TÉCNICO LISBOA

Deep Learning and Soft Tasking

Towards Respbots (Responsive Collaborative Robotics)

João Pedro Neves da Silva

Thesis to obtain the Master of Science Degree in

Mechanical Engineering

Supervisor: Prof. Mário António da Silva Neves Ramalho

Examination Committee

Chairperson: Prof. Duarte Pedro Mata de Oliveira Valério
Supervisor: Prof. Mário António da Silva Neves Ramalho
Members of the Committee: Prof. João Carlos Prata dos Reis
Prof. Susana Margarida da Silva Vieira

January 2021

To mom...

Acknowledgments

First, I would like to thank my supervisor, Professor Mário Ramalho, for the opportunity to write this dissertation and to learn on such an interesting topic.

A word of acknowledgment to Miguel Martins, whose help and opinion were invaluable to this work. Equally, a word of gratitude to Gabriel Nunes, for his help in the annotation of training examples.

For all my years in IST, my scientific knowledge has increase but, most importantly, I have grown as a person and as a citizen. For that, I have to thank all the organizations I have been a part of, but specially, all the people that were with me in those organizations. All the talks, all the arguments, all the physical work have contributed to who I am today. So, thanks to all the people that were with me in Fórum Mecânica, AEIST, CPMEMec and many, many more.

A special thanks to all my friends in Gouveia, Lisboa and around the world. From Farvão to the Travessa das Amoreiras a Arroios, you were essential to my well-being all these years.

Finally, but above all, thank you to all my family for being my support team and specially to my parents, my superheroes.

Resumo

Apesar do recente sucesso de algoritmos de última geração de detecção de objecto usando aprendizagem profunda, as aplicações práticas deste métodos e a existência de produtos que fazem uso desta tecnologia ainda é bastante limitada. Neste trabalho é avaliada a possibilidade de aplicar esta tecnologia a um robô de manipulação, que possa ser vendido ao público a baixo preço.

Para esse efeito, 2 cenários diferentes são usados, um referente a um ambiente doméstico e outro que pretende simular um laboratório de electrónica. Também são estudadas duas filosofias diferentes para os dados de treino. No primeiro caso, uma base de dados curada para o efeito é usada para recolher informação, enquanto no segundo caso a informação é recolhida através de pesquisas na internet e processada pelo autor. Ademais, vários métodos de detecção de objecto são testados (Faster R-CNN, YOLO, SSD) com o intuito de avaliar a viabilidade prática desta tecnologia, mas também para avaliar qual dos métodos seria o melhor para consumar este objectivo.

Assim sendo, este trabalho pode ser visto como uma prova de conceito do uso de algoritmos de detecção de objecto com aprendizagem profunda num robô de manipulação de pequenas dimensões.

Palavras-chave: Detecção de Objecto, Reconhecimento de Objecto, Aprendizagem Profunda em Visão Computacional, Robô de Manipulação.

Abstract

Despite the recent success of state-of-the-art deep learning algorithms in object detection, the practical applications of these methods and the availability of products which make use of this technology is still very limited. In this work it is evaluated the possibility of applying this technology to a table-top pick and place robot, which can be marketed at a low price to the general public.

To that end, two different scenarios are used, one referent to an household environment and another dedicated to simulating an electronics laboratory. It is also studied two different philosophies for the training data. In the first case a purposely curated dataset is used to collect the data, while on the second the data is collected from standard internet searches and processed by the author. Furthermore, several object detection methods (Faster R-CNN, YOLO, SSD) are tested in order to evaluate the practical feasibility of this technology but also to evaluate which method would be best in order to consummate this objective.

In light of that, this work can be seen as a proof of concept of using deep learning based object detection algorithms in a pick and place robot of small dimensions.

Keywords: Object Detection, Object Recognition, Deep Learning in Computer Vision, Pick and Place Robot.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

For the last century, robots have changed from science fiction to a reality in certain areas of our life. Since the coining of the term by the brothers Čapek [1], in the late 1910's, robots have greatly developed and are now present in all kinds of fields, such as industry, transportation or medicine. Their depiction in literature and cinematic fiction is extremely diverse and, in spite of a great evolution over the years, robots, especially those available to the public, are not comparable to those depictions and, in fact, are not able to emotionally connect with human. Even when they are incorporated on common daily life, it is in a very limited way, far from the humanoid general purposed robots depicted, for example, on the animated series "The Jetsons" or the cinematic franchise "Star Wars". However, these, and countless other, popular culture references show a widespread desire of integrating robots in menial and everyday tasks.

This desire has given origin to several wide fields of research, amongst which we can count Human-robot Collaboration (HRC) and Cobots. Bauer, Wollherr and Buss [2] defined HRC as "working with someone on something", that is, it implicates "a common goal", as opposed to Human-robot Interaction that functions as a "master-servant relationship" [3]. Cobots are defined as "a robotic device which manipulates objects in collaboration with a human operator" and "provides assistance to the human operator by setting up virtual surfaces which can be used to constrain and guide motion" [4]. In fact, both these fields have real applications, for instance, in healthcare [5], military applications [6], rescue [7], industrial production [8], domestic environment [9] and many more.

Companies and markets have also taken a keen interest in such devices, as they expect these to become a big profitable market. A 2018 report published by *MarketsandMarkets* [10] estimated that in 2025 the collaborative robot market will be worth more than $12.000 Million. This expectation led many companies to pursue the Cobots market and many major players have presented their solutions, such as, ABB (YuMi, 2015), KUKA (LWR3, 2004), Fanuc (CR-35iA, 2015), Omron (Lynx, 2017), Universal Robots (UR5, 2008; UR10, 2012; UR3, 2015), Epson (Flexion N-Series, 2016) and Festo (BionicCobot, 2010; OctopusGripper, 2017).

The interest from such companies is justified by the interest of end users and the population in general. A survey conducted by Khan [11] concluded that participants had, in general, positive attitudes and found usefulness in Intelligent Service Robots (ISR), that is, a mobile platform that can perform cleaning and transportation tasks in a domestic setting. Furthermore, it concluded that the preferential measures for such device are 30-50 cm in diameter and 100-150 cm in height. This was later confirmed by Wongphati, Matsuda, Osawa and Imai [12] for robotic arms, who also concluded that household tasks were preferred, followed by workplace tasks and desk or working surface tasks. This demonstrates that, despite the reality of this technology being far away from day-to-day use, there is an acceptability and, in fact, a demonstrated necessity in the general population for such kind of technology.

From this necessity, allied with the drop in price of most of the technologies needed, one can easily imagine the development in the near future of several devices that would complement humans into tasks more general, more abstract and less structured than the ones Cobots perform today. It is possible to imagine, for instance, that Cobots could be used in an household environment to help in mundane tasks such as cooking or cleaning a table. Similarly, they could be used in an office environment to perform comparable tasks. By contrast, they could also be used in more specialized environments such as a laboratory. There they could complement humans by recognizing components and either picking them up on cue or suggesting them to the user or, alternatively, alert for the dangerous use of those components. Independent of which task we are previewing, in order to achieve good performance, there is a set of operations of paramount importance: such a device must be capable of accurately depict the outside world. That is, any robot talked above must be capable of extracting the location of objects from an input sensory system, for instance a video feed, and be able to classify its nature, so that the object can be moved to a designated box, for example. This set of operations is the main focus of this work.

1.2 Topic Overview

Provided the circumstances, namely that there are several generic objects in an image and the main goals are to extract information related to its location as well as categorize such objects into classes, this clearly takes us to the computer vision task of object recognition. Vision is popularly defined as the process of discovering from images what is present in the world and where it is [13]. Andreopoulos and Tsotsos [14] divide such a problem into four levels of tasks, with each task incorporating the previous:

- *Detection*: is a particular item present in the stimulus?
- *Localization*: detection plus accurate location of item.
- *Recognition*: localization of all the items present in the stimulus.
- *Understanding*: recognition plus role of stimulus in the context of the scene.

Therefore, the authors state that the recognition problem denotes the more general problem of identifying all the objects present in the image and providing accurate location information of the respective objects.

Russakovsky at al. [15] use the term object recognition broadly to encompass both image classification, a task requiring an algorithm to determine what object classes are present in the image, as well as object detection, a task requiring an algorithm to localize, as in restrict to a particular place, all objects present in the image. In fact, in spite of being possible to find some differences between the tasks of object recognition and object detection, many authors use both terms interchangeably and the same will be true for this work.

This is visible, for example, when Szeliski [16] defines object detection as categorizing not just whole images but delineate (with bounding boxes) where various objects are located. This shows a great similarity with the previous definitions of object recognition, reason for which the terms are used interchangeably.

The tasks of identifying, locating and categorizing objects are among the earliest in the field of Computer Vision. In fact, one of the initial drives of the field was to artificially recreate this process, which comes effortlessly to virtually any animal and certainly to intelligent animals.

1.2.1 Historical Overwiew

Lawrence Roberts, commonly accepted as the father of Computer Vision, published in 1963 [17], one of the first efforts to process a photograph and build a computational model representative of the object from the information in that photograph, in what he called machine recognition of pictorial data. In his work, Roberts was able to build a 3-D model of an object from the pictorial data and produce 2-D projections of the model. He stated the biggest benefit of the investigation was ”an increased understanding of the possible processes of visual perception”. However, his final goal was to ”recognize the same similarity classes which humans do”, in what he called multiple object recognition.

In 1966, the Summer Vision Project [18] had the intention of solving most of the Computer Vision tasks researchers still struggle with today, during one single summer. The promoters expected the project to be ”a real landmark in the development of pattern recognition”. Furthermore, they stated that the primary goal of the project was to create programs that would be able to separate the foreground and background regions of an image and the final goal was ”Object Identification”, in other words, ”actually name objects by matching them with a vocabulary of known objects”.

