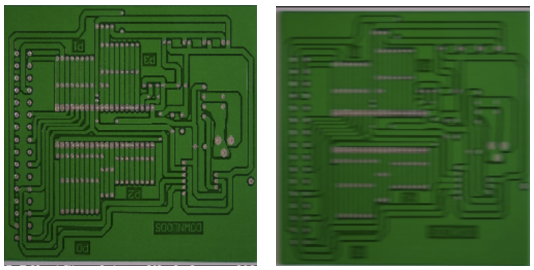


Figure 2. Example of a printed circuit board (PCB) image.

Manufacturing companies have been employing manual visual-inspection methods to detect defects in various products, including PCBs [4]. However, manual inspection methods can cause detection errors owing to the involved subjective estimation by inspectors [5]; moreover, they are expensive owing to the requirement of numerous inspection specialists. With the advancements in the fields of integrated circuits and semiconductors, PCBs have become smaller and more intricate [6], posing challenges for defect detection during their mass-production [7].

Therefore, PCB production and inspection are automated (Figure 1). During an automated PCB manufacturing process, the images captured using cameras are utilized for automatic PCB-defect detection [8]. However, if these images are defective (Figure 3b), the accuracy of PCB-defect detection via image inspection is compromised. Park et al. [9] primarily focused on conducting PCB-defect detection without specifically classifying the obtained images as defective and normal images. In cases where PCB-defect detection is performed using a dataset of mixed normal and defective images, the detection accuracy drops to 73.6%. Notably, studies have been conducted to classify collected images as defective and normal images using convolutional neural networks (CNNs). However, cases exist where normal images were incorrectly classified as defective images [10].



(**a**) (**b**)

Figure 3. Example of a normal and a defective PCB image. (**a**) Normal PCB image. (**b**) Defective PCB image.

To address these issues, this study proposes a method that employs CNNs and a Laplacian filter. The method involves first using a CNN to classify PCB images as normal and defective images and then improving the accuracy of this classification using a Laplacian filter. Although the ultimate goal is to increase the inspection accuracy of products in camera-image–based industrial process automation, the objective of this paper is to propose a method to filter out defective images and mitigate the accuracy reduction due to defective images; notably, these images were not excluded in related works.

The contributions of this study can be summarized as follows.

- The accuracy of PCB-defect detection is 11.87% higher when including defective-image filtering than that obtained without defective-image filtering.
- The accuracy of classifying collected images into normal and defective categories is significantly enhanced when utilizing both CNNs and Laplacian filters, compared with using CNNs alone.
- Defective-image filtering allows for the identification of the underlying causes of defects.
- With regard to images of electronic products other than PCBs, the classification accuracy obtained using our proposed method is 5.2% higher than that obtained using only CNNs.

The remainder of this paper is organized as follows. Section 2 discusses related studies on PCB-defect inspection and defective-image classification. Section 3 introduces our method of detecting PCB defective images using a CNN and Laplacian filter. Section 4 discusses the experimental configuration for filtering defective images (The code and data are contained at https://github.com/junhj14/pcb (accessed on 1 August 2023)) . Section 5 presents the results of defective-image filtering for the proposed method based on a confusion matrix and accuracy. Finally, Section 6 presents the conclusions of this study.

2. Related Works

This section provides an overview of the existing studies on PCB-defect inspection and defective-image determination. As shown in Table 1, PCB-defect detection has been studied with normal or defective images. Meanwhile, defective images have been detected using filters or via deep learning.

Table 1. Existing works on printed-circuit-board (PCB)-defect detection and defective-image detection.

Method		Existing Works
PCB-defect detection	Normal Image	[3,11–15]
	Defective Image	[9]
Defective-image detection	Filter	[8,16–18]
	Deep learning	[19–23]

2.1. PCB-Defect Detection

Huang et al. [3] utilized the PCB-defect dataset from Kaggle [24] to classify six types of defects in PCB images—missing hole, mouse bite, open circuit, short, spur, and spurious copper—using CNN. This approach could only identify types of defects within normal images, not in defective images. Niu et al. [11] used the same dataset and classification classes as used in [3], but employed the Regions with Convolutional Neural Networks features (R-CNN) method to achieve an improved accuracy. In [13,14], the YOLO model was employed to classify PCB defect types. In [15], the YOLOv5 model was employed to enhance the classification of PCB defect types using the same dataset in [3]. Furthermore, in [12], PCB images were classified as either defective or normal images based on the ResNet method. This approach allowed them to distinguish between normal and defective PCBs, in contrast to the method used in [3]. Niu et al. [11] used the ResNet50 method and the Faster R-CNN method for further enhancement of the detection accuracy. All these studies [3,11–15] exclusively used normal images during PCB-defect detection.

However, in PCB-manufacturing automation systems, various types of defective images could be obtained. Suhasini et al. [9] used the same dataset as that used in [3]. However, in contrast to the method presented in [3], an additional class of normal PCB was included, resulting in a total of seven classes for classification. This study expanded the classification to include both defect types and normal PCB images. Additionally, the research focused on the normal images as well as the defective images of PCBs. To enhance the accuracy of PCB-defect detection in defective images, they employed a method to

inspect an enlarged specific area of the PCB image. For defect detection in only the normal PCB image, a classification (normal or defective) accuracy of 98.3% was achieved, while defect detection using a partial specific area of PCB images yielded an accuracy of 98.8%. In addition, defect detection in only a defective PCB image achieved an accuracy of 71.9%, while that using a partial specific area resulted in an accuracy of 73.6%. However, as the accuracy of defect detection in defective images is still relatively low, we implemented a method of detecting defects in filtered normal images and excluded the defective images.

