Learning and Nonlinear Models - Journal of the Brazilian Society on Computational Intelligence (SBIC), Vol. 22, Iss. 1, pp. 17-31, 2024

REAL-TIME DETECTION OF CUSTOMER-INDUCED DAMAGE IN PRINTED CIRCUIT BOARDS USING MOBILE DEVICES AND YOLO DETECTORS

Joao Pedro Santiago, Victor Farias, Lucas Sena, Joao Paulo Pordeus, Javam Machado

Systems and Databases Laboratory

Computer Science Department, Universidade Federal do Ceara

{pedro.santiago, victor.farias, lucas.sena, joao.pordeus, javam.machado}@lsbd.ufc.br

Abstract – The identification of Consumer-induced damage is essential for electronics manufacturers' warranty programs. Consumer Induced Damage (CID) is any damage caused by an unauthorized person, including the consumer. The product warranty does not cover these damages, avoiding expenses in the manufacturer's revenue. The consumer-induced damage warranty process is usually done manually by technically trained people. However, this task demands a lot of attention to detail, can be time-consuming, and is susceptible to human errors. With this in mind, this work presents an object detection model for low-computational-cost devices that uses computer vision and deep learning methods with YOLO detectors embedded in mobile devices to identify consumer-induced damages on printed circuit boards (PCB). We conducted sixteen experiments with four YOLO neural network architectures and successfully developed a mobile application for CID detection. Our best model achieved a mAP@0.5 of 33.1% and an average of 5.7 FPS on real mobile devices.

Keywords – Customer Induced Damage, Printed Circuit Board (PCB), Deep Learning, Computer Vision.

1 Introduction

Currently, electronic device manufacturers have to bear a significant cost to cover the warranty of products that consumers have purchased. In many cases, the consumer causes one or more defects due to improper use or maintenance by an unskilled individual [1]. When the consumer causes the damage, the manufacturers do not cover the repair costs and are referred to as Costumer Induced Damage (CID).

A printed circuit board (PCB) is fundamental to the internal structure of many electronic devices. A PCB consists of a board containing conductive and insulating layers that integrate various electronic and electrical components [1]. Due to the great importance of PCB in the composition of electronic devices, evaluating this structure in the product quality assurance and control process is extremely important. However, the PCB diagnosis process is mostly done manually, through specialized technicians [2]. This task is time-consuming and prone to errors, as it contains defects that can be difficult to identify. Due to these facts, inspection of PCB automatically is considered, and some researchers have concentrated their efforts on performing defect detection on PCB automatically. Recent studies [1] show that it is possible to build defect detection models in PCB using computer vision techniques.

Computer vision is becoming increasingly prevalent in everyday tasks, including image classification, object detection, object segmentation, and face recognition. The primary method that enables these tasks is deep learning, achieved through the use of Convolutional Neural Networks (CNNs), which yield excellent results in these activities [3]. Always seeking better accuracy and precision, deep learning models are generally robust and have a high computational cost, making them inefficient for mobile device use. However, when a PCB arrives at the repair center, the technician responsible for inspecting the part has to check what defects are occurring, and this must be done quickly by a mobile device. A defect detection model for PCBs can be utilized on a cloud server and accessed through an API on a mobile device. However, this approach introduces additional infrastructure, latency, and higher financial costs. Moreover, accessing the external server requires an available internet connection, negatively impacting privacy. In this context, the PCB defect detection model must be executed directly on mobile devices. Mobile devices have low computational power, therefore, some works aim to develop deep learning models for detecting objects with greater performance and fluidity on mobile devices, such as *ShuffleNet* [4], *EfficientDet* [5] and *MobileNet* [6–8].

You Only Look Once (YOLO) has revolutionized the field of object detection in computer vision. Due to its high speed and good accuracy, it has become a very popular choice for various real-time applications. YOLO approaches differ from traditional methods by directly predicting bounding boxes and class probabilities in a single stage, as opposed to the two-stage process of region proposals and classification used in older techniques [9]. This approach significantly reduces computational overhead, making YOLO ideal for resource-constrained environments.

Currently, there is a wide variety of YOLO models based on the original architecture [9–14]. Each of these works strives to improve accuracy and performance.

This work tackles the problem of detecting CIDs using mobile devices with low computational consumption YOLO models.

The work's contributions are summarized as follows:

- We conducted a survey on the state of the art of low-computational-power object detectors.
- We conducted a series of experiments with the models obtained from our survey, using our dataset to compare the results across different YOLO model architectures.
- We deployed our best models on real mobile devices for application validation and performance testing on different devices.

The work is organized as follows. In Chapter 2, all the important theoretical foundations for the reader's understanding are presented. In Chapter 3, related work on this research topic is presented, providing support for this work. In Chapter 4, the steps taken for the execution of this work are presented. Chapter 5 presents the experiments and obtained results. Finally, in Chapter 6, the conclusions obtained in this work are presented.

2 Theoretical Background

2.1 Customer Induced Damage

Damage to electronic components, such as motherboards, can result in malfunctions or failures within the associated devices. The origins of these damages can be varied, and a considerable body of literature predominantly focuses on problems stemming from the manufacturing process [1, 2, 15, 16]. Notably, electrical component companies employ stringent quality control procedures to mitigate manufacturing-induced damages.

Nevertheless, another prevalent source of damage is CID. CID encompasses harm caused by unauthorized individuals, including customers [1]. Such damage can have significant financial implications for companies, particularly in terms of warranty costs. Effective CID detection mechanisms can serve as a means for companies to optimize cost savings.

2.2 Computer Vision

Humans find it easy to perceive structures and recognize objects. Computer Vision uses techniques to describe the world in one or more images and reconstruct its properties, such as shape and color distribution [17].

Computer vision can be applied to various tasks; in image classification tasks, it is possible to analyze images of products on a production line to classify these products automatically [3]. According to [3], object detection aims to locate where an object of interest is in the image, making it possible to detect tumors in brain scans.

According to [18], digital image processing and computer vision are strongly related, and it becomes evident when dividing digital processing into three paradigms (low, medium, and high level). At the low level, there is image pre-processing for noise reduction, contrast, enhancement, and sharpening of images. At the medium level, object segmentation (separation of the image into regions or objects) and classification (recognition) of individual objects. Finally, the high level includes image analysis and vision-related cognitive functions.

2.2.1 Object Detection

Computer vision comprises various core subfields to address visual recognition challenges, including image classification [19], semantic segmentation [20], instance segmentation [21], and object detection [22]. In the image classification problem, given an image, the objective is to recognize semantic categories of objects that appear in the image. On the other hand, semantic segmentation aims to predict pixel classifiers to assign specific category labels to each image pixel. Instance segmentation arises to address a limitation of semantic segmentation: the inability to distinguish between objects of the same class. Instance segmentation can achieve pixel-level localization of each object in addition to recognizing the category of each object [23].