Another important paper on the field was published in 1973 by Fischler and Elschlager [19]. Given a description of a visual object, they tried to find that object in a photograph. Their framework consisted of breaking down the object into a number of more primitive parts, specifying an allowable range of spatial relations between these primitive parts and then using an embedding metric to evaluate how well any composition of primitive picture pieces matched the desired composite picture.

Another field that has close ties with object detection is Optical Character Recognition. Both fields share several techniques and have advanced each other throughout the years. In 1980, when working on optical character readers, Fukushima [20] proposed the Neocognitron, a neural network model for pattern recognition. As stated by the author, this self organizing network had the capability of recognizing stimulus patterns and learning them without a teacher, being invariant to position or small distortions.

Efforts in the subsequent decades, all focused on how an artificial systems could perceive, identify or recognize the objects and patterns of the real world [13, 21, 22]. These proposed different models and different concepts in order to represent the world and achieve those tasks.

One of the main interests in object detection is the detection of humans and especially of human faces. One of the first successful application of object detection was inspired by the work of Paul Viola and Michael Jones [23], in what became known as the Viola-Jones Algorithm. Their work used Adaboost algorithm and Haar-like features to perform near real-time face detection. The industry rapidly took notice and, in 2006, Fujifilm announced the first digital camera with face recognition technology, using the Viola-Jones Algorithm.

This was part of a trend, popular in the 2000's, where methods would compute different types of feature representations, with varying degrees of invariance to illumination, scale, rotation, occlusion and other problems, and later feed those representations to a classifier. The works of Dalal and Triggs [24], Lazebnik et al. [25] and Felzenswalb [26] presented important approaches to the problem.

With the advancements in this field, the need for a standardized dataset to be used as benchmarking became clear. In 2005, the Pascal Visual Object Classes (Pascal VOC) project was launched [27]. It provided a standardized dataset and created an annual competition both for the classification and the detection task. The dataset was initially composed of 4 classes and 1578 images, although it was expanded in the subsequent years. In 2010, the ImageNet Large Scale Visual Recognition Competition (ILSVRC) started [15]. The dataset presented for the competition [28] was a major improvement from Pascal VOC 2010, scaling up from 20 object classes and around 20 000 images to 1000 classes and about 1.5 million images. This dataset has become one of the leading benchmarks in object classification and object detection. Another important dataset for the field was launched in 2014, Microsoft COCO [29]. The database presented an harder task because, rather than presenting images that were specially thought to adjust to object recognition, it presented objects in its naturally occurring contexts.

In the 2012 edition of the ImageNet challenge, the biggest revolution for the task of object recognition appeared. A team from the University of Toronto enter a convolutional neural network (CNN) model [30], and won the competition with a major error-rate drop from previous years [15]. Since 2012, all the winners have used CNN models and error-rates have steadily dropped to a very small percentage, being on par or even surpassing human capacities [15].

Hence, from 2012 onward, it has been common practice and, in fact, the leading strategy to tackle problems such as Image Classification and Object Detection to use some form of algorithm with a CNN backbone. For that reason, this will be the strategy used for this work.

1.3 Objectives

The general purpose of this thesis is to study the input sensory system of a device such as a small Cobot, that is, the video feed of a tabletop pick and place robot, and to be able to extract location and class information from such a collection of images. It is, therefore, the study of applying object detection algorithms to extract that information. Thus, the objectives of this work can be summarized as follows:

1. Implement object detection algorithms that can be used for the task at hand.
2. Compare the performance of three object recognition algorithms (YOLO, Faster R-CNN and SSD) on the task.
3. Evaluate the impact of different environments in the performance of the different algorithms. The intention is to use environments that simulate an opportunity of collaborative robotics.
4. Assess the benefits of different methods of data collection, one using a curated database and one generating a custom database.

1.4 Thesis Outline

This work is divided into 5 chapters. The present chapter establishes the reasons for this work, its motivation, along with defining the problem and providing some historical context for it. In Chapter 2 some important concepts for the development of this work are clarified. Next, Chapter 3 indicates the practical approach to the problem, that is, the resources used to solve the enunciated issue. The results of this approach are detailed in Chapter 4, while Chapter 5 is dedicated to summarising the overall quality of the solutions, inferring some hypotheses from the results and recommending some future work.

Chapter 2

Background

The next sections will be dedicated to providing some contextual knowledge required to object detection. First, some theoretical concepts regarding convolutional neural networks are enunciated. Then, some CNN architectures are focused. Next, we introduce object detection methods and some metrics designed to compare them. Finally, the methods used in this work are discussed in detail.

2.1 Convolutional Neural Networks

Convolutional Networks are defined by Goodfellow at al. [31] as a specialized kind of neural network for processing data that has a known, grid-like, topology and employ a mathematical operation called *convolution*, which is a specialized kind of linear operation. In other words, convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

Convolutional Neural Networks had been successfully used before the 2012 edition of the ImageNet challenge, mainly in pattern recognition with the works of LeCun et al. [32–34]. They had resurfaced around 2006 with the work of Hinton and Salakhutdinov in dimensionality reduction [35] and with the work of Mohamed et al. in speech recognition [36]. However, it was from 2012 onward they proved ideal for several computer vision tasks.

Being a feedforward neural network, its goal, stated by the same authors, is to approximate some function f^*. A feedforward network defines a mapping $y = f(x;\theta)$ and iteratively learns the value of the parameters θ, also known as weights, that result in the best approximation. On the other hand, there is another set of parameters that affect the training of a neural network and cannot be adapted by the learning algorithm itself, the hyperparameters[1]. These can be used to control the behaviour of the learning algorithm. Some hyperparameters relevant to the training of CNNs will be discussed later, because of their importance in the training process and their contributions to the convergence of the process.

These mappings are called networks because they are typically represented by composing together

[1] There is however a field of research dedicated to automated hyperparameter tuning, hyperparameter optimization.

many different functions. The model is associated with a directed acyclic graph describing how the functions are composed together. For example, we might have three functions $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$ connected in a chain, to form $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. These chain structures are the most commonly used structures of neural networks. In this case, $f^{(1)}$ is called the first layer of the network, $f^{(2)}$ is called the second layer, and so on. The final layer of a feedforward network is called the output layer. The overall length of the chain gives the depth of the model. As will be discussed later, the depth of a network is an important hyperparameter on its training and is the object of much scientific discussion [37–39]. Also important, when selecting the hyperparameters, is the width of the network, that is, the dimensionality of the hidden layers [37, 40, 41].

During neural network training, we drive $f(x)$ to match $f^*(x)$. The training examples specify directly what the output layer must do at each point x; it must produce a value that is close to y. The behavior of the other layers is not directly specified by the training data. It is the learning algorithm that must decide how to adjust the weights of these layers to best approximate f^* and, for that reason, these are called hidden layers.

Many authors describe the task of driving $f(x)$ to match $f^*(x)$ as an optimization problem [31, 42, 43], that is, either minimizing or maximizing some function $L(x)$, the loss function, also called the objective function, the cost function or the error function. These functions usually include two terms: the first, frequently called data loss, evaluates the performance of the network when compared to the ground-truth; the second term, the regularizer, is used to reduce the generalization error, that is, to improve the model's performance for examples outside the training data. The technique most often used to minimize the loss function is called Gradient Descent [44] or Method of Steepest Descent and is based on the observation that the gradient of a function points in the direction of maximum increase and, therefore, the negative gradient will point in the direction of maximum decrease. Thus, the gradient descent method applied to neural networks can be summarized as shown in Figure 2.1.

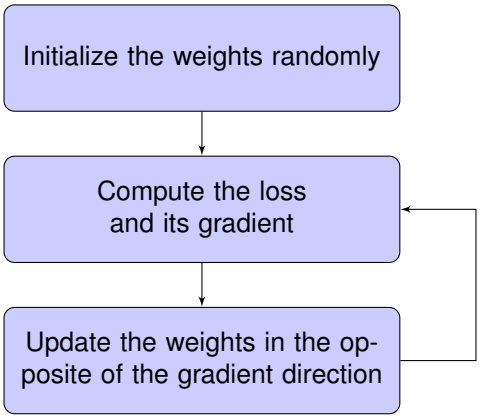


Figure 2.1: Simplified algorithm for the gradient descent method.

Knowing the direction of the gradient, it is also necessary to know the magnitude of the update, in other words, how big of a step do we take in that direction. This is maybe the most important hyperparameter when training a neural network and is called the learning rate [31].

However, most machine learning applications don't use the gradient descent method in its original

formulation. This happens because the full loss of a dataset is the average loss across the entire dataset. Such a computation isn't feasible when the number of examples in a dataset increases and a dataset used for computer vision applications can reach millions of examples. Hence, rather than using the entire dataset a small sample of training examples is sampled at each iteration, a mini batch, and an estimate of the full sum on that set is computed. This is called Stochastic Gradient Descent (SGD). Another technique used to improve the performance of SGD is Momentum [45], an accumulation of an exponentially decaying moving average of past gradients that promotes movement in their direction. Simplifying, the size of the step in the negative gradient direction, which was previously only dependent on the learning rate, now depends on how large and how aligned a sequence of gradients are. The step size is largest when many successive gradients point in exactly the same direction.