2.2. Defective-Image Classification

Some studies have already used employed methods to classify images as normal or defective images. Hsu and Chen [16] used the point spread function to numerically measure image blurriness and classify such images as defective images. In addition, they set separate threshold values of images that were uniformly blurry and those that exhibited partial blurriness. In [17], a Laplacian filter was used to detect blurry regions in defective images by analyzing the values of different parts of the image. Li et al. [8] used a Laplacian filter to quantify the degree of blurriness in images. If the blurriness exceeded a certain threshold in the entire image, it was classified as a defective image. This approach could classify normal images and identify defects caused by focus or fast motion. In [18], the grad-CAM method was introduced to classify defective images based on the weights of blurry regions. Although these aforementioned studies [8,16–18] provided methods for classifying images as defective, they did not provide information about the specific causes of defects. Some other studies have focused on classifying images as normal or defective images using deep learning techniques instead of filtering methods. In [22], a DNN was employed to classify everyday images captured by a camera into normal and defective images. In [20], a CNN was used to classify directly captured images into normal and defective images. Hendrycks and Dietterich [23] employed ResNet50 for image classification. Furthermore, in [19,21], a CNN was used to classify images as normal or defective images and identify the specific causes of defects. In [21], four defect types—focus, Gaussian, haze, and motion—were classified into four classes, and in [19], five defect types—lighting, focus, motion, file conversion, and white noise—along with normal images were classified into six classes. Unlike the filtering-based approaches, these studies [19–21] provided methods to classify images and determine the causes of defects. However, the provided methods may, in some cases, misclassify normal images as defective images.

3. Defective-Image Detection Using CNN and Laplacian Filter

We used both a CNN and Laplacian filter to filter defective images. This section introduces our CNN and Laplacian filter, and presents a method for detecting defective images.

3.1. CNN

In the automation of PCB-defect detection, we used a CNN for classifying PCB images as defective and normal images. Our CNN model was implemented using the Keras API in TensorFlow 2.12.0. The main architecture comprises three convolutional, three max-pooling, and two dense layers (Figure 4 and Table 2). Each layer utilizes the ReLU activation function, and the final layer uses Softmax. In addition, the Adam optimizer was used in the model, which accepts RGB images as input, with an input image size of 512 $\times$ 512 pixels. The model was trained at a learning rate of 0.001 for a duration of 10 epochs. The output presents the classification of images into five classes: normal and four defective images (focus, motion, light, and noise).

Table 2. Layers and parameters in the proposed convolutional neural network (CNN).

Layer (Type)	Output Shape	Parameters
conv2d (Conv2D)	(None, 510, 510, 32)	896
max_pooling2d (MaxPooling2D)	(None, 255, 255, 32)	0
conv2d_1 (Conv2D)	(None, 253, 253, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 126, 126, 64)	0
conv2d_2 (Conv2D)	(None, 124, 124, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 62, 62, 128)	0
flatten (Flatten)	(None, 492,032)	0
dense (Dense)	(None, 128)	62,980,224
dense_1 (Dense)	(None, 5)	645
Total params: 63,074,117 Trainable params: 63,074,117 Non-trainable params: 0		

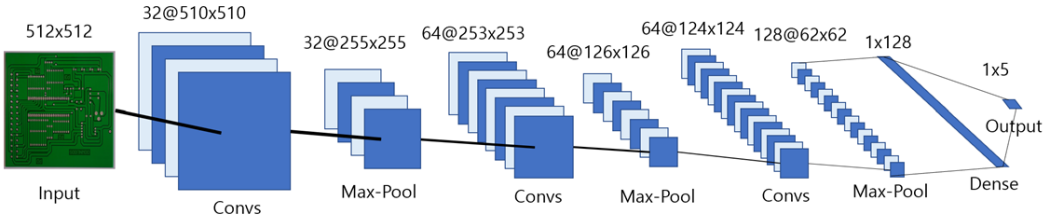


Figure 4. Architecture of our convolutional neural network (CNN).

3.2. Laplacian Filter

A Laplacian operator is defined as a scalar differential operator for scalar functions. In 2D images, the Laplacian operation [25] can be performed using Equation (1). The Laplacian operator highlights areas within the vicinity displaying rapid intensity changes, facilitating edge detection and the detection of strong transitions at boundaries in digital image processing [26]. By utilizing this principle, we can identify regions with considerable blurriness above a certain threshold. These areas can then be used for detecting defective images.

$$Laplacian(f) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{1}$$

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y) \tag{2}$$

$$Laplacian(f) = \nabla \cdot \nabla f = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y) \tag{3}$$

The application of a Laplacian filter is equivalent to using the Laplacian operator; it helps in achieving an image with a Laplacian mask. Herein, a 3×3 mask was utilized; it comprised a basic mask with a zero corner element, a positive central element, and the remaining elements as -1 (Figure 5) . The 3×3 mask was used to perform the Laplacian operation and obtain the difference between the central pixel and its four neighboring pixels in all directions. In addition, the x-direction partial derivative was computed using Equation (2). Similarly, for obtaining a partial derivative in the y direction, Equation (3) was used. To satisfy Equation (3), the central pixel in the 3×3 mask was multiplied by 4, and the surrounding pixels were set to -1 (the sign being opposite to that of the central pixel). By setting the surrounding pixels to -1, we obtained the mask depicted in Figure 5. Figure 6 shows a pseudo code of our Laplacian filter. When an image is passed into the function $Laplacian\_filter$, it returns the image with the Laplacian filter applied.

0	−1	0
−1	4	−1
0	−1	0

Figure 5. Laplacian mask.

```
Laplacian Filter Mask:
  0 -1  0
 -1  4 -1
  0 -1  0

Function Laplacian_filter(image):
        For each row in range(1, height of image - 1):
                For each col in range(1, width of image - 1):
                        New Pixel Value = Sum of (Neighborhood Pixels * Laplacian Filter Mask)
                        Result Image[row, col] = New Pixel Value

        Return Result Image


Function Normal_Area_Ratio(image):
        Filtered Image = Laplacian_filter(image)
        White Pixels = 0

        squared_mean = Calculate_Squared_Mean(Filtered Image)
        xmean = Calculate_Mean(Filtered Image)
        variance = squared_mean - mean^2

        For each pixel in variance:
                If pixel value ≥ Threshold:
                        White Pixels = White Pixels + 1

        White Ratio = White Pixels / Total Pixels

        Return White Ratio


Normal_Area_Ratio_Value = Normal_Area_Ratio(input_image)
```

Figure 6. Pseudo code of our Laplacian filter.