Object detection aims to identify the category (cars, animals, humans) and locate the position of objects in images through bounding boxes. The objective of object detection is to develop computational models and techniques that provide one of the most basic pieces of knowledge necessary for computer vision, such as Where objects are located and what the categories of those objects are [24]. More complex computer vision techniques can be performed through object detection, such as target tracking, event detection, behavior analysis, and semantic scene understanding [25].

The first object detection models used handcrafted image feature extractors, which made the process slow, imprecise, and performed poorly on familiar datasets [26]. Traditional detection algorithms typically involve six steps: preprocessing, window sliding, feature extraction, feature selection, feature classification, and post-processing, tailored for specific recognition tasks. However, they suffer from limitations such as small data size, limited portability, lack of relevance, high time complexity, window redundancy, and limited robustness to diverse changes, resulting in optimal performance only in specific simple environments [25]. In recent years, object detection has been propelled through deep learning [27] methods.

According to [28], deep learning has allowed image classification, object detection, and natural language processing problems to achieve state-of-the-art results, making deep learning very popular in recent years. Something that greatly contributed to the advancements of deep learning-based methods for better results was Convolutional Neural Networks (CNN), which does not require manually creating feature extractors or filters and can achieve good results for large datasets. However, CNNs have a notable drawback. They require large amounts of data to perform effectively and often struggle to deliver satisfactory results with limited data [29].

2.2.2 Data Augmentation

A possible solution to the problem of requiring a large amount of data for a CNN to achieve good results is the use of data augmentation. Data augmentation is a process of artificially increasing the diversity of training data through techniques such as rotation, flipping, and brightness adjustments in images or adding noise to data [30]. This strategy is essential for dealing with limited and unbalanced datasets, as well as for improving the generalization capability of models.

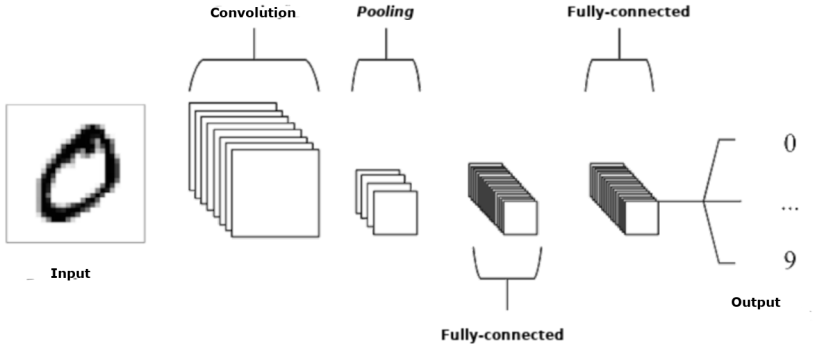
A common issue with image classification datasets, but one that can also occur in object detection, is the challenge of class imbalance. The challenge of imbalance refers to the disparity in the distribution of examples among different classes in training datasets. This imbalance can lead to bias problems during model training, impairing its generalization ability and resulting in inferior performance, especially for minority classes [31].

2.3 Deep Learning for Object Detection

2.3.1 Convolutional Neural Networks

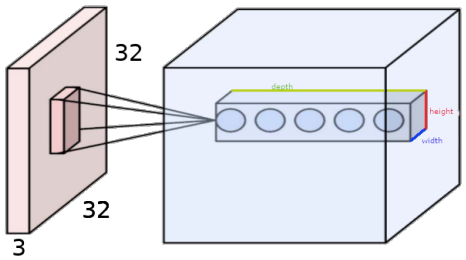
Convolutional Neural Networks perform a mathematical operation called convolution. Generally speaking, convolutional networks are neural networks that use the convolution operation instead of general multiplication in at least one of their layers [27]. Figure 1 shows a simple CNN with five layers.

Figure 1: Architecture of a simple CNN with only five layers.



Although matrices are commonly known as two-dimensional arrays, tensors generalize this concept to encompass arrays with higher dimensions. The convolution operation takes place over three-dimensional tensors known as feature maps, with two spatial axes (height and width) as well as a depth axis [32]. During the convolution operation, output feature maps are generated, which are a three-dimensional tensor with width and height, while their depth is arbitrary since the output depth is a parameter of the layer representing filters, also known as kernels [32]. Figure 2 shows the representation of a CNN.

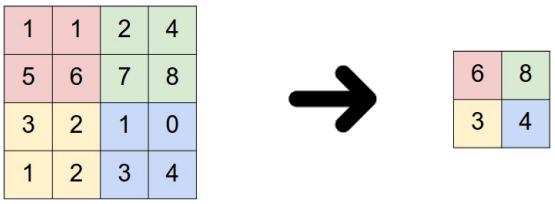
Figure 2: Multiple layers corresponding to different filters in the same image region.



The convolution layer applies several filters to its inputs, being able to detect features in any part of its input. The pooling layer reduces the spatial dimensionality of the input to decrease the number of parameters and computational load [3]. Figure 3 shows the representation of a max-pooling layer, which is the most common type of pooling layer. In the max-pooling operation, each quadrant represented by a color in the image has the maximum value used to represent the layer after the spatial dimensionality reduction.

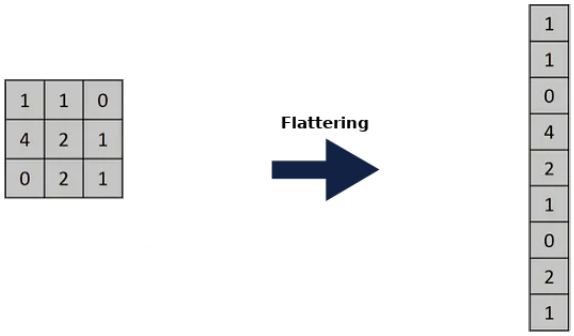
In fully connected layers, also known as dense layers, each neuron is connected to every neuron in the next layer.

Figure 3: Max-Pooling operation on a 4x4 image with a 2x2 filter and a stride of 2.



Between the pooling layer and the dense layer, there is the Flatten layer, which is responsible for converting the data dimension after applying pooling into a one-dimensional format. Figure 4 shows the Flatten operation.