The convolution operation[2] is defined by Goodfellow at al. [31] as an operation on two functions of a real-value argument given by:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da \tag{2.1}$$

In convolutional network terminology, the first input, x, is referred to as the input and the second, the function w as the kernel or filters. The output is referred to as the feature map. In machine learning applications, the input is usually a multidimensional array of data and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm, both referred to as tensors. This function are, therefore, discretized and zero everywhere but the finite set of points for which we store the values. This means that in practice we can implement Equation 2.1 as a summation over a finite number of array elements. We also use convolutions over more than one axis at a time. For example, if we use a two-dimensional image I as our input, we probably also want to use a two-dimensional kernel K. Thus a convolution is given by:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.2}$$

As is depicted in Figure 2.2, a typical layer of a convolutional network consists of three stages. In the first stage, the layer performs several convolutions in parallel to produce a set of linear activations. In the second stage, each linear activation is run through a nonlinear function, an activation function. The most popular choices for activation function include the Logistic Sigmoid and Hyperbolic Tangent functions and more recently the Rectified Linear Unit (ReLU) function. In the third stage, we use a pooling function to modify the output of the layer further. A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs. For example, the max pooling operation reports the maximum output within a rectangular neighborhood. Other popular pooling functions include the average of a rectangular neighborhood, the L2 norm of a rectangular neighborhood, or a weighted average based on the distance from the central pixel. This operation is very relevant to translation and size invariance. The combination of these tools in different structures makes up the architecture of a

[2]The operation used in a convolutional neural network does not correspond precisely to the definition of convolution as used in other fields such as pure mathematics.

network.

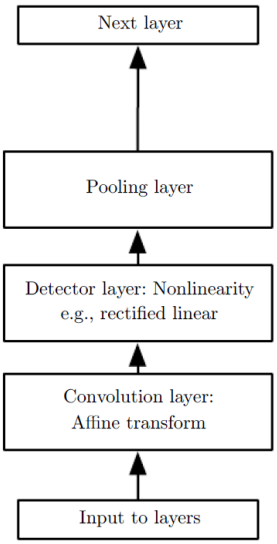


Figure 2.2: The components of a typical convolutional neural network layer (Goodfellow et al. 2016).

2.2 CNN Architectures

The 2012 winner of the ImageNet competition introduced a CNN architecture, later known as AlexNet [30], composed of 5 convolutional layers and 3 fully-connected layers, combined with some normalization and max-pooling layers. The use of dropout and data augmentation to avoid overfitting, the use of the ReLU function as activation function and the implementation of training on multiple GPU's were also determinant in the success of the architecture. Using stochastic gradient descent (SGD) with a batch size of 128 examples, momentum of 0,9 and weight decay of 0,0005, a 7 models ensemble and also a learning rate initiated at 0,01 and reduced by 10 every time the validation accuracy would plateau, AlexNet was able to achieve a top-5 error rate of 16,4% for classification. In other words, the fraction of test images for which the correct label is not among the five labels considered most probable by the model was 16,4 out of 100. That represented a major advancement from the 25,8% rate registered by the previous year's winner.

In the 2014 edition, two influential architectures were introduced, and both presented important improvements and produced another jump in performance. Both proposals explored the concept of depth of a network, to a great effect.

The first of these network, called VGG [46], had two version, one with 16 trainable layers and another with 19 layers. However, the architectural aspects of the network were largely maintained from the work on ALexNet, except for the restriction to 3×3 kernels alone and for the use of normalization, which was proved not to improve performance, but rather just to increase memory and computation time consumption. Using training parameters also very similar to [30], it achieved a top-5 error rate for classification of 7.3%, placing it second in the challenge, and won the localization challenge.

The second one, called GoogLeNet [47], presented a model with 22 trainable layers and introduced

the concept of an Inception module as its building block, in order to improve computational efficiency. The simplest version, the naïve version, of an Inception module is shown in Figure 2.3. The main idea of this module is based on finding out an optimal local sparse structure, which is crucial to computational and memory efficiency. This also allows for a certain cluster of highly correlated units in each layer, that concentrates higher abstraction characteristics captured by the bigger filter and lower ones captured by the smaller filters. However, there is one big problem with the module presented above: this approach is prohibitively expensive in terms of computational complexity, seen as the number of output filters equals the sum of filters from the previous stage. In order to tackle this issue, the strategy shown in Figure 2.4 was proposed, inspired by the work of Lin et al. [48].

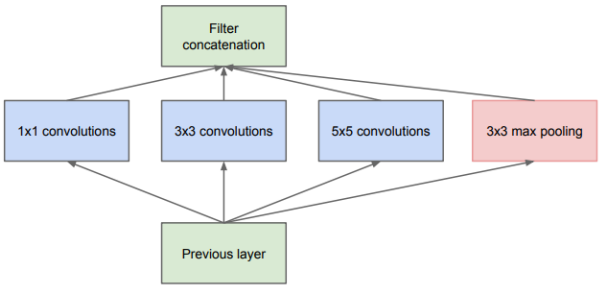


Figure 2.3: Inception module, naïve version (Szegedy et al. 2014).

The introduction of additional 1×1 bottleneck layers works as a linear combination of the depth information and achieves the desired dimension reduction. Furthermore, this layers can function as learnable pooling layers and introduce additional non-linearity to the network, which improves performance. This model won the 2014 edition of the ImageNet Challenge with a top-5 error rate of 6.7% in the classification task.

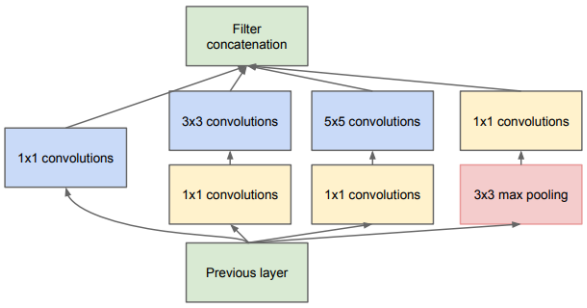


Figure 2.4: Inception module with dimension reductions (Szegedy et al. 2014).

The success of these two models demonstrated the importance of depth in the performance of a Convolution Neural Network. Taking this into consideration, the 2015 winner [49] presented a model with 152 trainable layers. This evidences the importance researchers gave to the concept of depth, as they believed deeper networks would bring a great improvement in performance. Nonetheless, it was an early finding of this work that deeper models are harder to optimize than shallower ones. Taking that into consideration, the team introduced *residual learning* and proposed the building block show in Figure 2.5. They hypothesized that, instead of learning the entire desired mapping $\mathcal{H}(x)$, a residual

mapping $\mathcal{F}(x) := \mathcal{H}(x) - x$ would be easier to optimize. This hypothesis proved very successful and the resulting network, *ResNet*, achieved great success, sweeping all competition it entered and having a top-5 error rate on the ImageNet classification challenge of 3.6%. This was a great feat, given that Russakovsky et al. [15] estimated the the top-5 error rate of a trained human annotator at 5.1%.

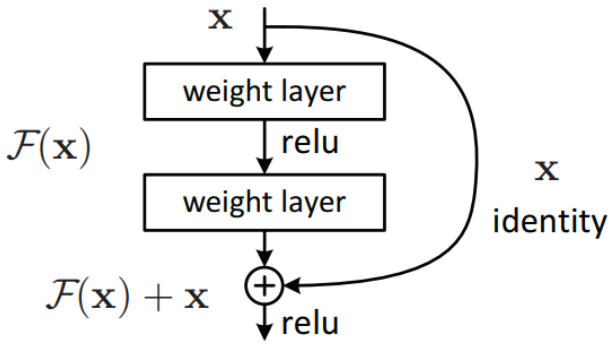


Figure 2.5: Residual learning building block (He et al. 2016).

The several networks mentioned above are generally the architectural backbone of object recognition methods. Models usually start with these networks, trained on very large datasets, and fine-tune its parameters to suit a certain situation.

2.3 Object Detection Methods

Having these networks as reference, there is a pretty elementar solution to the task of object recognition. Called the sliding window object localization, it consists of running a classifier function (CNN or not) over many rectangular subregions of an image and taking its maximum as the object's location. However, this is evidently an inefficient method, because objects can appear in several locations, sizes or aspect ratios, which leads to an enormous quantity of regions needed to be evaluated.

Consequently, several methods emerged to tackle this issue. These methods can generally be categorized into two types [50]:

- One follows a more traditional object detection pipeline: generating candidates, region proposals, at first and then proceeding to classify each proposal. The region proposal stage usually makes use of one of several different possible algorithms to generate regions of interest (RoIs) to be analyzed, like selective search [51], objectness [52] or fully connected networks. It regularly is the bottleneck of the method. The region proposal based methods may include R-CNN [53], Fast R-CNN [54], Faster R-CNN [55], R-FCN [56] and FPN [57].

- The second adopts a unified framework to achieve classification and localization directly, regarding object detection as a regression and classification problem. Examples of this approach include MultiBox [58], YOLO [59] and SSD [60].

2.3.1 Mean Average Precision (mAP) and Frames per Second (FPS)

To compare these models there is the need to define metrics capable of giving important information about the most important aspects of the task of object detection, accuracy and speed. When comparing accuracy the standard single-number metric used is mean Average Precision (mAP).

To understand the concept of mAP is helpful to separate it into two parts:

- **mean:** stands for the arithmetic mean, that is, the sum of all values divided by the number of items.
- **Average Precision:** an average of the precision values for the class.