3.3. Defective-Image Detection

Even after classification using a CNN, normal images may be misclassified as defective images or vice versa. To address this issue, this study proposes a method utilizing both the CNN in Section 3.1 and the Laplacian filter in Section 3.2. First, a CNN classifies PCB images into normal and defective images. Then, a Laplacian filter enhances the classification accuracy. Subsequently, defective PCBs are determined based on the analysis of the classified normal images. The process of defective-image detection via Laplacian filtering involves the following steps:

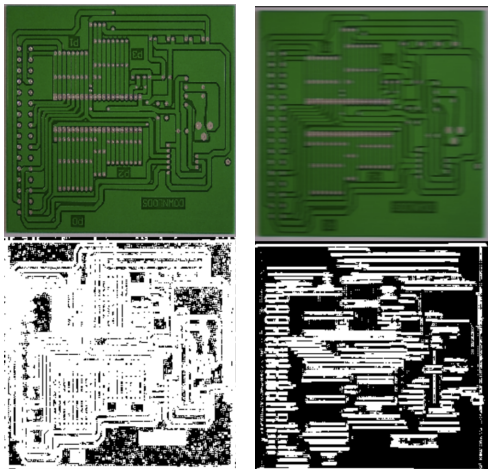
(1) Apply the Laplacian mask to the target image to obtain an image applied with the Laplacian filter.
(2) Calculate the squared values of each pixel in the obtained image.
(3) Use Equation (4) to calculate the variance $V(X)$ of each pixel by considering its surrounding pixels.

$$V(X) = E(X^2) - E(X)^2 \tag{4}$$

(4) Set the pixel values to 0 (indicating a defective region) if they are below the threshold value, and set them to 1 (indicating a normal region) if they are equal to or above the threshold.
(5) Calculate the proportion of the normal regions in the whole image, and use the obtained value to determine the severity of defects.

Figure 7 presents the result after the five-step procedure was applied. As shown, the pixel values below the threshold are labeled as 0 (defective region) in black color

and those above or equal to the threshold are labeled as 1 (normal region) in white color. In Figure 7a, the result (the lower image) of processing a normal PCB image (the upper image) shows that the normal (1) region occupies 78% of the image. Figure 7b presents the result (the lower image) of processing a defective PCB image (the upper image), indicating that the normal region occupies 36% of the image. The difference in the proportion of the normal region between Figure 7a,b is used to classify the image as a normal or a defective image.



(**a**) (**b**)

Figure 7. Detection of the normal area ratio. (**a**) Normal region occupying 78% of the total region in a normal PCB image. (**b**) Normal region occupying 36% of the total region in a defective PCB image.

4. Experiments

Herein, we conducted the following two experiments. Experiment 1 was conducted to enhance the accuracy of PCB-defect inspection by filtering defective images using our method (i.e., using a CNN and Laplacian filter). Experiment 2 was conducted to expand on Experiment 1 by applying the same approach to the images of computer components used in the automation of manufacturing processes to improve the classification accuracy by employing our method.

4.1. Dataset

4.1.1. Public PCB Dataset

This study utilized the PCB-defect dataset from Kaggle [24] (Table 3). There exist 10 types of PCBs (Figure 8). The defect-inspection dataset comprises six classes: Short, Open_circuit, Missing_hole, Spurious_copper, Mouse_bite, and Spur. Figure 9 presents the six types of defective PCB images.

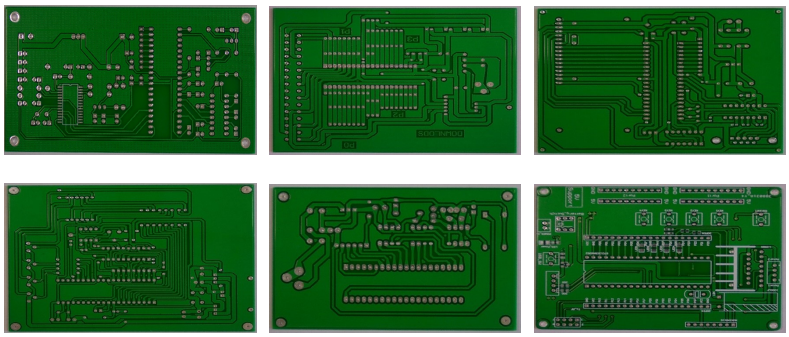


Figure 8. *Cont.*

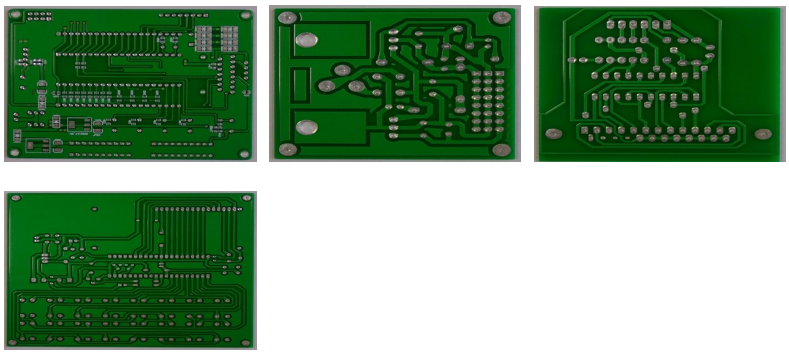


Figure 8. Ten types of normal PCB images.