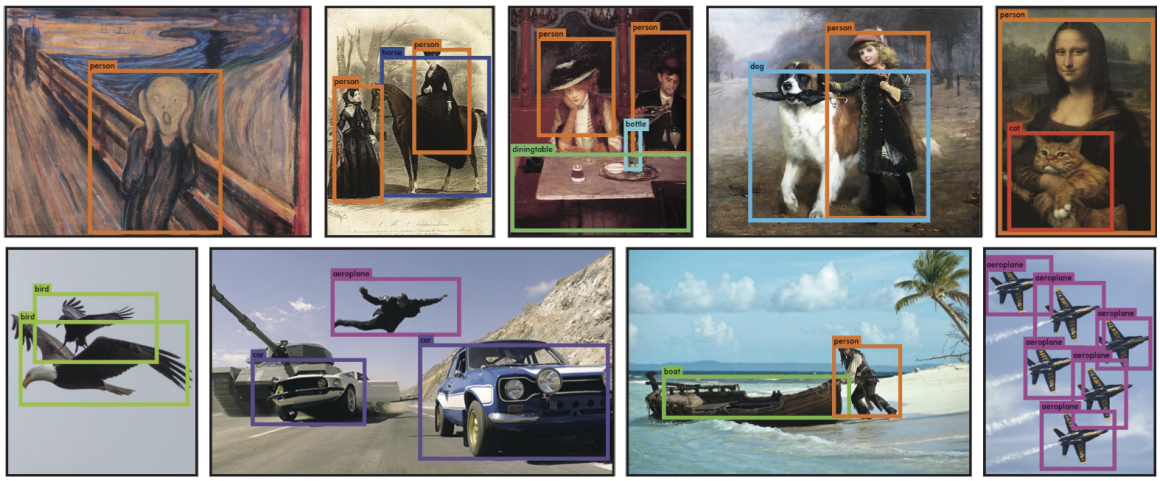
Figure 4: Flatten operation.



2.3.2 YOLO

YOLO was proposed by [9] and consists of a series of state-of-the-art detectors for object detection in real-time applications due to their precision-balanced performance. Detection in YOLO is achieved using only one detection stage to extract image features from a CNN. According to [9], a single CNN simultaneously predicts several bounding boxes and object class probabilities for each of these boxes. In this way, YOLO achieves good performance and accuracy for object detection tasks. Figure 5 shows object detection using YOLO models.

Figure 5: Object detection with YOLO.



YOLOv3 brings improvements compared to its predecessors, presenting a more complex classifier network with better accuracy and without compromising performance [11]. It is also possible to use different backbones for the neural network with YOLOv3, and there is an implementation that uses MobileNetv2 to perform well on low computational power devices.

YOLOv6 [33] brings a better trade-off in terms of accuracy and speed. The main difference of YOLOv6 is the fact that it's anchor-free, providing better generalization capabilities and simplicity in decoding prediction results, and being 51% faster than

the anchor-based YOLO series. Object detection anchor-based consists of predefined bounding boxes as ground truth proposals. Furthermore, it also has improvements designed for industrial applications, such as longer training periods and quantization [34] to reduce the number of bits in the network weights. YOLOv6 v3.0 [13] is an evolution of the YOLOv6 [33] work and brings a new design to the network and training strategy. According to [13], in practice, there was an integration of features at multiple scales, becoming a critical and effective factor in object detection. This new network design, along with the new training strategies, drive YOLOv6 v3.0 for real-time object detection with high accuracy.

YOLOv7 introduces a new approach to real-time object detection methods. Instead of merely optimizing the architecture, the methods proposed by the authors emphasize enhancing the training process, focusing on improving specific modules and optimization techniques aimed at enhancing object detection accuracy without increasing inference costs [14]. These specialized modules and optimization techniques are referred to as trainable bag-of-freebies, promising significant performance for real-time applications without compromising accuracy.

2.4 Evaluation metrics for object detection

Ideally, the metrics for evaluating object detection models should reflect the best model based on the intended application.

2.4.1 Outputs in object detection

In the domain of object detection using YOLO models, performance evaluation relies on several key metrics. A True Positive (TP) indicates that the model correctly predicts and localizes an object, aligning with the ground truth. Conversely, a False Positive (FP) signifies that the model incorrectly predicts an object's presence that is not part of the ground truth. A False Negative (FN) denotes the model's failure to predict an object present in the ground truth annotation. These metrics are essential for assessing the model's accuracy in detecting objects in images. Ground Truths (GT) is fundamental to this evaluation, representing the real objects in the images, and it is established during the image labeling process in the dataset. Throughout training and testing, the object detection model's predictions are compared to the ground truth to assess its accuracy and ability to identify objects correctly [35].

2.4.2 Intersection over Union (IoU)

Intersection over Union (IoU) is a metric commonly used in computer vision to evaluate the performance of object detection algorithms, particularly in tasks like image segmentation and object localization. It measures the overlap between the predicted bounding box and the ground truth bounding box of an object.

IoU is calculated by taking the ratio of the area of intersection between the predicted and ground truth bounding boxes to the area of their union. The formula for calculating theIoU is shown in Equation 1.

$$IoU = \frac{\text{Area of Intersection}}{\text{Area of Union}} \tag{1}$$

The result of the IoU calculation is a value between 0 and 1. According to [36], typically, a predefined IoU threshold with a value of 0.5 is used to classify whether the prediction is a true positive or a false positive.

2.4.3 Precision

Precision measures the proportion of true positives predicted correctly. It can be obtained using Equation 2. Another way to understand precision is that it is associated with instances predicted as positive and how many of these instances are actually positive.

$$P = \frac{\text{True Positive}}{\text{True Positive + False Positive}} \tag{2}$$

Precision is important in situations where false positives are critical, meaning it is desired to be certain that instances predicted as positive are indeed positive.

2.4.4 Recall

Recall measures the proportion of true positives that are correctly predicted. It can be obtained using Equation 3. Intuitively, recall is related to the true positive instances that are correctly predicted by the model and indicates how many of these instances the model managed to predict correctly.

$$R = \frac{\text{True Positive}}{\text{True Positive + False Negative}} \tag{3}$$

Recall is crucial in situations where false negatives are more critical, such as in disease detection systems, where missing a positive case is more serious than a false positive.

2.4.5 Precision-Recall Curve

Calculating precision and recall for different confidence thresholds allows you to construct the precision-recall curve. The area under this curve is known as the Average Precision (AP), which is a common metric used to evaluate the performance of classification algorithms, especially in information retrieval and object detection tasks [17]. AP summarizes the trade-off between precision and recall across different confidence thresholds.

2.4.6 mean Average Precision

The mean Average Precision (mAP) is a widely used metric in computer vision, especially in tasks such as object detection and image segmentation. It is an extension of the AP metric and is employed to evaluate the overall performance of an object detection model across multiple classes [3].

Let the AP be the area under the precision-recall curve, and N be the number of classes. The mAP can be calculated using the Equation 4.

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{4}$$

2.4.7 Average Recall

The Average Recall (AR) metric evaluates the ability of an object detection model to recognize and retrieve objects at various Intersection over Union (IoU) thresholds. It quantifies the model's performance by measuring the count of true positive detections achieved across a range of overlap levels between predicted bounding boxes and reference bounding boxes (GT).

3 Related Work

3.1 Detecting Customer Induced Damages in Motherboards with Deep Neural Networks

In the work of [1], the authors develop a computer vision model that, through deep learning methods, identifies and classifies damage caused by the customer on the surface and motherboard components of electronic devices. The authors carry out the applicability of concepts covered in their work, such as deep neural networks, network architectures *Mask R-CNN* [37], *Soft Teacher* [38], and Swin Transformer [39] as the backbone of Mask R-CNN network for applications in damage caused by customers.