To compute the Average Precision the concepts of precision and recall are crucial. Precision is the rate of the items predicted as positive that actually are positive and is given by Equation 2.3. Recall is the rate of positive items in the set that actually are predicted as positive, given by Equation 2.4.

$$Precison = \frac{TP}{TP + FP} \tag{2.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

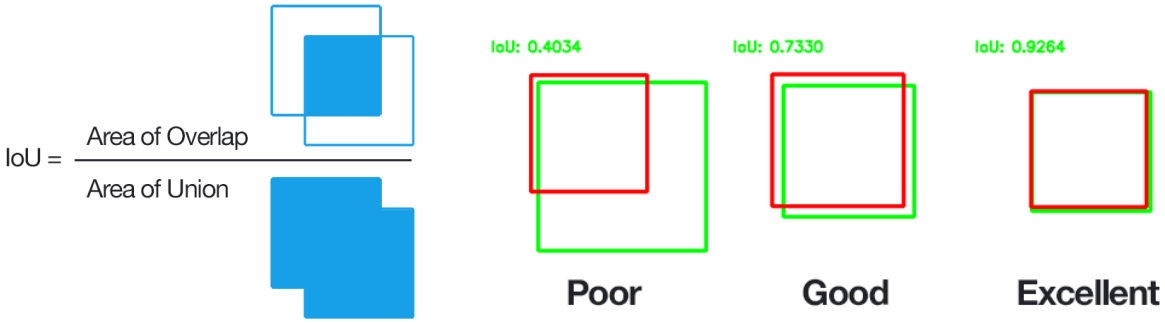
where TP stands for true positive, FP stands for false positive and FN stands for false negative. Hence, when applying to object detection, precision represents the number of times the models is correct when it detects a certain class, while recall represents the number of times a class is detected when present. The remaining problem is: how do we determine if a prediction is right of wrong. For that, it is important the concept of Intersection Over Union (IOU). The IOU is given by Equation 2.5 and it denotes the area of the overlap of the predicted bounding box and the ground-truth bounding box over the area of their union. In Figure 2.6 we can see a visual representation of IOU computation (a) and what it represents to have high or low values of IOU (b). Two bounding boxes match if their IOU is over 0.5. This is the criterion defined by the PASCAL VOC competition [27]. Other competitions have stricter criteria [29], however this will be the criterion used for this work.

$$IOU = \frac{Area(B_p \cap B_{gt})}{Area(B_p \cup B_{gt})} \tag{2.5}$$

Finally, to compute Average Precision, a precision-recall curve is produced. This curve summarizes the trade-off between precision and recall values for a model using different confidence thresholds. An example of a precision-recall curve can be seen in Figure 2.7. As computed on the PASCAL VOC competition, the computation of AP for this work is as follows:

- First, a version of the precision-recall curve with precision monotonically decreasing is produced, by setting the precision for recall r to the maximum precision obtained for any recall $r' > r$.
- The AP is finally computed as the area under this curve by numerical integration.

In this work the framework create by Cartucho et al. [61] is used to compute the mAP.



(a) Visual representation of IOU computation

(b) IOU examples

Figure 2.6: Intersection over area: computation and examples (Rosebrock 2017).

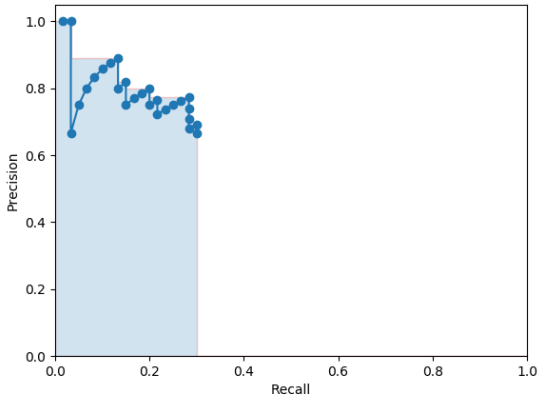


Figure 2.7: Precision-recall curve.

The second aspect important when comparing models is speed. The metric generally used in to evaluate speed is the rate of prediction, in frames per second (FPS). This is simply the inverse of the time it takes the model to predict the objects in one image.

2.3.2 Comparing Methods

Having discussed the metrics generally used to evaluate the performance of models, it is now possible to compare them. Table 2.1 shows a comparison of the methods, from the results obtained by Zhao et al. [50].

Method	mAP(%)	Test Time (sec/img)	Rate (FPS)
R-CNN	66.0	32.84	0.03
Fast R-CNN	66.9	1.72	0.6
Faster R-CNN(VGG16)	73.2	0.11	9.1
Faster R-CNN(ResNet101)	**83.8**	2.24	0.4
YOLO	63.4	**0.02**	**46**
SSD300	74.3	**0.02**	**45**
SSD512	76.8	0.05	19
YOLOv2	78.6	0.03	40

Table 2.1: Object Detection methods: comparison of performance on VOC07 test set (Zhao et al. 2018).

From the analysis of Table 2.1 it becomes pretty clear the general advantages and disadvantages of both types of methods. The methods that have a candidate generation stage are generally more accurate than the single shot methods, however that accuracy comes at the expense of speed. When considering an application of object detection on video feeds, both these parameters are determinant to the success of the application and their relative importance should be evaluated in a case by case basis.

2.3.3 Transfer Learning

In practice, very few people train an entire Convolutional Network from scratch as it requires large quantities of data to perform well and a training process that takes an incredibly large amount of time. Instead, it is common to pretrain a CNN on a very large dataset, such as ImageNet or MSCOCO, and then use it either as an initialization or a fixed feature extractor for the task to be implemented. In fact, all of the methods described above make use of this pretrained networks to implement the task of object detection. This process is known as Transfer Learning. Goodfellow et al. [31] define Transfer Learning as the situation where what has been learned in one setting is exploited to improve generalization in another setting. The same authors give the following example: "we may learn about one set of visual categories, such as cats and dogs, in the first setting, then learn about a different set of visual categories, such as ants and wasps, in the second setting. If there is significantly more data in the first setting, then that may help to learn representations that are useful to quickly generalize from only very few examples drawn from the second setting". Furthermore, Olivas et al. [62] define Transfer Learning as the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned. When training an object detector, it is extremely common to use Transfer Learning. Several libraries and frameworks provide *Model Zoos* with several example models, which can be fine-tuned to perform a specific task. This will be the procedure adopted for this work.

2.4 R-CNN: Regions with CNN features

R-CNN emerged in 2014 by the works of Girshick et al. [53] and was one of the first algorithms to propose applying Convolutional Neural Networks to the problem of object detection. A simplified model of the algorithm can be observed in Figure 2.8. The method, as proposed by the authors, can be divided into three modules: the generation of category-independent region proposals; the use of a Convolutional Neural Network, resulting in a fixed length features vector; and finally a classification step.

The first module uses selective search [51] to generate candidate regions, called region proposals or Regions of Interest (RoI), were objects may appear. The selective search algorithm works by generating sub-segmentations of the image that could belong to one object — based on color, texture, size and shape — and iteratively combining similar regions to form objects. In the case of R-CNN, selective search generates 2000 region proposals that have different sizes and aspect ratios. Seen as the second module can only deal with fixed sized inputs, the regions have to be warped into 227×227 RGB images.

For the second module, the authors originally used the AlexNet [30] to generate a 4096-dimensional

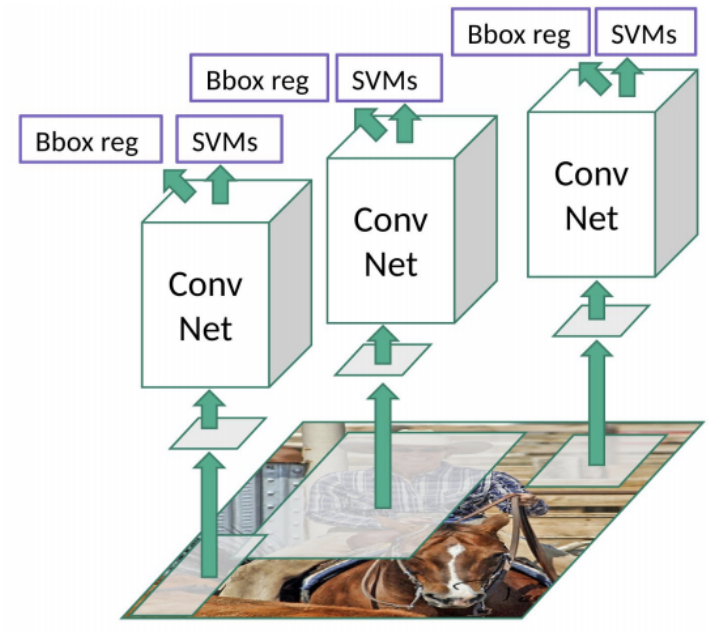


Figure 2.8: R-CNN representation (Girshick et al. 2014).

features vector for each of the 2000 region proposals. This meant that on each step (each stage of training or at test time) there were 2000 passes through a CNN. The network was firstly pre-trained on the ILSVRC2012 classification dataset for a 1000 classes image classification task, so that convolution layers could learn general basic image features. The parameters are then fine-tuned to be applicable to the situation at hand, specifically the distorted input images and specific target classes for the detection task. This is achieved by:

- Replacing the final 1000 classes layer of the network obtained in pre-training by a (N + 1) classes randomly initialized softmax classifier, where the additional class is use for the general background class.
- Each region proposal is mapped to the ground truth instance with which it has maximum IoU. If the IoU is greater than 0.5, the RoI is labeled as a positive for the ground truth class. Otherwise, the box is treated as background.
- The network is trained using SGD at a learning rate of 0.001. In each iteration 32 windows with object and 96 background windows are sampled to form a mini-batch of size 128. The sampling is biased to positive windows because they are much scarce.

The third module corresponded to a set of class-specific Support Vector Machines (SVMs) - an individual classifier for each class - that scored each of the features vector for the region proposals, resulting in a $2000 \times N$ matrix of scores, where 2000 corresponds to the number of region proposals and N to the number of classes. For this stage, when training for each particular class, all regions that have a IoU of more than 0.3 with the bounding box ground truth are considered positive. The overlap threshold, 0.3, was selected by the authors through a grid search over $\{0, 0.1, ..., 0.5\}$ on a validation set.