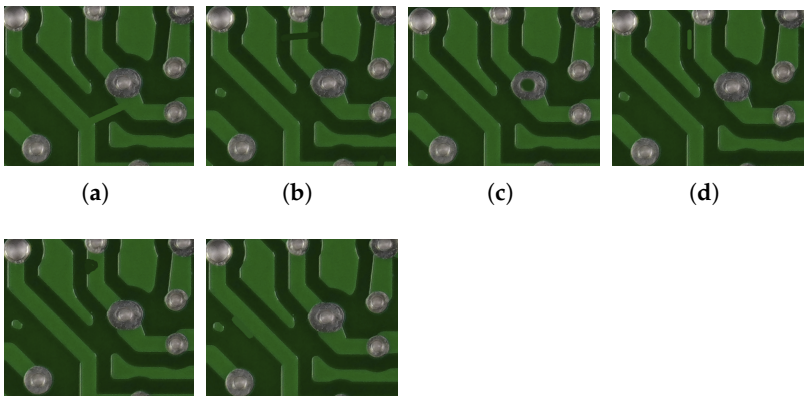


Figure 9. Six types of PCB defects. (**a**) Short. (**b**) Open_circuit. (**c**) Missing_hole. (**d**) Spurious_copper. (**e**) Mouse_bite. (**f**) Spur.

Table 3. PCB dataset from Kaggle [24].

Normal Images	Defective PCB Images					
	Short	Open_Circuit	Missing_Hole	Spurious_Copper	Mouse_Bite	Spur
10	116	116	115	116	115	115

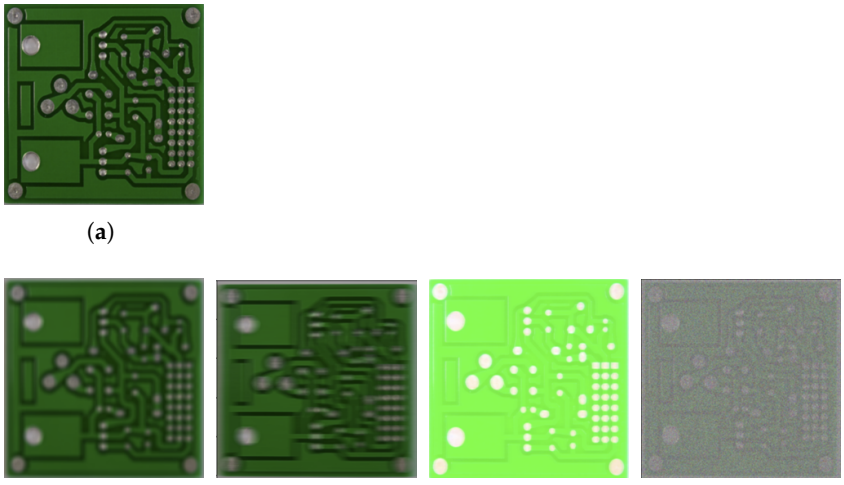
For each PCB defective type, one image was selected from each type of PCB image in Table 3 and 10 images were captured in total. Thereafter, the corresponding defective images were augmented for training our CNN model and evaluating the model, resulting in a total of $10 \times 10 \times 6 = 600$ images (Table 4). For defective-image augmentation, image-processing techniques were applied. Four types of defective images were created considering focus blur (focus), tremors caused by motion (motion), the effects of light (lighting), and noise defects. Figure 10 displays one normal image and four defective images. The focus image of Figure 10b was created using the cv2.GaussianBlur function. The motion image presented in Figure 10c was created by applying the cv2.filter2D function using a kernel that replaces each pixel with the weighted average of its neighboring pixels, thereby imparting a motion effect. The light image of Figure 10d was prepared after adjusting the value channel ("v") of the image using the HSV color space, thereby manipulating the brightness values of individual pixels. The noise image of Figure 10e was prepared after applying noise generated with np.random.normal and subsequently adding it to the image using cv2.add.

In addition, five levels of image processing were applied, resulting in $5 \times 5 = 25$ normal images and 4 defective images. Figure 11 illustrates examples of the five levels of image processing. The dataset was created using 10 normal PCB images from Table 3 and 40 different defective PCB images from Kaggle [27], resulting in a total of 50 images. Com-

bining the four defective images and normal image and the five levels of image processing, a dataset of 50 × 5 × 5 = 1250 samples was generated for training the CNN model.

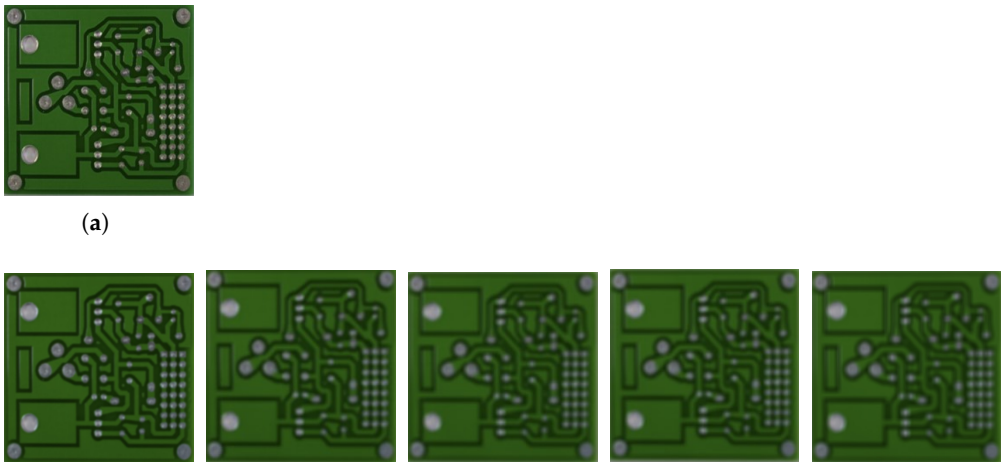
Table 4. Defective PCB-image dataset augmented with the defective PCB images (Table 3).