Three experiments were conducted, considering each of the deep neural network models Mask R-CNN, Soft Teacher, and Swin, using as performance metrics for predicting the damage region the Intersection over Union (IoU) together with Average Precision (AP) and Average Recall (AR) and verified that the Swin model obtained better results than other architectures, being featured by Swin-S with AP of 42.4%, AP@0.5 [1] of 80.2% and AP@0.75 of 39.8%.

3.2 Defect Detection in Printed Circuit Boards Using You-Only-Look-Once Convolutional Neural Networks

In the study conducted by [2] work, the authors use a deep learning algorithm based on Tiny-YOLOv2. In this study, the authors developed a classifier from CNN that recognizes various electrical components in a PCB and subsequently locates and detects defects in the PCB components.

To assess the model's performance, the authors employed 5-fold cross-validation, where the data was randomly divided into five equal segments, with four of these segments used for training and the remaining one for testing. The procedure was performed five times on the training and testing data set. The model accuracy was 98.79% for batch size 32, and the precision was constantly 0.99.

3.3 A Real-time PCB Defect Detector Based on Supervised and Semi-supervised Learning

Unlike the work presented in Sections 3.1 and 3.2, the work of [40] jointly uses supervised learning through deep neural networks and semi-supervised learning. According to the authors, a challenge faced by deep neural networks is the large amount of data required for training; the data must be labeled, and the labeling process is costly for large amounts of data. In this way, semi-supervised learning is proposed as a solution for labeling large sets of unlabeled data.

The medium weight consistency [41] method was used as a pillar of semi-supervised learning to label unlabeled data. For object detection, a new deep module was proposed that efficiently combines features at different resolutions and makes predictions to detect PCB defects at various scales. The model achieved results of 98.6% mAP @ 62 FPS for the DeepPCB dataset.

[1] The AP@'threshold' in object detection is a metric that represents the average precision (AP) calculated at a specific IoU (Intersection over Union) value of 'threshold'%.

3.4 Comparison of related work

Table 1 shows the main characteristics of each approach on the related work and our approach. The term FLOPS corresponds to the number of floating point operations per second, being a measure of a computer's computational capacity. FPS (frames per second) refers to the number of frames processed or displayed per second. The latency and FPS present in the table were obtained through the original articles of the related work; therefore, the hardware used to capture these results is not necessarily comparable as they are different.

Table 1: Comparative table of related work.

Work	Used Architecture	FLOPS (G)	Latency (ms)	FPS	Metric
1	Swin-S	359	-	437	mAP
2	Tiny-YOLO-v2	5.41	-	244	Accuracy
40	VGG16-tiny	15.3	128.62	-	mAP
Proposed Work	YOLOv6-N6	48.9	3.56	281	mAP

4 Materials and Methods

4.1 Metrics

The criteria for selecting the deep neural network architecture in this work are as follows: i) The primary evaluation metric is the mAP, offering a comprehensive assessment of the model's performance in object detection tasks. ii) Average Recall is also included as part of the criteria because it is a metric that measures the system's ability to correctly identify all instances of a target object belonging to a class in the dataset. iii) We set a limit of 200G FLOPs for the computational cost of the models trained during our experiments to maintain consistency between model complexity and the hardware limitations of mobile devices.

4.2 Methods

In Section 2.3.2, we introduced methods based on the YOLO architecture, which is considered state-of-the-art for real-time object detection in the COCO dataset [2]. The neural network architectures used in the experiments were selected based on their performance on the COCO dataset and the results obtained on the metrics introduced in Section 4.1.

The models chosen for the experiments were: i) YOLOv3-MobileNetV2, which utilizes MobileNetV2 as the backbone of the neural network; ii) YOLOv6, featuring versions optimized for mobile devices; iii) YOLOv7, promising good performance in real-time applications; and iv) YOLOv8 [42], chosen due to its recent popularity.

The four models with different YOLO-based architectures were trained and validated using subsets of our dataset. Sixteen experiments were conducted with different model architectures: one for YOLOv3-MobileNetV2, nine for YOLOv6 (Lite-L variations, N, S, M, L, N6, and S6), two for YOLOv7 (YOLOv7 and YOLOv7-X), and four for YOLOv8 (N, S, M, and L variations). It is important to note that YOLOv6 comprises nine experiments, conducted with Lite-L variations using different image resolutions (320x320, 640x640, and 1280x1280).

Each model was trained until convergence. Convergence in the training process occurs when the training error reaches an acceptable level, indicating that the model has satisfactorily learned to represent patterns in the training data.

4.3 Dataset

The images used as a database come from repair centers of a private sector electronic device manufacturer. Photo capture of images was carried out between July 1, 2022, and January 26, 2023, containing approximately 82 thousand unlabeled images, which make up a repository of raw images. All images have at least one CID according to the technicians who analyzed the PCBs and captured the images.

6383 images were selected for data labeling using a random selection method. The images were labeled using the Label Studio[3] tool. Of the images selected for labeling, 4223 images were discarded because they presented some image quality problems, such as lighting, low image quality, and minor damage. Therefore, the image discard rate was 65.86%.

The labeled dataset consists of 2179 images, with 514 images for validation, 1468 images for training, and 197 images for testing, totaling 9% for testing, 67.4% for training, and 23.6% for validation. Figure 6 shows the representation of the classes of CIDs present in the labeled dataset. The distribution of images per CID class in the training and validation sets is shown in Table 2.

All labeling was carried out by a specialist who marked and classified the CIDs in the images. The labeling process is carried out by identifying the damage followed by delimiting the damaged region according to the damage class. Our dataset has a total of 2179 images with 2529 labeled damages. Table 3 shows the distribution of annotations by class in the dataset.

[2] Available at: https://cocodataset.org/
[3] Available at: https://labelstud.io/

Table 2: Distribution of images per classes for train, test, and validation datasets.

Class	Validation dataset	Train dataset	Test dataset
S, Scratch	188	557	72
P, Pin damage	81	225	71
D, Distortion/break	245	686	58
Total	**514**	**1468**	**197**

(a) Class D, distortion/break (b) Class S, scratch (c) Class P, pin damage

Figure 6: Representation of CID classes in the dataset.

Our dataset has unbalanced classes, which can be noticed when comparing the ”P” class with the other classes, as shown in Table 3. Each of the different YOLO series used during the experiments has integration with data augmentation modules, allowing the minority classes to be represented and balanced so that the bias that may occur due to majority classes is reduced.

Table 3: Distribution of dataset classes.