In addition to predicting the presence of an object within the region proposals, the algorithm also predicts four values which are offset values to increase the precision of the bounding box. The input to the bounding box regressor are the pairs of predicted proposals P and the target proposals G, shown in Equation 2.6.

$$\begin{aligned} P^i &= (P^i_x, P^i_y, P^i_w, P^i_h) \\ G^i &= (G^i_x, G^i_y, G^i_w, G^i_h) \end{aligned} \tag{2.6}$$

The target transformations that need to be learned are given by Equation 2.7.

$$\begin{aligned} t_x &= (G_x - P_x)/P_w \\ t_y &= (G_y - P_y)/P_h \\ t_w &= \log(G_w/P_w) \\ t_h &= \log(G_h/P_h) \end{aligned} \tag{2.7}$$

where the first two represent a scale-invariant translation of the center of the bounding box and the final two represent logarithmic space transformations of the width w and height h.

The predicted transformation is given by $d_k(P)$, which is modeled as a linear function of the features vector resulting from the CNN, denoted by ϕ_5. So, we have $d_k(P) = w_k^T \phi_5(P)$, where w_k^T is a vector of learnable model parameters.

$\hat{G}$, that is, the corrected predicted bounding box is given by Equation 2.8.

$$\begin{aligned} \hat{G}_x &= P_w d_x(P) + P_x \\ \hat{G}_y &= P_h d_y(P) + P_y \\ \hat{G}_w &= P_w \exp(d_w(P)) \\ \hat{G}_h &= P_h \exp(d_h(P)) \end{aligned} \tag{2.8}$$

Finally w_k is learned by optimizing Equation 2.9

$$w_k = \underset{x}{\operatorname{argmin}} \sum_i^N (t^i_k - \hat{w}_k^T \phi_5(P^i))^2 + \lambda \left\| \hat{w}_k \right\|^2 \tag{2.9}$$

Although R-CNN represented a major evolution in the object detection problem, some major drawbacks were easily spotted. First of all, the method was still quite slow, as visible in Table 2.1 with test times for each image reported between 40 and 60 seconds, putting it far from real-time object detection. It is also very time-consuming and computationally costly to train on 2000 region proposals. Another issue concerns the fact that training is a multi-stage process (first fine-tuning a CNN, then fitting SVMs to CNN features and then learning bounding box regressors). Finally, the selective search algorithm is fixed, that is, no learning is happening at that stage, which leads to the generation of bad region proposals.

Aware of these drawbacks, Girshick improved on his earlier work and published in 2015 Fast R-CNN [54]. The focus of this work was to improve on R-CNN and, mainly, making it faster. The main issue that caused R-CNN to be slow was the fact that 2000 different crops of images were fed into a Convolutional Neural Network. Taking that into consideration, Fast R-CNN takes an entire image as input to a CNN. The original image is also used to generate region proposals, again using selective search. Several convolutional and max pooling layers produce a convolutional feature map of the original image. This feature map is then cropped using the projections of the region proposals obtained before, using a RoI pooling layer. This layer works by dividing the $h \times w$ RoI window into an $H \times W$ grid of sub-windows of approximate size $h/H \times w/W$ and then max-pooling the values in each sub-window into the corresponding output grid cell. This gives rise to fixed-size output features of size $(H \times W)$ irrespective of the input size. H and W are chosen such that the output is compatible with the network's first fully-connected layer. These features vector are then fed into a sequence of fully-connected networks that finally branch into two sibling output layers: one used to classify the objects in the image and the other to refine the bounding boxes produced through offsets. A representation of the method can be seen in Figure 2.9.

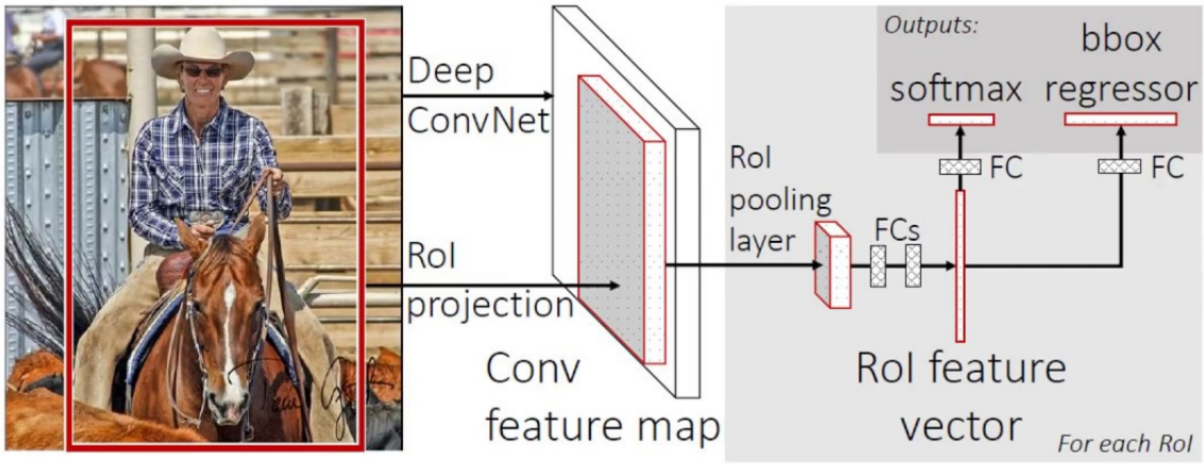


Figure 2.9: Fast R-CNN representation: the original chosen CNN was a *VGG16* network (Girschick 2015).

The first of the two outputs of the model (per RoI) is a discrete probability distribution, $p = (p_0, ..., p_N)$, over $N + 1$ categories. p is computed by a softmax classifier over the $N + 1$ outputs of a fully connected layer. The second of the outputs are bounding-box regression offsets, $t^n = (t^n_x, t^n_y, t^n_w, t^n_h)$, for each of the N object classes. t^n specifies a scale-invariant translation and log-space height/width shift relative to an object proposal, calculated as in Equation 2.7.

The fine-tuning of the parameters, that is, the training process, can jointly optimize the classifier and the bounding box regressors, through a multi-task loss, given by Equation 2.10.

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v) \tag{2.10}$$

L_{cls} is the factor related to the classification task and is given by $L_{cls}(p, u) = -\log p_u$, that is, the log loss for true class u. The loss related to the localization task, L_{loc}, is defined over a ground truth

v for class u, given by $v = (v_x, v_y, v_w, v_h)$ and the prediction for the same class, $t = (t_x^u, t_y^u, t_w^u, t_h^u)$. The function $[u \geq 1]$ is equal to 1 when $u \geq 1$ and 0 otherwise. The background class is labeled, by convention, $u = 0$ and, therefore, L_{loc} for such a class is ignored. L_{loc} is then given by Equation 2.11.

$$L_{loc}(t^u, v) = \sum_{i \in \{x,y,w,h\}} \text{smooth}_{L_1}(t_i^u - v_i) \tag{2.11}$$

where smooth_{L_1} is a robust L_1 loss, given by Equation 2.12.

$$\text{smooth}_{L_1} = \begin{cases} 0.5x^2, & \text{if} |x| < 1 \\ |x| - 0.5, & \text{otherwise} \end{cases} \tag{2.12}$$

The parameter λ of Equation 2.10 is used to control the balance between the two losses. The original results were obtained for $\lambda = 1$.

Concerning the training pipeline, like in R-CNN, 25% of the ROIs are object proposals that have at least 0.5 IoU with a ground-truth bounding box of a foreground class. Unlike R-CNN, each mini-batch is constructed from 2 images, with 64 RoIs from each image.

In essence, Fast R-CNN produced several improvements to R-CNN family of algorithms, namely:

- Higher detection quality (mAP) than R-CNN.
- Training is single-stage, using a multi-task loss.
- Training can update all network layers.
- The method performs much faster at test time. In fact, there was such an improvement, that selective search became the bottleneck for the method, as can be seen in Figure 2.10. Analysing the figure we easily conclude that most of test time for Fast R-CNN, 2.3 seconds, it taken by selective search, which takes about 2 seconds.

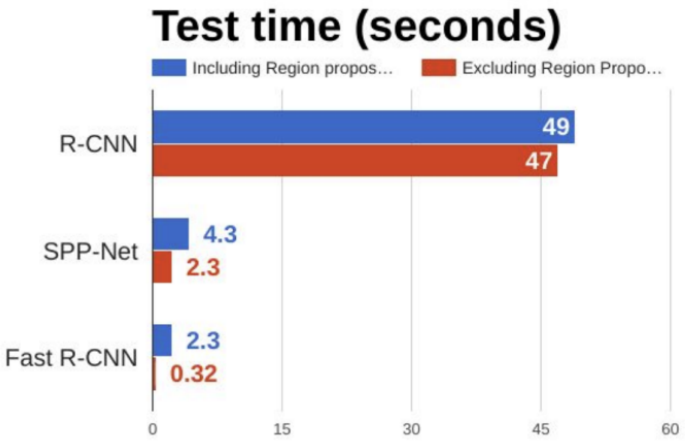


Figure 2.10: Test time comparison between R-CNN and Fast R-CNN (Girshick et al. 2014, He et al. 2014, Girschick 2015).

Taking the last statement into account, a new method was proposed in 2016 by Ren et al. [55]. The authors introduced a new method, called Faster R-CNN, that eliminates the selective search algorithm and lets the network learn the region proposals. Similar to Fast R-CNN, the image is provided as an input to a convolutional network which generates a convolutional feature map. Instead of using selective search algorithm on the feature map to identify the region proposals, a separate network is used to predict the region proposals - the Region Proposal Network (RPN). This lowers the region proposal time from 2 seconds per image, to about 10 ms, while it also allows the region proposal stage to share layers with the following detection stages, causing an overall improvement in feature representation. A simple representation of the model can be seen in Figure 2.11.