PCB Defect Type		All		Short		Open _Circuit		Missing _Hole		Spurious _Copper		Mouse _Bite		Spur	
Normal Image (N) Defective Image (D)		N	D	N	D	N	D	N	D	N	D	N	D	N	D
Defective image type	Normal	60	60	10	10	10	10	10	10	10	10	10	10	10	10
	Focus	60	60	10	10	10	10	10	10	10	10	10	10	10	10
	Motion	60	60	10	10	10	10	10	10	10	10	10	10	10	10
	Light	60	60	10	10	10	10	10	10	10	10	10	10	10	10
	Noise	60	60	10	10	10	10	10	10	10	10	10	10	10	10
	Total	600		100		100		100		100		100		100	



(**a**) (**b**) (**c**) (**d**) (**e**)

Figure 10. Normal and defective images of a PCB in Experiment 1. (**a**) Normal Image. (**b**) Focus. (**c**) Motion. (**d**) Light. (**e**) Noise.



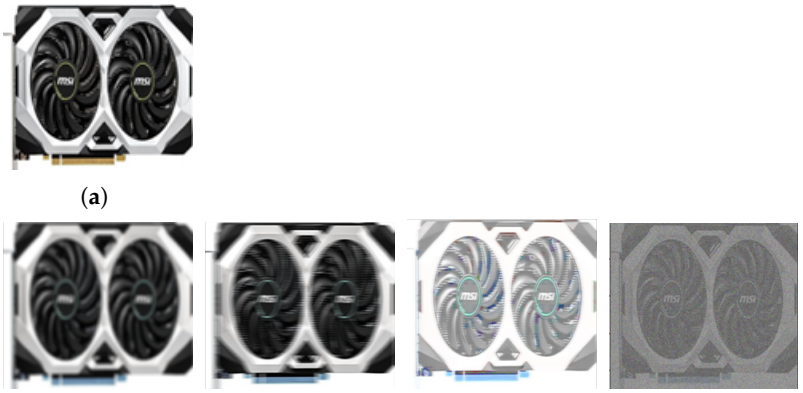
(**a**) (**b**) (**c**) (**d**) (**e**) (**f**)

Figure 11. Examples of the five levels of image processing for Experiment 1. (**a**) Normal Image. (**b**) Focus blur state 1. (**c**) Focus blur state 2. (**d**) Focus blur state 3. (**e**) Focus blur state 4. (**f**) Focus blur state 5.

4.1.2. Experiment 2: Dataset

In Experiment 2, we used the Computer Parts dataset of Roboflow [28]. Industrial automation utilizes images of both PCBs and computer components. The Roboflow Computer Parts dataset comprises images, specifically captured under controlled conditions, with a consistent white background. Similar to the method used in Experiment 1, 50 data were experimented at 25 stages for each normal data, thus totaling 1250 data.

Figures 12 and 13 present the images of a cooling fan, a component of computers. The cooling fan images were the subset of the Roboflow Computer Parts dataset. Figure 12 shows examples of normal and defective images obtained by considering factors such as light, focus, motion, and white noise. Figure 13 presents examples of focus-related defective images created by applying a five-level blurring effect to the normal images. The techniques employed for creating these images are consistent with the methods elucidated for Experiment 1.



(**a**)

(**b**) (**c**) (**d**) (**e**)

Figure 12. Examples of defective images caused by various factors in Experiment 2. (**a**) Normal Image. (**b**) Focus. (**c**) Motion. (**d**) Light. (**e**) Noise.

(**a**)

(**b**) (**c**) (**d**) (**e**) (**f**)

Figure 13. Examples of the five levels of image processing for Experiment 2. (**a**) Normal Image. (**b**) Focus blur state 1. (**c**) Focus blur state 2. (**d**) Focus blur state 3. (**e**) Focus blur state 4. (**f**) Focus blur state 5.

4.2. Experimental Environment

We executed two experiments. Experiment 1 measures the defect-detection accuracy in PCB images. Experiment 2, which extends Experiment 1, applies our method to the images of computer components.

4.2.1. Experiment 1

Experiment 1 was executed in the Google Colab environment. First, the images were classified using our CNN model (Table 2 and Figure 4). To create the CNN model, the dataset with 1250 images was split into training and test data in a 4:1 ratio (i.e., 1000 images for training data and 250 images for test data). After classifying the images as normal and defective images by using our CNN model, the Laplacian filter in Figure 6 was applied. Next, we calculated the threshold $V(x)$ using Equation (4) with the largest difference in the area ratios between the normal and defective images. In this experiment, a threshold of 100 was used, thus resulting in a normal-image ratio of 65–80% and a defective-image ratio of 20–45% after the application of the Laplacian filter. Based on these values, images predicted as defective by the CNN with a normal-image ratio of $\geq$65% were classified as normal images, whereas images predicted as normal images with a normal-image ratio of $\leq$45% were classified as defective images.

4.2.2. Experiment 2

This experiment was also conducted in the Google Colab environment. First, the images were classified using the CNN (Table 2 and Figure 4). To create the CNN model, the dataset was split into the training and test data in a 4:1 ratio (1000 images for training data and 250 images for test data, similar to Experiment 1). The experiment was performed for 50 epochs. After classifying the images into normal and defective images using our CNN, the Laplacian filter in Figure 6 was applied. Next, we calculated the threshold $V(x)$ using Equation (4) with the largest difference in the area ratios between the normal and defective images. For example, for the cooling fan image shown in Figures 12 and 13, a threshold of 100 was used, thus resulting in a normal-image ratio of 60–85% and a defective-image ratio of 10–35%. Based on these values, images predicted as defective images by the CNN with a normal image ratio of $\geq$60% were classified as normal images, whereas images predicted as normal images with a normal-image ratio of $\leq$35% were classified as defective images.