Class	Amount
S, Scratch	927
P, Pin damage	568
D, Distortion/break	1034
Total	**2529**

4.4 Fine-tunning

The model undergoes training using a meticulously labeled dataset derived from raw images, followed by a fine-tuning phase. Fine-tuning is conducted to obtain improved models through a selection of hyperparameters for model training. Our fine-tuning process takes into account the following training parameters: i) The batch size, representing the number of images processed simultaneously within the training algorithm. ii) The number of training epochs, where each epoch involves processing a number of batches equivalent to the entirety of the training set. iii) The image size is a critical parameter under consideration, contributing to the accuracy and adaptability of the fine-tuning process to the specific data at hand.

Within this fine-tuning phase, we delve deeper into the intricate relationship between model complexity, batch size, and image size in object detection. It becomes evident that as model complexity increases, there is a corresponding escalation in the demand for computational resources. We carefully explore the impact of varying batch sizes and their potential efficiency drawbacks, notably the risk of exceeding the memory capacity of the training machine’s GPU. Furthermore, we analyze the effects of adjusting image dimensions on accuracy, acknowledging the trade-off between improved performance and heightened computational requirements. Thus, our methodology underscores the importance of striking a delicate balance among these factors to achieve optimal performance during the fine-tuning process.

4.5 Deploy on mobile devices

The steps required for converting the model for deployment on mobile devices are as follows: i) Serialization and optimization of the PyTorch model is the initial step, eliminating the Python[4] programming language dependency using the TorchScript module[5]. ii) Subsequently, the model is adapted to be compatible with the ncnn framework[6], an optimized high-performance neural network inference computing framework tailored for mobile platforms, notable for its lack of third-party dependencies. iii) Finally, an application for Android[7] devices, developed in the Kotlin[8] language, leverages the trained model, converted to the ncnn framework through the OpenCV-mobile[9] development module, for the purpose of identifying consumer-induced damages in printed circuit boards PCB via the device's camera.

4.6 Experimental Procedure

A virtual machine on the Google Colab[10] platform was used during all experiments carried out in this work. The specifications of the experiment machine are described in Table 4. The experiments were conducted with Python version 3.10.12, PyTorch 2.0.1, and NVIDIA CUDA[11] 11.8.

Table 4: Experiment environment specifications.

OS	Linux 5.15.120+ #1 SMP x86_64 GNU-Linux
CPU	Intel(R) Xeon(R) CPU @ 2.20GHz
Mem. RAM	51 GB
GPU	NVIDIA TESLA V100 (16 GB)

In our experiments, we employed a diverse set of hyperparameters for fine-tuning across different YOLO models. For YOLOv3-MobileNetv2, we utilized a batch size of 16, 320x320 image dimensions, and 50 training epochs. For YOLOv6, we explored various model complexities, including Lite-L, N, S, M, L, N6, and S6, along with image sizes of 320x320, 640x640, and 1280x1280, batch sizes of 8, 16, and 32, and the number of training epochs ranging from 100 to 400. For YOLOv7, we conducted experiments for YOLOv7 and YOLOv7-X model complexities, a batch size of 16, an image size of 640x640, and a fixed number of 100 training epochs. Finally, for YOLOv8, we conducted experiments using the N, S, M, and L model variations, with a batch size of 16 and an image size of 640x640. These parameter combinations were selected to investigate the performance and versatility of the YOLO models across a wide spectrum of settings.

We selected the best model obtained in our experiments for testing on mobile devices to observe the model's behavior during real-time inference.

5 Results and Discussion

Table 5 shows the results of our experiments. In a general analysis, it can be observed that YOLOv6 achieved better results than the other evaluated architectures. The YOLOv6-N6 is highlighted for being the model that presents the best results in terms of overall balance, considering not only mAP and AR metrics but also in terms of processing costs associated with FLOPs and the number of parameters.

Analyzing the results of Average Precision (AP) and their IoU threshold variations, we can observe that the best results were achieved by YOLOv6-S6 with an AP @0.5 of 33.8% and an AP @0.5:0.95 of 12.9%. The YOLOv6-N6 achieves results very close to those obtained with YOLOv6-S6, with an AP @0.5 of 33.1% and an AP @0.5:0.95 of 12.4%. In addition to surpassing YOLOv6-S6 in Average Recall (AR), another highlighted aspect for YOLOv6-N6 is its computational cost, which is 3.98 times lower in FLOPs and 4 times smaller in terms of parameters compared to YOLOv6-S6. This makes it the model with the best balance and is considered our top-performing model. From the perspective of Average Recall (AR), the best result was obtained by YOLOv6-Lite-L with an image resolution of 1280x1280, reaching an AR @0.5:0.95 of 37.4%, surpassing YOLOv6-N6 and YOLOv6-S6, which achieved an AR @0.5:0.95 of 34.7% and 33.9%, respectively.

The family of models that achieved the best results was the YOLOv6 family, followed by the YOLOv8 family. The family that presented the worst results was the YOLOv7 family, performing even worse than YOLOv3-MobileNetv2, which was the sole representative of the YOLOv3 architecture family.

The YOLOv6 and YOLOv3 model families include models with neural network architectures optimized for mobile devices. These models are YOLOv3-MobileNetv2 and YOLOv6-Lite, designed to operate efficiently on mobile hardware. In terms of FLOPs, YOLOv3-MobileNetv2 is the model with the lowest computational cost. However, its AP and AR results are compromised due to the focus on achieving high computational performance and rank among the three worst outcomes in our

[4]Available at: https://www.python.org/
[5]Available at: https://pytorch.org/docs/stable/jit.html
[6]Available at: https://github.com/Tencent/ncnn
[7]Available at: https://www.android.com/
[8]Available at: https://kotlinlang.org/
[9]Available at: https://github.com/nihui/opencv-mobile
[10]Available at: https://colab.research.google.com/
[11]Available at: https://developer.nvidia.com/cuda-toolkit

Table 5: Experiment Results.

Model	Image Resolution	mAP@0.50	mAP@0.50:0.95	(AR) @[IoU=0.50:0.95]	FLOPs	Parameters
YOLOv3-MobilenetV2	320x320	0.055	0.014	0.065	0.32G	3.50M
YOLOv6 Lite-L	320x320	0.117	0.043	0.216	3.31G	1.06M
YOLOv6 Lite-L	640x640	0.214	0.076	0.291	3.31G	1.06M
YOLOv6-N	640x640	0.208	0.080	0.275	11.34G	4.63M
YOLOv6-S	640x640	0.294	0.099	0.264	45.17G	18.50M
YOLOv6-M	640x640	0.307	0.119	0.253	85.63G	34.80M
YOLOv6-L	640x640	0.331	0.129	0.285	150.50G	59.54M
YOLOv6 Lite-L	1280x1280	0.274	0.109	0.374	13.25G	1.06M
YOLOv6-N6	**1280x1280**	**0.331**	**0.124**	**0.347**	**49.57G**	**10.34M**
YOLOv6-S6	1280x1280	0.338	0.129	0.339	197.52G	41.32M
YOLOv7	640x640	0.03	0.011	0.044	103.2G	36.49M
YOLOv7-X	640x640	0.016	0.005	0.025	188G	70.79M
YOLOv8-N	640x640	0.211	0.076	0.255	8.1B	3.2M
YOLOv8-S	640x640	0.269	0.090	0.279	28.4G	11.12M
YOLOv8-M	640x640	0.237	0.088	0.274	78.7G	25.84M
YOLOv8-L	640x640	0.269	0.089	0.326	164.8G	43.60M