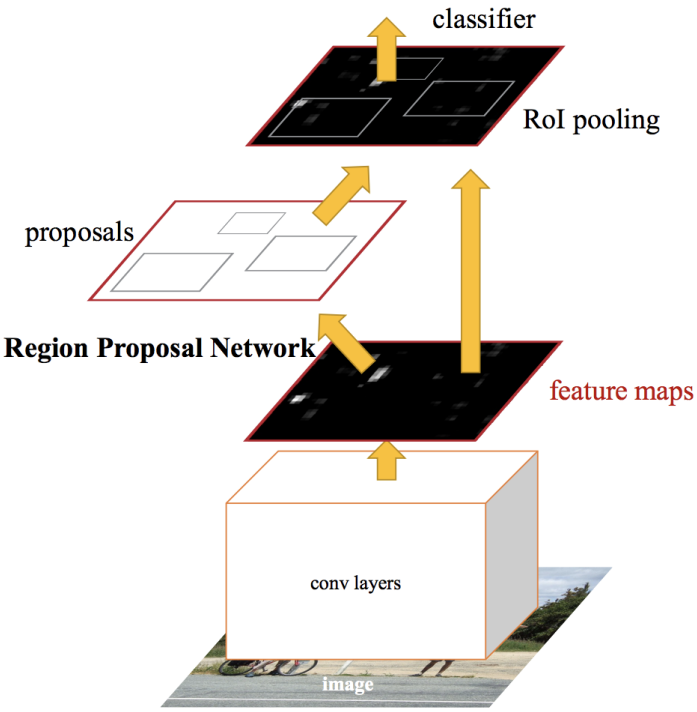


Figure 2.11: Faster R-CNN representation (Ren et al. 2016).

The Region Proposal Network also makes the use of a set of Anchors. For every point in the output feature map, the network has to learn whether an object is present in the input image at its corresponding location and estimate its size. This is done by placing a set of Anchors on the input image for each location on the output feature map of the backbone network. These anchors are a set of rectangular shapes in various sizes and aspect ratios at this location.

A representation of the Region Proposal Network architecture is shown in Figure 2.12. It shows that the network is divided in two branches: the regression branch and the classification branch. The former outputs the 4 regression coefficients used to improve the coordinates of the anchors boxes that contain objects as done in Fast R-CNN. The latter gives the probabilities of whether or not each point of the feature map contains an object within all 9 of the anchors at that point.

Overall, Faster R-CNN can be thought as the RPN as a region proposal algorithm and Fast R-CNN as a detector network. The results of the model were quite good, mainly in its objective of reducing the

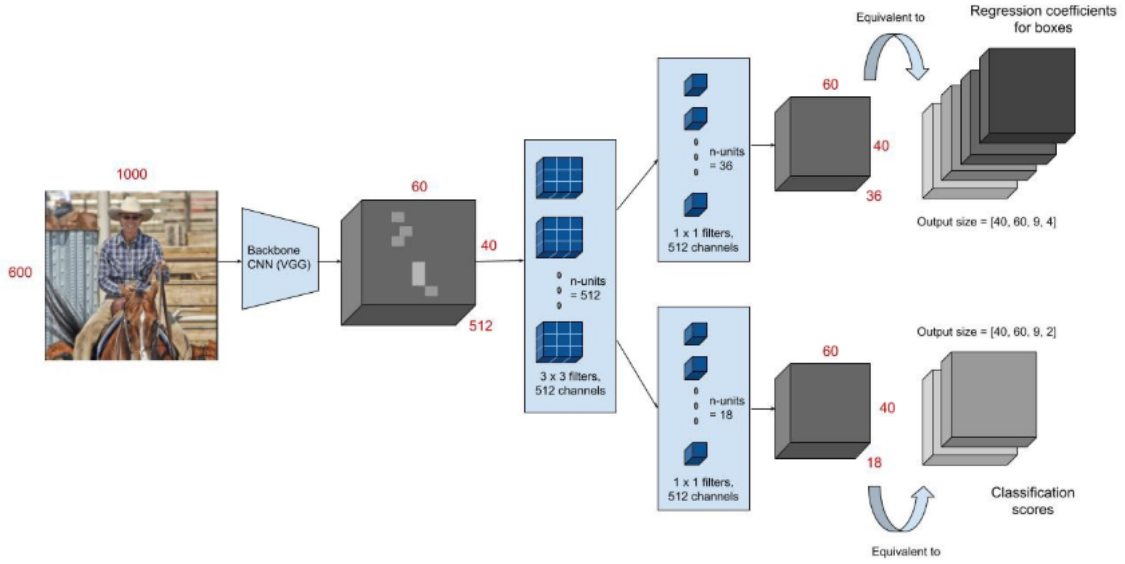


Figure 2.12: The architecture of the Region Proposal Network (Ananth 2019[1]).

time taken by the region proposal stage. The authors reported that detection with a VGG RPN takes 198ms compared to the 1.8 seconds of Selective Search.

This work makes use of the Faster R-CNN algorithm, seen as it is the most improved and faster of the R-CNN family. It is also important to note it achieve the best results in terms of accuracy when compared to others methods.

2.5 YOLO: You Only Look Once

You Only Look Once (YOLO) is a general purposed object detection algorithm that frames the task as a unified problem, using the same deep neural network to predict bounding boxes and class probabilities [59]. It belongs, therefore to the second of the categories described in Section 2.3. YOLO was the first of the deep learning based algorithms to achieve real time object detection.

YOLO divides the input image into a $S \times S$ grid, as can be seen in Figure 2.13. Each grid cell is responsible for the prediction of B bounding boxes and the confidence scores for those boxes. This confidence score reflects how certain the model is that the box contains an object and also how accurate it estimates the predicted box is. It is given by Equation 2.13.

$$Pr(Object) \times IOU_{pred}^{truth} \tag{2.13}$$

where $Pr(Object)$ is the probability of an object existing in the box and IOU_{pred}^{truth} is the Intersection Over Union of the areas of the predicted and ground truth bounding boxes.

[2]Available at https://towardsdatascience.com/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46

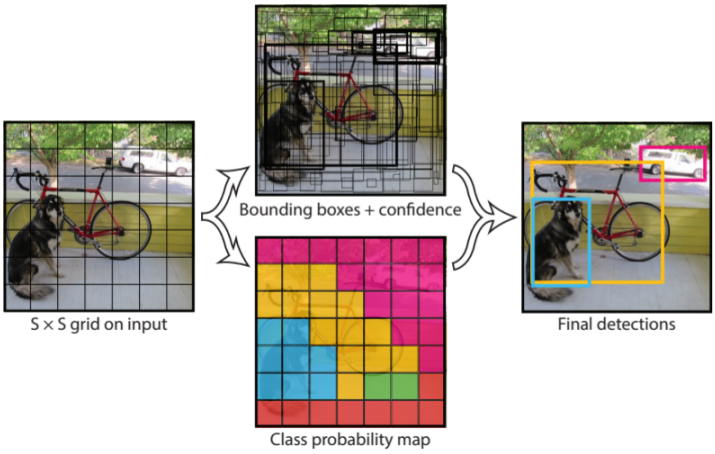


Figure 2.13: YOLO Pipeline (Redmon et al. 2016).

Each bounding box consists of 5 predictions: x and y, the coordinates of the center of the box, h and w, the height and width, and the confidence score. Each grid cell is also responsible for providing one, and only one, set of conditional probabilities for all the C classes $Pr(Class_i|Object)$, independent of the number of bounding boxes. Thus, the final confidence of a prediction at test time is given by:

$$Pr(Class_i|Object) \times Pr(Object) \times IOU_{pred}^{truth} = Pr(Class_i) \times IOU_{pred}^{truth} \tag{2.14}$$

It is clear from Equation 2.14 that the test time score takes into consideration both the probability of a class appearing in a box and how well a box fits the object. The final prediction is, thus, encoded in a $S \times S \times (B * 5 + C)$ tensor.

The convolutional neural network architecture of YOLO is inspired by the GoogLeNet [47] and Lin et al. [48] works discussed above. Yet, instead of using the Inception modules used by Szegedy et al. [47], it simply uses the 1×1 bottleneck layers followed by 3×3 convolutional layers. It has 24 convolutional layers followed by 2 fully connected layers. The full architecture of the network can be seen in Figure 2.14. The shown example was trained on Pascal VOC, that possesses 20 classes, with $S = 7$ and $B = 2$. Hence, the output is a $7 \times 7 \times 30$ tensor

Except for the final layer, the used activation function is a modified version of a ReLU, a leaky rectified linear unit (Leaky ReLU), given by:

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases} \tag{2.15}$$

During training, the loss function to be optimized is an adjusted version of sum-squared error. The adjustments are needed because in every image many cells do not contain any object. This leads to very low confidence scores that can overpower the gradient from cells with objects. Hence, it is necessary to increase the loss from bounding box coordinates predictions and decrease the loss from confidence for boxes that do not contain objects. To this objective, the parameters λ_{coord} and λ_{noobj} are used, and

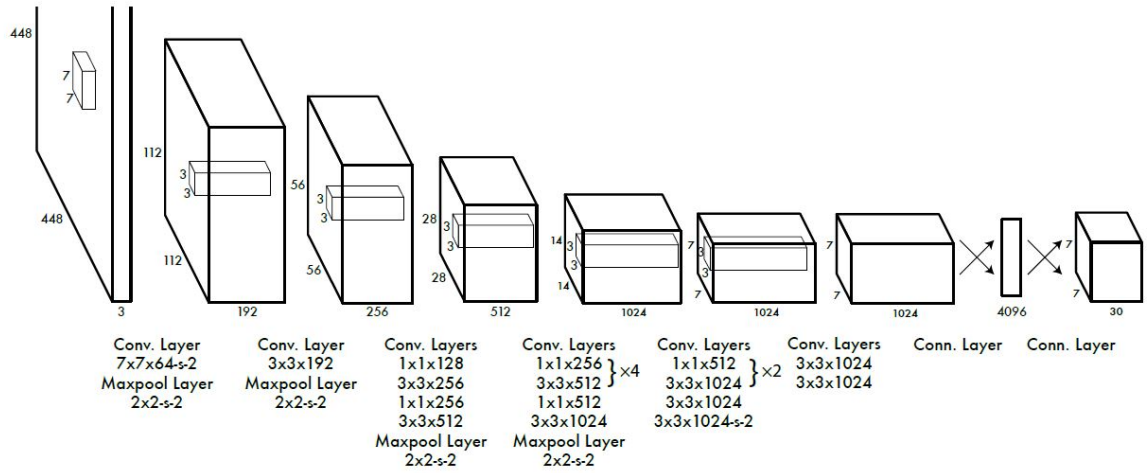


Figure 2.14: YOLO Network Architecture (Redmon et al. 2016).