5. Evaluation and Analysis

5.1. Results of Experiment 1: PCB-Defect Detection

The defect inspection was performed using the ResNet50 model, which is a state-of-the-art deep learning model [3,11], to compare it with our model. We used the dataset presented in Table 4, which comprised 600 normal images. These images were organized based on PCB and image defect types, with 10 instances for each combination of PCB and image defect type. Figure 14 shows the PCB-defect detection accuracy before defective-image filtering for all 600 images as well as for each defect type having 100 instances. The accuracy was assessed by correctly discerning between the normal and defective PCB classifications in a set of 100 images. Figure 14 shows the accuracies for each defect type: 84%, 88%, 84%, 87%, 86%, and 88% for Short, Open_circuit, Missing_hole, Spurious_copper, Mouse_bite, and Spur, respectively. It demonstrates that the overall defect-inspection accuracy before applying the defective-image filtering was 86.13% for all images.

Figure 15 shows the confusion matrix of the defective image classification using our CNN on the same dataset in Table 4. The classification results using our CNN achieved an accuracy of 98%.

Figure 16 shows the confusion matrix of the defective-image detection using our CNN on the dataset presented in Table 4. The classification results using our CNN achieved an accuracy of 98%. And, in Figure 16, PCB-defect detection using our CNN and Laplacian filters exhibited 99% accuracy. As shown in Figures 15 and 16, when a PCB image is classified as a defective image, the defect cause is classified into one of the four causes shown in Figure 10.

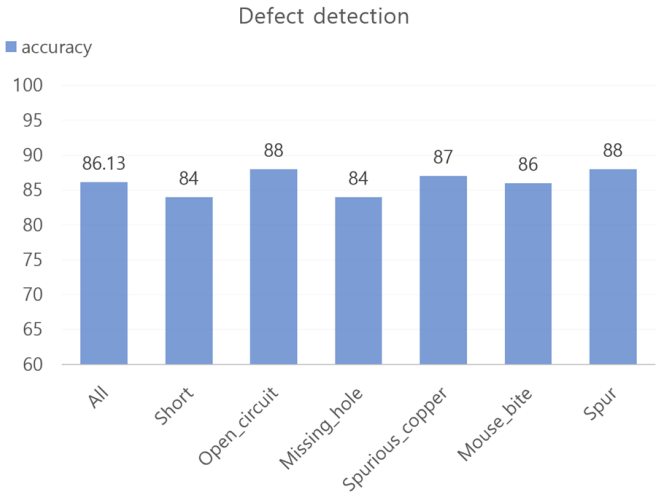


Figure 14. PCB-defect detection accuracy before defective-image filtering using ResNet50.

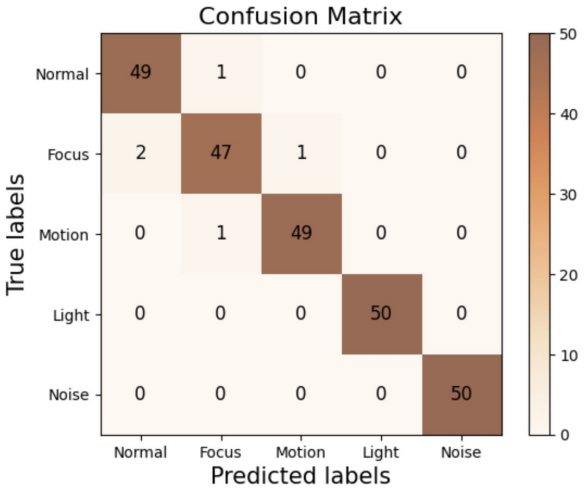


Figure 15. Image-classification confusion matrix using our CNN.

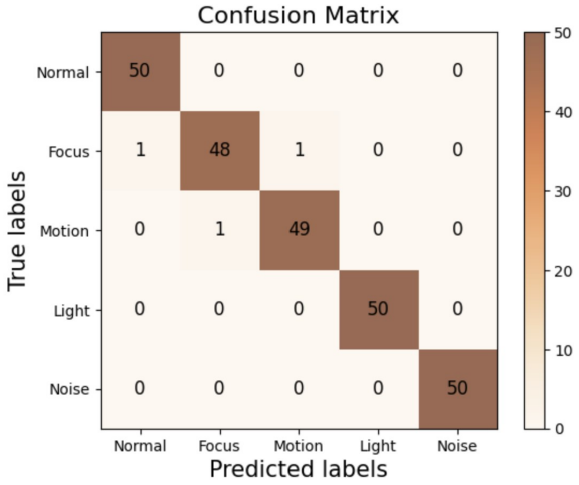


Figure 16. Image-classification confusion matrix using our CNN and a Laplacian filter.

Figure 17 shows the accuracy for each defect type before and after defective-image filtering. As the accuracy of the defect detection is low when defective images are present, the accuracy can be improved by performing defect detection only on normal images after filtering defective images. As a result, the accuracy for each defect type improved after

filtering, compared with before filtering. The accuracy of inspecting the defects for all images before the filtering process was 86.13%, on average. After filtering the defective images, as shown in Figure 16, 51 images were classified as normal images. The accuracy was assessed by correctly discerning between normal and defective PCB classifications in a set of 51 images. When conducting defect-detection experiments on these filtered 51 defective images, the defect-detection accuracy improved to 98% on average. Moreover, the filtering process resulted in an 11.87% improvement in the defect-inspection accuracy.