experiments. Among the models optimized for low computational cost using default settings and with better AP and AR metrics is YOLOv6-Lite-L with an image resolution of 320x320 trained for 400 epochs, achieving only 3.31G FLOPs and 1.06 million parameters, with an AP@0.5 of 11.7% and an AR of 21.6%. There is also an experiment with YOLOv6-Lite and an image resolution of 1280x1280 that obtained better results than the best results achieved with architectures optimized for mobile devices, with an AP@0.5 of 27.4% and an AR of 37.4%. However, the computational cost in terms of FLOPs is 13.25G, which is higher than that of YOLOv6-N, which does not have specific optimizations for mobile devices. Table 6 shows the hyperparameters used to achieve the best results for each of the different YOLO architectures used.

Table 6: Hyperparameters of the networks.

Model	optimizer	lr	momentum	batch size	image size
YOLOv3-MobilenetV2	SGD	0.003	0.9	16	320x320
YOLOv6 Lite-L	SGD	0.01	0.9	8	1280x1280
YOLOv6-N	SGD	0.012	0.843	32	640x640
YOLOv6-S	SGD	0.012	0.843	32	640x640
YOLOv6-M	SGD	0.012	0.843	16	640x640
YOLOv6-L	SGD	0.012	0.843	16	640x640
YOLOv6-N6	SGD	0.01	0.937	32	1280x1280
YOLOv6-S6	SGD	0.01	0.937	16	1280x1280
YOLOv7	Adam	0.1	0.937	16	640x640
YOLOv7-X	Adam	0.1	0.937	16	640x640
YOLOv8-N	AdamW	0.1	0.937	16	640x640
YOLOv8-S	AdamW	0.1	0.937	16	640x640
YOLOv8-M	AdamW	0.1	0.937	16	640x640

One particularity among the YOLO families trained during our experiments is that the YOLOv6 family can calculate the Average Precision (AP) and Average Recall (AR) during the validation process for different object sizes (large, medium, small). This allows us to determine which model performed better for these different object sizes. Table 7 shows the AP and AR obtained by the YOLOv6 models for different object sizes.

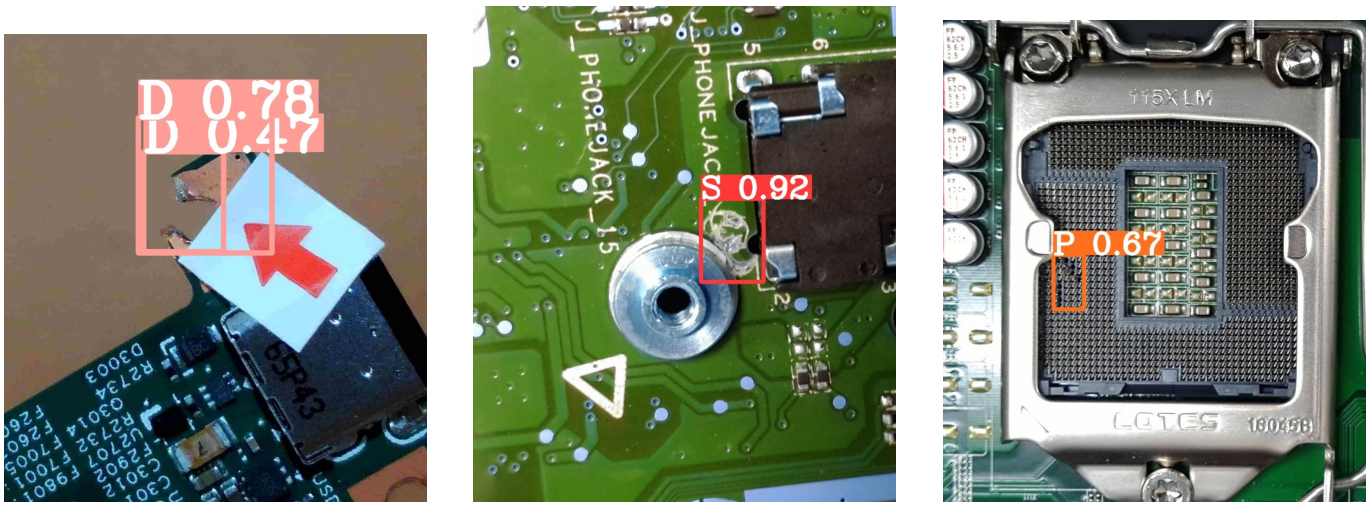
The results obtained for AP for objects of different sizes allow us to observe that all models have difficulties in detecting small objects, with the best of them being YOLOv6-M with an AP of less than 9%. For medium-sized objects, YOLOv6-N6 achieved the best results with 12.2%. For large objects, YOLOv6-L and YOLOv6-M performed the best with an AP of 16.9%. The worst result was from YOLOv6-Lite-L (320x320), obtaining the worst AP for each of the different sizes.

Evaluating the AR for objects of different detection sizes, we can see that for small objects, the best model was YOLOv6-S6, achieving 7.8%. The best model for medium-sized objects was YOLOv6-Lite-L (1280x1280) with 33.1%. In the case of large objects, once again, the model with the best results was YOLOv6-Lite-L (1280x1280) with 43.6%.

Figure 7 shows the inference of the YOLOv6-N6 model on a sample of images from each class of the dataset. We can observe that the model performs well on these images, achieving a confidence level of 92% for class S, Figure 7b; 67% for class P, Figure 7c; and two predictions in the same image for class D with confidence levels of 47% and 78%, Figure 7a. For the

Table 7: Results for different object sizes (YOLOv6).

Model	AP^s	AP^m	AP^l	AR^s	AR^m	AR^l
YOLOv6-Lite-L (320x320)	0.000	0.038	0.053	0.000	0.203	0.238
YOLOv6-Lite-L (640x640)	0.000	0.061	0.106	0.000	0.219	0.392
YOLOv6-N	0.000	0.053	0.127	0.000	0.188	0.395
YOLOv6-S	0.005	0.095	0.136	0.011	0.247	0.330
YOLOv6-M	**0.009**	0.100	**0.169**	0.016	0.232	0.313
YOLOv6-L	0.000	0.116	**0.169**	0.002	0.268	0.344
YOLOv6-Lite-L (1280x1280)	0.000	0.094	0.135	0.036	**0.331**	**0.436**
YOLOv6-N6	0.004	**0.122**	0.150	0.024	0.307	0.416
YOLOv6-S6	0.008	0.111	0.160	**0.078**	0.304	0.398



(a) Inference on a sample of class D (b) Inference on a sample of class S (c) Inference on a sample of class P

Figure 7: Inference on a sample of the dataset.

inference on these images, a confidence threshold of 0.4 and an IoU threshold of 0.45 were used, which are standard values for image inference in the YOLOv6 architecture.