set, in the original article, to 5 and 0.5, respectively. Moreover, sum-squared error would equally weight deviations in large boxes and small boxes. As this is undesirable, the square root of the bounding box width and height are used instead of its width and height. The loss function is given by:

$$\begin{aligned}
&\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2]+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}[(\sqrt{w_i})-\sqrt{\hat{w}_i})^2+\\
&+(\sqrt{h_i})-\sqrt{\hat{h}_i})^2]+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}(C_i-\hat{C}_i)^2+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{noobj}(C_i-\hat{C}_i)^2+\\
&+\sum_{i=0}^{S^2}\mathbb{1}_{i}^{obj}\sum_{c\in classes}(p_i(c)-\hat{p}_i(c))^2
\end{aligned} \tag{2.16}$$

where $\mathbb{1}_{ij}^{obj}$ indicates that the bounding box j in cell i is responsible for that prediction and $\mathbb{1}_{i}^{obj}$ indicates whether and object is present in cell i.

The first term of Equation 2.16 penalizes the deviations of the localization of the center of the bonding boxes. The second one penalizes the disparity in width and height. The third tries to maximize confidence for cells where objects exist. The fourth forces the confidence scores to 0 when there is no object in a cell. The final term is the one responsible for the classification loss. It is important to denote that the loss function only penalizes classification when an object is present in a cell.

The second version of YOLO, YOLOv2 [63], improves on the initial model. Added to some changes in the training process, namely the use of Batch Normalization, pre-training at higher resolutions and multi-scale training, the most important changes are detailed below. These alterations account for the improvement in performance one can see in Table 2.1, where YOLOv2 achieves a higher mAP compared to YOLO, despite being a bit slower.

- **The use of anchor boxes:** while the first version would randomly initialize the aspect ratio of bounding boxes, YOLOv2 uses anchor boxes, that is, chosen priors with a determined aspect

ratio, predicting, then, offsets to said boxes. This simplifies the problem and makes it easier for the network to learn. Instead of hand picking the boxes, YOLOv2 uses k-means clustering to find good priors.

- **Cell grid dimensions:** transition to a 13×13 cell grid ($S = 13$).
- **Changes to the network architecture:** the second version of YOLO discards the fully-connected layers, assuming a fully convolutional architecture, and reduces the number of convolutions to 19.

YOLO has a third version, YOLOv3 [64], published in 2018 by the original author of YOLO. There were changes made, which account for a small but important improvement when comparing with YOLOv2, in terms of accuracy. This version can be considered as an incremental improvement to the works published before. The changes of this version are:

- **Changes in the loss function computation:** The objectness scores of the loss function of YOLOv3 - the last three terms of Equation 2.16 - are computed using logistic regression, instead of the squared errors used in YOLOv2.
- **Changes in class prediction:** change from the use of softmax classifiers to independent logistic classifiers, because the former impose the assumption that each box has exactly one class. This is not true for datasets with overlapping labels - for instance, a bounding box that corresponds both to the *Woman* and *Person* classes.
- **Prediction across 3 scales:** The network downsamples the input image until the first detection layer. Then, layers are upsampled by a factor of 2 and concatenated with feature maps of previous layers having identical feature map sizes. Another detection is now made. The same upsampling procedure is repeated, and a final detection is made. YOLO authors report that this helps get better results when detecting small objects.
- **New Feature Extractor Network:** the authors introduce a new network, Darknet-53, constituted by 53 convolutional layers and no pooling layers.

This work makes use of YOLOv3 because it is the last stable version of the YOLO family of algorithms[3] and it has achieve state-of-the-art results in terms of speed, while approaching the state-of-the-art results in terms of accuracy.

2.6 SSD: Single Shot MultiBox Detector

Introduced in 2016 by Liu et al. [60], the Single Shot Detector (SSD) is a single stage method for object detection, just like YOLO, reportedly bringing a great improvement in accuracy for such class of methods. This is fairly important because the greater speed of single shot methods is many times obtained at the expense of accuracy. The algorithm uses a pretty simple and straightforward framework. It discretizes

[3] In fact, during the course of this work there were two further versions of YOLO published, YOLOv4 (April 2020) and YOLOv5 (June 2020) but further investigation is necessary before they are stable, in the opinion of the author.

the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. The use of this default bounding boxes is similar to the uses of anchors in Faster R-CNN, however they are applied on several feature maps of different resolution, that is, on different stages of the CNN. A representation of the method can be seen on Figure 2.15.

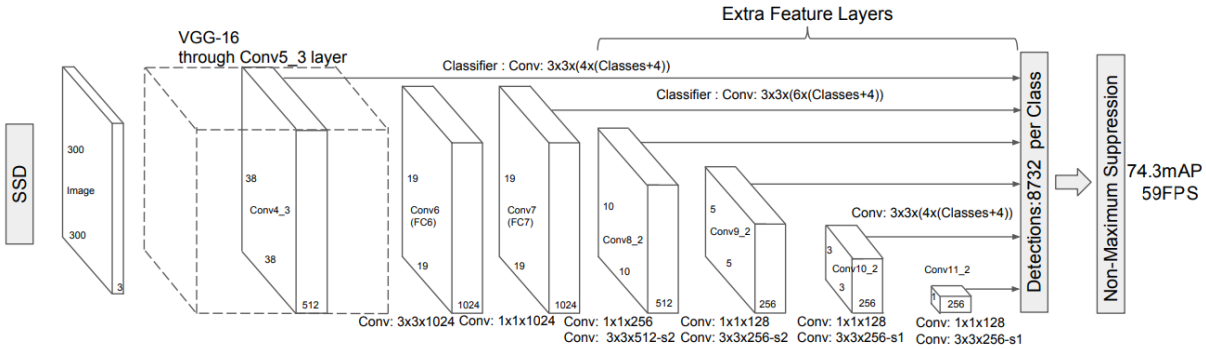


Figure 2.15: Representation of Single Shot MultiBox Detector (Liu et al. 2016).

The early network layers are based on a standard architecture used for high quality image classification. The original paper makes use of the VGG-16 architecture, truncating the fully-connected networks, however, several other architectures can be used to implement the method.

Instead of the fully-connected networks, convolutional layers are added, allowing prediction of detections at multiple scales. Each added feature layer can produce a fixed set of detection predictions using a set of convolutional filters. For a feature layer of size $m \times n$ with p channels, the basic element for predicting parameters of a potential detection is a $3 \times 3 \times p$ kernel that produces either a score for a category, or a shape offset relative to the default box coordinates.

At each feature map cell, the model predicts the offsets relative to the default box shapes in the cell, as well as the per-class scores that indicate the presence of a class instance in each of those boxes. Specifically, for each box out of k at a given location, it computes c class scores and the 4 offsets relative to the original default box shape. This results in a total of $(c + 4)k$ filters that are applied around each location in the feature map, yielding $(c + 4)kmn$ outputs for a $m \times n$ feature map.

When training the model, a key characteristic is its matching of the default boxes to a ground truth. First, each ground truth box is matched to the default with which it has the greatest IoU. Then, the default boxes are matched to any ground truth with which it has an IoU greater than 0.5. This second step is an important difference from other works [58] from which this matching is inspired.

The objective function to be minimized is shown in Equation 2.17, where $x_{ij}^{p} = \{1, 0\}$ is an indicator for matching the i-th default box to the j-th ground truth box of category p, and N is the number of matched default boxes.

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g)) \tag{2.17}$$

The localization loss, $L_{loc}(x, l, g)$, is a Smooth L1 loss between the predicted box (l) and the ground truth box (g) parameters, given by Equation 2.18 for the offsets regressed for the center (cx, cy) of the

default bounding box (d) and for its width (w) and height (h).

$$
\begin{aligned}
L_{loc}(x, l, g) &= \sum_{i \in Pos}^{N} \sum_{m \in (cx, cy, w, h)} x_{ij}^{k} \text{smooth}_{L_1}(l_i^m - \hat{g}_j^m) \\
\hat{g}_j^{cx} &= (g_j^{cx} - d_i^{cx})/d_i^w \\
\hat{g}_j^{cy} &= (g_j^{cy} - d_i^{cy})/d_i^h \\
\hat{g}_j^{w} &= \log(\frac{g_j^w}{d_i^w}) \\
\hat{g}_j^{h} &= \log(\frac{g_j^h}{d_i^h})
\end{aligned}
\tag{2.18}
$$

The confidence loss, $L_{conf}(x, c)$ is the softmax loss over multiple classes confidences (c), described in Equation 2.19.

$$
L_{conf}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^{p} \log(\hat{c}_i^p) - \sum_{i \in Neg}^{N} \log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p c_i^p} \tag{2.19}
$$

Additionally, still on the subject of training, data augmentation and positive biased sampling are used in order to improve results, the same way they are used in R-CNN.

This works makes use of this algorithm because it achieved results very closely matched with the ones obtained by the YOLO algorithm, to an extend that is difficult to separate them in terms of performance.