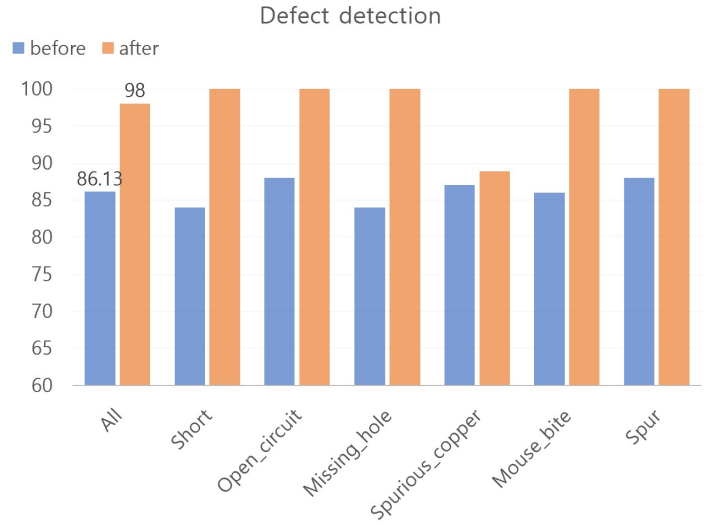


Figure 17. Accuracy of defect detection before and after filtering.

5.2. Results of Experiment 2: Classification of the Images of Computer Parts

In Experiment 2, we used the trained CNN model in Table 2 to predict 250 test data samples. Figure 18 presents the confusion matrix, showing the predicted labels compared with the true labels for the test data. The true labels represent the actual class, whereas the predicted labels represent the predicted class. If the predicted class matches the true class, the classification is considered to be correct. The confusion matrix provides a visual representation of the model performance in classifying the test data. Figure 18 shows that the achieved accuracy was 83.2%.

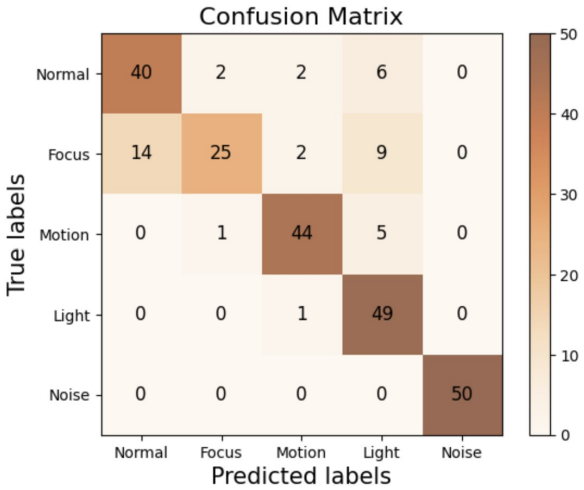


Figure 18. Image-classification confusion matrix using our CNN.

After CNN-based image classification, the misclassified images that were initially classified as defective images were further processed using the Laplacian filter. As shown in Figure 19, these misclassified images were correctly reclassified as normal images, thus improving the accuracy. After applying the Laplacian filter and reclassifying six misclassi-

fied normal images as defective images and seven misclassified defective images as normal images, the accuracy of the defective-image detection improved to 88.4%, representing a 5.2% increase.

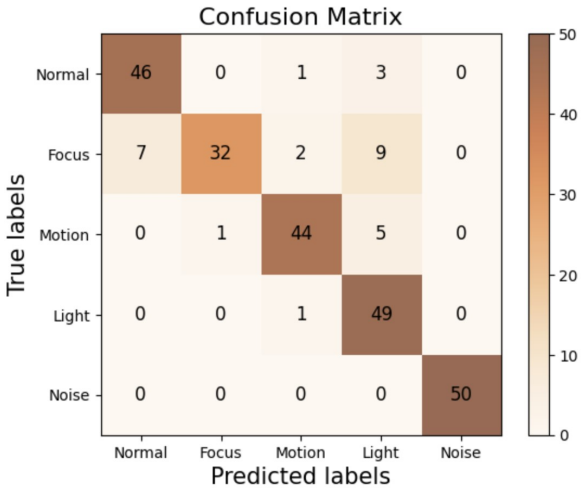


Figure 19. Image-classification confusion matrix using our CNN and a Laplacian filter.

5.3. Comparison

Table 5 provides a comparison between our method and other research methods in terms of PCB-defect detection and defective-image classification. The table compares the aspects of PCB-defect detection, including defective PCB images, image filtering, and the ability to classify defective images. Table 6 presents a comparison between the accuracy results of various methods used for classifying defective images and our method. Notably, the accuracy was calculated by considering all four types of defective images as defective and classifying normal images as normal.

Table 5. Comparison of the proposed method with other classification methods.

Type	Method	PCB-Defect Detection	Inappropriate PCB Images	Image Filtering	Classification of Images
PCB Detection	Normal Image [3,11,15]	O	X	X	X
	Defective Image [9]	O	O	X	X
Image Classification	Filter [8,18,25]	X	X	O	X
	Deep Learning [19,20]	X	X	O	O
Ours		O	O	O	O

Table 6. Comparison of the proposed method with other classification methods.

Method	Precision	Recall	Accuracy	F1 Score
Laplacian [25]	74.13	84	91.2	78.75
DNN [29]	83.92	94	95.2	90.71
CNN [19]	96.07	98	98.8	97.02
Res-Net50 [21]	97.95	96	98.8	96.96
CNN + Laplacian (Ours)	98.03	100	99.6	99.00

6. Conclusions and Future Works

This study proposed a method using a CNN and Laplacian filter for classifying PCB images as normal or defective images, aiming to improve the accuracy of PCB-defect detection in automated PCB-manufacturing processes. By incorporating the Laplacian filter into the CNN model for image classification, we observed an improvement of about 12% in the PCB-defect detection accuracy. Moreover, we demonstrated that the application of the Laplacian filter can enhance the accuracy of defective-image detection in non-PCB images, exhibiting a 5% improvement. These findings suggest the potential applicability of Laplacian filters in enhancing defect-detection accuracies for PCB images, as well as for other types of images.

In the future, we plan to study how to adopt two or more cameras for dual verification and how to process classified defect images in industrial automation.