5.1 Mobile Application

The application does not depend on third-party libraries, such as internet infrastructure, APIs, or cloud servers, and it should run directly on the mobile device where the application is installed.

At the top of the application interface, there are two selection buttons: one to choose the model to be used and another to switch between the device's CPU and GPU. The rest of the application interface is utilized by the rear camera to capture images and perform real-time inferences. The application also allows checking the number of frames per second (FPS) at which the model makes inferences on the images captured by the camera, enabling the user to assess the image processing performance for each of the available models.

The best family of YOLO models from our experiments was the YOLOv6 family; therefore, these models were chosen to be validated in our application. In general, the models can identify the CIDs on PCBs from a sample of images that were not shown to the models during training.

The models were tested on three mobile devices with different hardware specifications. The hardware specifications of the tested devices can be seen in Table 8.

Table 8: Mobile Devices.

Device ID	CPU	GPU	RAM
1	Helio P35 MediaTek MT6765	PowerVR GE8320	4 GB
2	Exynos 1330	Mali-G68 MP2	4 GB
3	Exynos 1380	Mali-G68 MP5	8 GB

The performance of the models in terms of image processing during inference on each of the devices varies depending on which model is currently being executed, with models having lower computational costs in FLOPs achieving higher frames per

second than devices with higher computational costs.

The YOLOv6 (N, S, M, and L) models encountered memory access issues during the deployment process on mobile devices, and as a result, they were excluded from testing on real mobile devices. The remaining models were tested, and we recorded the average FPS during the execution of these models on different devices. The average FPS was captured by displaying the smartphone camera feed for five minutes using a sample of images that were not used during the model training. Both the GPU and CPU of the devices were used for performance comparisons. Table 9 shows the average FPS with the models running on the CPU and GPU for each of the evaluated models on our three test devices.

Table 9: Average FPS on mobile devices.

Device ID	Model	FPS (GPU)	FPS (CPU)
1	YOLOv6-Lite-L (320x320)	6.7	12
1	YOLOv6-Lite-L (640x640)	2.6	4.5
1	YOLOv6-Lite-L (1280x1280)	0.76	1.5
1	YOLOv6-N6	0.58	1.2
1	YOLOv6-S6	0.15	0.6
2	YOLOv6-Lite-L (320x320)	16	30
2	YOLOv6-Lite-L (640x640)	6.5	19.5
2	YOLOv6-Lite-L (1280x1280)	1.6	5
2	YOLOv6-N6	1.4	4
2	YOLOv6-S6	0.8	1.2
3	YOLOv6-Lite-L 320x320	20.3	30.8
3	YOLOv6-Lite-L 640x640	12.3	28.3
3	YOLOv6-Lite-L 1280x1280	2.7	6.6
3	**YOLOv6-N6**	**2.6**	**5.7**
3	YOLOv6-S6	1	2.1

In general, the models performed better when utilizing the CPU resources of the mobile devices during the performance tests. The application's performance when using the GPU of the devices was considerably worse for all smartphones, with performance issues occurring in our tests on the more limited hardware of Devices 1 and 2.

Device 1, which is a 2020 input device, was used to assess the application's performance on older hardware. The YOLOv6-Lite-L (1280x1280), YOLOv6-N6, and YOLOv6-S6 models experienced occasional crashes when run on the GPU due to a low frame rate. When using only the CPU, YOLOv6-S6 exhibited performance issues, showing slowness in performing inference on images. Device 2, released in 2023, showed good performance with the tested models on our mobile devices, with no stutters or freezes even when using the CPU. During GPU tests, slight stutters were noticed when using the YOLOv6-S6 model due to the low FPS achieved with this model. Device 3, with the best hardware specifications, experienced no performance issues when running the models on either the CPU or GPU of the smartphone.

The model that achieved the best FPS was the YOLOv6-Lite-L (320x320) with an average of 20.3 FPS on the GPU and 30.8 FPS on the CPU. Models with higher resolutions showed lower FPS averages, with the image resolution of the trained models being a significant factor affecting model complexity. This is evident when comparing YOLOv6-Lite-L models trained with different image resolutions.

Our best model for detecting CIDs on PCBs is the YOLOv6-N6, which achieved an average of 2.6 FPS when using the GPU and 5.7 FPS when using the CPU, in addition to mAP and AR results comparable to YOLOv6-S6 but with lower computational cost. It is the ideal model for running on limited hardware devices.

During the execution of the models on different mobile devices, we can observe that if the model is run at a frame rate higher than 1 FPS, the user experience is satisfactory, without the presence of noticeable slowness or small freezes during inference on the images captured by the device. Despite this, running the models with higher FPS brings greater comfort, with greater fluidity noticed when capturing images.

6 Conclusion

This work presented a proposal for an object detection model designed to detect Customer-Induced Damages (CID) on Printed Circuit Boards (PCB) directly on Android devices through a mobile application. A series of experiments were conducted with various model architectures derived from YOLO (YOLOv3-MobileNetV2, YOLOv6, YOLOv7, and YOLOv8) to select the best model within the context of our data. The YOLOv6-N6 neural network architecture achieved the best results with a mAP@0.5 of 33.1%, an AR of 34.7%, and an average FPS of 5.7 when running on the CPU of the best available mobile device during testing. It showed very close AP and AR results to the most complex evaluated model (YOLOv6-S6) with a computational cost of 3.98 times lower.

The dataset on which the experiments were conducted is derived from repair centers. It consists of images of printed circuit boards with CIDs in various lighting conditions, image orientations, and positions, which posed challenges during data labeling and model training.

In the proposed methodology, we contribute with three points of innovation: we conducted a study on low-computational-cost object detectors; a series of experiments with YOLO object detector architectures were carried out to identify CIDs on PCBs with limited computational cost; the trained models were deployed on real mobile devices, where the performance in detecting CIDs on PCBs was verified.

For future work, we plan to highlight the inclusion of new classes of CIDs in our dataset and label more data to provide better quality and diversity of data to the trained models. The addition of new data to the dataset may be directly associated with the use of Active Learning for data labeling. Additionally, incorporating MLOps for versioning, monitoring, and continuous model training is essential to provide improved production models linked to user feedback.

7 Acknowledgement

This research was partially funded by Lenovo, as part of its R&D investment under Brazilian Informatics Law.

REFERENCES

[1] D. Alves, V. Farias, I. Chaves, R. Chao, J. P. Madeiro, J. P. Gomes and J. Machado. "Detecting Customer Induced Damages in Motherboards with Deep Neural Networks". In *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2022.