Author Contributions: H.J. and I.Y.J. conceived and designed the experiments; H.J. performed the experiments; H.J. and I.Y.J. analyzed the data; H.J. wrote the paper; and I.Y.J. reorganized and corrected the paper. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by a National Research Foundation of Korea (NRF) grant funded by the Korea government (No. 2021R1F1A1064345).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Karnik, N.; Bora, U.; Bhadri, K.; Kadambi, P.; Dhatrak, P. A comprehensive study on current and future trends towards the characteristics and enablers of industry 4.0. *J. Ind. Inf. Integr.* **2022**, *27*, 100294. [CrossRef]
2. Zhang, Q.; Liu, H. Multi-scale defect detection of printed circuit board based on feature pyramid network. In Proceedings of the IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), Dalian, China, 28–30 June 2021; pp. 911–914.
3. Huang, W.; Wei, P. A PCB Dataset for Defects Detection and Classification. *arXiv* **2019**, arXiv:1901.08204.
4. Anoop, K.P.; Sarath N.S.; Sasi Kumar, V.V. A review of PCB defect detection using image processing. *Int. J. Eng. Innov. Technol.* **2015**, *4*, 188–192.
5. Suhasini, A.; Sonal, D.K.; Prathiksha, B.G.; Meghashree, B.S.; Phaneendra, H.D. PCB defect detection using image subtraction algorithm. *Int. J. Comput. Sci. Trends Technol.* **2015**, *3*, 8887.
6. Mujeeb, A.; Dai, W.; Erdt, M.; Sourin, A. One class based feature learning approach for defect detection using deep autoencoders. *Adv. Eng. Informat.* **2019**, *42*, 100933. [CrossRef]
7. Băjenescu, T.I. Miniaturisation of electronic components and the problem of device overheating. *Electroteh. Electron. Autom.* **2021**, *69*, 53–58. [CrossRef]
8. Li, Y.; Kuo, P.; Guo, J. Automatic Industry PCB Board DIP Process Defect Detection System Based on Deep Ensemble Self-Adaption Method. *IEEE Trans. Components Packag. Manuf. Technol.* **2021**, *11*, 312–323. [CrossRef]
9. Park, J.-H.; Kim, Y.-S.; Seo, H.; Cho, Y.-J. Analysis of Training Deep Learning Models for PCB Defect Detection. *Sensors* **2023**, *23*, 2766. [CrossRef] [PubMed]
10. Golchubian, A.; Marques, O.; Nojoumian, M. Photo quality classification using deep learning. *Multimed. Tools Appl.* **2021**, *80*, 22193–22208. [CrossRef]
11. Niu, J.; Huang, J.; Cui, L.; Zhang, B.; Zhu, A. A PCB Defect Detection Algorithm with Improved Faster R-CNN. In Proceedings of the International Conference on Big Data & Artificial Intelligence & Software Engineering, Guangzhou, China, 21–23 October 2022.
12. Zhang, H.; Jiang, L.; Li, C. CS-ResNet: Cost-sensitive residual convolutional neural network for PCB cosmetic defect detection. *Expert Syst. Appl.* **2021**, *185*, 115673. [CrossRef]
13. 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. [CrossRef]
14. Santosoa, A.D.; Cahyonoa, F.B.; Prahastab, B.; Sutrisnob, I.; Khumaidi, A. Development of PCB Defect Detection System Using Image Processing With YOLO CNN Method. *Int. J. Artif. Intell. Res.* **2022**, *6*.
15. Tang, J.; Liu, S.; Zhao, D.; Tang, L.; Zou, W.; Zheng, B. PCB-YOLO: An Improved Detection Algorithm of PCB Surface Defects Based on YOLOv5. *Sustainability* **2023**, *15*, 5963. [CrossRef]
16. Hsu, P.; Chen, B. Blurred image detection and classification. *Adv. Multimed. Model.* **2008**, *4903*, 277–286.
17. Shi, J.; Xu, L.; Jia, J. Discriminative Blur Detection Features. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 17–24 June 2014; pp. 2965–2972.
18. Selvaraju, R.R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; Batra, D. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017; pp. 618–626.

19. Aggarwal, N.; Deshwal, M.; Samant, P. A Survey on Automatic Printed Circuit Board Defect Detection Techniques. In Proceedings of the International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE), Greater Noida, India, 28–29 April 2022; pp. 853–856.
20. Szandała, T. Convolutional Neural Network for Blur Images Detection as an Alternative for Laplacian Method. In Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI), Canberra, ACT, Australia, 1–4 December 2020; pp. 2901–2904.
21. Wang, R.; Li, W.; Zhang, L. Blur image identification with ensemble convolution neural networks. *Signal Process.* **2019**, *155*, 73–82. [CrossRef]
22. Yan, R.; Shao, L. Blind Image Blur Estimation via Deep Learning. *IEEE Trans. Image Process.* **2016**, *25*, 1910–1921.
23. Hendrycks, D.; Dietterich, T. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
24. A PCB Dataset Containing 1386 Images with 6 Kinds of Defects. Available online: http://robotics.pkusz.edu.cn/resources/dataset/ (accessed on 20 July 2023).
25. Bansal, R.; Raj, G.; Choudhury, T. Blur Image Detection using Laplacian Operator and Open-CV. In Proceedings of the International Conference on System Modeling & Advancement in Research Trends, Moradabad, India, 25–27 November 2016.
26. Dey, N.; Ashour, A.S.; Fuqian Shi, F.; Balas, V.E. Chapter 9—Dempster-Shafer Fusion for Effective Retinal Vessels' Diameter Measurement. In *Cognitive Systems and Signal Soft Computing Based Medical Image Analysis*; Academic Press: Cambridge, MA, USA, 2018; pp. 149–160
27. A PCB Data. Available online: https://www.kaggle.com/datasets/pkompally/pcb-data (accessed on 17 August 2023).
28. A Computer Parts Dataset. Available online: https://universe.roboflow.com/my-datasets-r9xst/computer-parts (accessed on 20 July 2023).
29. Zhou, Y.; Song, S.; Cheung, N. On classification of distorted images with deep convolutional neural networks. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), New Orleans, LA, USA, 5–9 March 2017.