[2] V. A. Adibhatla, H.-C. Chih, C.-C. Hsu, J. Cheng, M. F. Abbod and J.-S. Shieh. "Defect detection in printed circuit boards using you-only-look-once convolutional neural networks". *Electronics*, vol. 9, no. 9, pp. 1547, 2020.

[3] A. Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O'Reilly Media, Inc.", 2022.

[4] X. Zhang, X. Zhou, M. Lin and J. Sun. "Shufflenet: An extremely efficient convolutional neural network for mobile devices". In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6848–6856, 2018.

[5] M. Tan, R. Pang and Q. V. Le. "Efficientdet: Scalable and efficient object detection". In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10781–10790, 2020.

[6] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". *arXiv preprint arXiv:1704.04861*, 2017.

[7] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen. "Mobilenetv2: Inverted residuals and linear bottlenecks". In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[8] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*. "Searching for mobilenetv3". In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1314–1324, 2019.

[9] J. Redmon, S. Divvala, R. Girshick and A. Farhadi. "You only look once: Unified, real-time object detection". In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[10] J. Redmon and A. Farhadi. "YOLO9000: better, faster, stronger". In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, 2017.

[11] J. Redmon and A. Farhadi. "Yolov3: An incremental improvement". *arXiv preprint arXiv:1804.02767*, 2018.

[12] A. Bochkovskiy, C.-Y. Wang and H.-Y. M. Liao. "Yolov4: Optimal speed and accuracy of object detection". *arXiv preprint arXiv:2004.10934*, 2020.

[13] C. Li, L. Li, Y. Geng, H. Jiang, M. Cheng, B. Zhang, Z. Ke, X. Xu and X. Chu. "Yolov6 v3. 0: A full-scale reloading". *arXiv preprint arXiv:2301.05586*, 2023.

[14] C.-Y. Wang, A. Bochkovskiy and H.-Y. M. Liao. "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors". In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7464–7475, 2023.

[15] L. Cabral, V. Farias, L. Sena, I. Chaves, J. P. Pordeus, J. P. Santiago, D. Sá, J. Machado and J. P. Madeiro. "An Active Learning Approach for Detecting Customer Induced Damages in Motherboards with Deep Neural Networks". *Learning & Nonlinear Models*, vol. 21, no. 2, pp. 29–42, 2023.

[16] M. A. Alghassab. "Defect detection in printed circuit boards with pre-trained feature extraction methodology with convolution neural networks". *Computers, Materials & Continua*, vol. 70, no. 1, pp. 637–652, 2022.

[17] R. Szeliski. *Computer vision*. Springer Nature, 2022.

[18] R. C. Gonzalez and R. E. Woods. *Processamento Digital de Imagens*. Pearson, 2009.

[19] L. Chen, S. Li, Q. Bai, J. Yang, S. Jiang and Y. Miao. "Review of image classification algorithms based on convolutional neural networks". *Remote Sensing*, vol. 13, no. 22, pp. 4712, 2021.

[20] H. Yu, Z. Yang, L. Tan, Y. Wang, W. Sun, M. Sun and Y. Tang. "Methods and datasets on semantic segmentation: A review". *Neurocomputing*, vol. 304, pp. 82–103, 2018.

[21] A. M. Hafiz and G. M. Bhat. "A survey on instance segmentation: state of the art". *International journal of multimedia information retrieval*, vol. 9, no. 3, pp. 171–189, 2020.

[22] K. He, G. Gkioxari, P. DollÃ¡r and R. Girshick. "Mask R-CNN", 2018.

[23] X. Wu, D. Sahoo and S. C. Hoi. "Recent advances in deep learning for object detection". *Neurocomputing*, vol. 396, pp. 39–64, 2020.

[24] Z. Zou, K. Chen, Z. Shi, Y. Guo and J. Ye. "Object detection in 20 years: A survey". *Proceedings of the IEEE*, vol. 111, no. 3, pp. 257–276, 2023.

[25] Y. Xiao, Z. Tian, J. Yu, Y. Zhang, S. Liu, S. Du and X. Lan. "A review of object detection based on deep learning". *Multimedia Tools and Applications*, vol. 79, pp. 23729–23791, 2020.

[26] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar and B. Lee. "A survey of modern deep learning based object detection models". *Digital Signal Processing*, vol. 126, pp. 103514, 2022.

[27] I. Goodfellow, Y. Bengio and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[28] A. R. Pathak, B. Agarwal, M. Pandey and S. Rautaray. "Application of deep learning approaches for sentiment analysis". *Deep learning-based approaches for sentiment analysis*, pp. 1–31, 2020.

[29] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie and L. Farhan. "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions". *Journal of big Data*, vol. 8, pp. 1–74, 2021.

[30] C. Shorten and T. M. Khoshgoftaar. "A survey on image data augmentation for deep learning". *Journal of big data*, vol. 6, no. 1, pp. 1–48, 2019.

[31] E. Strelcenia and S. Prakoonwit. "A survey on gan techniques for data augmentation to address the imbalanced data issues in credit card fraud detection". *Machine Learning and Knowledge Extraction*, vol. 5, no. 1, pp. 304–329, 2023.

[32] F. Chollet. *Deep learning with Python*. Simon and Schuster, 2021.

[33] C. Li, L. Li, H. Jiang, K. Weng, Y. Geng, L. Li, Z. Ke, Q. Li, M. Cheng, W. Nie *et al.*. "YOLOv6: A single-stage object detection framework for industrial applications". *arXiv preprint arXiv:2209.02976*, 2022.

[34] S. Han, H. Mao and W. J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding". *arXiv preprint arXiv:1510.00149*, 2015.

[35] D. Shah. "Mean Average Precision (mAP) Explained: Everything You Need to Know". Available at: `https://www.v7labs.com/blog/mean-average-precision`. Accessed on: Jun 08, 2023, 2022.

[36] J. Hui. "mAP (mean Average Precision) for object detection". Available at: `https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173`. Accessed on: Jun 09, 2023., 2018.

[37] K. He, G. Gkioxari, P. Dollár and R. Girshick. "Mask r-cnn". In *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

[38] M. Xu, Z. Zhang, H. Hu, J. Wang, L. Wang, F. Wei, X. Bai and Z. Liu. "End-to-end semi-supervised object detection with soft teacher". In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3060–3069, 2021.

[39] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin and B. Guo. "Swin transformer: Hierarchical vision transformer using shifted windows". In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 10012–10022, 2021.

[40] F. He, S. Tang, S. Mehrkanoon, X. Huang and J. Yang. "A Real-time PCB Defect Detector Based on Supervised and Semi-supervised Learning." In *ESANN*, pp. 527–532, 2020.

[41] A. Tarvainen and H. Valpola. "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results". *Advances in neural information processing systems*, vol. 30, 2017.

[42] G. Jocher, A. Chaurasia and J. Qiu. "Ultralytics YOLOv8". Available at: `https://github.com/ultralytics/ultralytics`. Accessed on: Oct 15, 2023., 2023.