Chapter 3

Implementation

In this chapter, the implementation of the algorithms discussed in Chapter 2, as well as some additional tools used for their implementation, will be discussed. The implementation can be separated into two distinct phases corresponding to two different environments and groups of objects to be detected.

The first phase intends to mirror an household or office environment, more specifically a desk with everyday items that are used in these environments, such as pens, cellphones, mugs, bottles, among others. In this stage, the images and annotations used for training were downloaded from an online dataset, in order to evaluate if the quantity and quality of such open sourced data is enough to obtain a working model.

The second phase corresponds to a simulated electronics laboratory or workshop, where the detection is intended for objects such as capacitors, resistors or potentiometers. The images used at this point, were downloaded from standard searches on the internet and manually annotated by the author. This was done in order to determined if an untrained subject could provide quality data to train the models.

All the models were implemented in the Python programming language. This language was chosen because Python is the most widely used language for Deep Learning applications, with a significant open sourced community, several frameworks dedicated mostly to Deep Learning, such as Tensorflow [65], PyTorch [66], Theano [67], Keras [68] or Caffe [69], and extensive bibliography on the subject [70–72].

All the discussed models were trained, except otherwise stated, until the loss stagnated. Some exceptions to the former statement are due to blunders or time and resources limitations. For example, some models were trained overnight. Training stoppage was forced manually by the author, that is, no automatic stoppage criteria was programmed. Furthermore, all the hyperparameters of training were the default parameters of the frameworks used, unless otherwise stated.

The next sections will detail the frameworks used to implement this work, the database used on the first phase as well as some specifics of the trained models.

3.1 Tensorflow API

According to its authors, Tensorflow [65] is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms. It falls under the category of Deep Learning Frameworks, some examples of which were declared in the introduction to this chapter. In a simple manner, these frameworks are interfaces, libraries or tools which allow an individual to build Deep Learning models in a simple and fast way, while also optimizing the computational process.

Tensorflow is perhaps the most popular and used of these frameworks. Created by the Google Brain team and open-sourced in 2015, it presents several advantageous features. It supports multi-CPU and GPU executions, which not only greatly impacts the time needed for training but also decreases the time needed for testing and implementation. Furthermore, it is possible to use the same computation expressed with Tensorflow in a wide range of systems, ranging from large-scale distributed systems to small mobile devices (Tensorflow Lite).

TensorFlow Object Detection Application Programming Interface (Tensorflow API) [73] is an open source framework built on top of TensorFlow that aims to make it easy to construct, train and deploy object detection models. Together with the TensorFlow Model Zoo, TensorFlow Object Detection API provides the user with multiple pre-trained object detection models with instructions and example codes for fine-tuning and transfer learning.

There is a great advantage in using pre-trained models, initially trained with massive databases, requiring computational times and resources not feasible for most individuals or institutions. This pre-train contributes to the model's "perception" of what is an object and its general characteristics, even before dividing the objects into specific classes. However, these are not final models. Using transfer learning, they need to be adjusted to their final purpose.

In this work, a Faster R-CNN model and a SSD model, both with a backbone convolutional architecture using the Inception module and both pre-trained on MSCOCO [29], were selected from the Model Zoo and trained using the Tensorflow API. The reported mAP were 28 for the former and 24 for the latter and the speeds of inference were of 58ms and 42ms, respectively.

3.2 Darknet and Ultralytics repository

While the SSD and Faster R-CNN models were possible to be trained using the Tensorflow API, the same cannot be said for the YOLO models. When YOLO was proposed, the author also released a specific open-sourced framework written in C, *Darknet*, in order to facilitate the training of such models. However, seen as C is not as user-friendly of a programming language as Python, several translators were built, generally with compatibility with Tensorflow or PyTorch. These libraries are very useful, seen as, they enable Transfer Learning from the pre-trained models in huge datasets. This work makes use of one of those libraries [74], with compabilbity with PyTorch. It was chosen because it is a stable library with extensive work and good documentation.

3.3 Models Trained Using a Curated Dataset

3.3.1 The Open Images Dataset

The Open Images Dataset V4 [75][1] was launched in 2018, by a team working in Google AI. It contains over 9.2 Million annotated images for the tasks of image classification, object detection and visual relationship detection. For object detection in particular, it provides 15.4 Million bounding boxes for 600 object classes, annotated on 1.9 Million images, which, the authors state, is 15 times more bounding boxes than the next largest datasets.[2]

One of the major advantages of this dataset has to do with the way it was created. The images it contains were collected from *Flickr*, a popular images and video hosting site, without any predefined list of classes or tags. The images that appeared somewhere else on the internet were then removed, in order to reduce the bias towards web imagery. This is greatly beneficial when training an object detection algorithm because the images are presented in a natural ambient, often in cluttered environments, easily verified by its average of 8 annotated objects per image. This technique also leads to natural class proportions. These characteristics are highly desirable because they construct training data very similar to the expected images when testing real world applications while avoiding initial design bias.

Besides the bounding boxes' location and dimensions, images in this dataset are annotated with five other attributes:

- *GroupOf*: the box covers more than five instances of the same class which heavily occlude each other.
- *Partially occluded*: the object is occluded by another object in the image.
- *Truncated*: the object extends outside of the image.
- *Depiction*: the object is a depiction such as a cartoon, drawing, or statue.
- *Inside*: the box captures the inside of an object (e.g. inside of an aeroplane or a car).

Attribute	Occluded	Truncated	GroupOf	Depiction	Inside
Frequency	66,06%	25,09%	5,99%	5,45%	0,24%

Table 3.1: Percentage of boxes with the five different attributes on Open Images (Kuznetsova et al. 2018).

Table 3.1 details the percentage of images labeled with each of this five attributes. The table further evidences that the dataset was carefully built to be mostly composed of images in natural, everyday scenes. These characteristics can, hopefully, lead to an object detection model capable of detecting an object even if it is behind another object or not entirely in the frame. The low percentage of the inside attribute can lead the reader to assume a bias towards outdoor scenes, however this can be misleading.

[1] available at `https://storage.googleapis.com/openimages/web/index.html`

[2] A flare dendrogram with all the 600 boxable objects can be found in `https://storage.googleapis.com/openimages/2018_04/bbox_labels_600_hierarchy_visualizer/circle.html`

This happens because the percentages refer to all the bounding boxes in an image, while the *Inside* attribute only refers to the object that is generally the background of said image. For instance, an apple, a pen and a mug can be inside an aeroplane, and while all the bounding boxes will be referent to an indoor scene, only the aeroplane's box will be annotated with the *Inside* attribute.

Obviously, not all the classes in the Open Images Dataset are equally frequent. Figure 3.1 plots the number of annotated boxes for each class. Besides further evidencing the size advantage of this dataset - most classes have one order of magnitude more boxes annotated than the next largest dataset - it also hints at a difficulty for the work of this report. The most common classes when analysing most datasets used in object detection applications tend to be related to the presence of humans or animals. The classes related to the task of driving - cars, traffic signs, sidewals, etc. - are also quite common. However, when creating applications that deal with other classes, for instance office supplies or household appliances, the quantity of data tends to diminish.

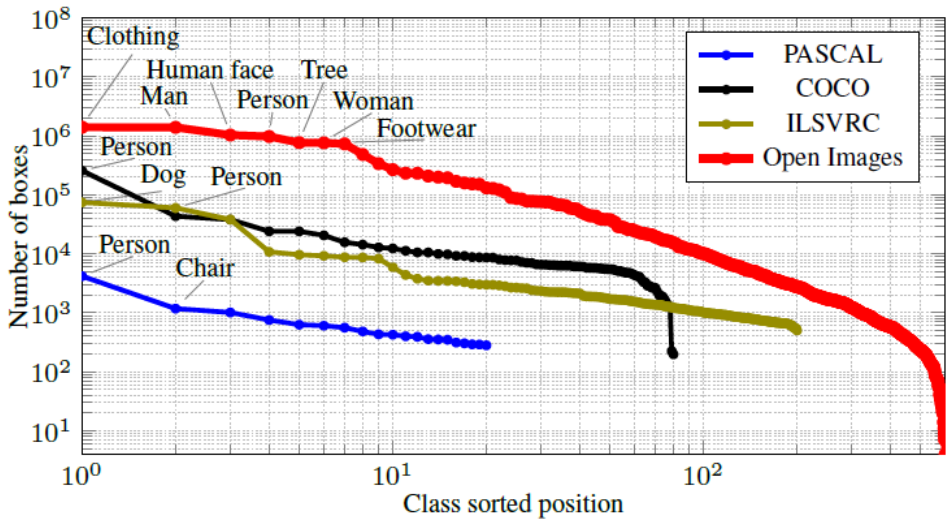


Figure 3.1: Number of boxes per object class on the Open Images Dataset V4 (Kuznetsova et al. 2018).

Finally, when analysing the quality of the dataset the authors report quite good results for the dataset. The precision and recall for the analysed images are respectively 97.7% and 98.2%, both very high values. Furthermore, the authors also analysed the quality of the bounding boxes, reporting an IoU of 0,87 when evaluating the geometric agreement of the boxes, that is, the IoU of two boxes drawn of the same object by two different annotators.

OIDv4 Toolkit [76] was used when downloading the images.

3.3.2 Models trained on 1 class

In order to test the feasibility of training an object detector and get a feeling for some of the conditions needed for the train process, for example the number of images and the time needed for the process, the first practical application tested was training a model on just one class. Such a class should be, in the opinion of the author, an object of everyday, intensive and widespread use. It was also needed to take into consideration both the scenarios and the environments that serve as reference for this report.

Several classes of the Open Images Dataset were considered, finally settling on the *Pen* class.

In the used dataset, 198 images were available for such a class, annotated with 293 bounding boxes. Figure 3.2 shows some examples of the images used for training with the respective ground-truth bounding boxes annotated.

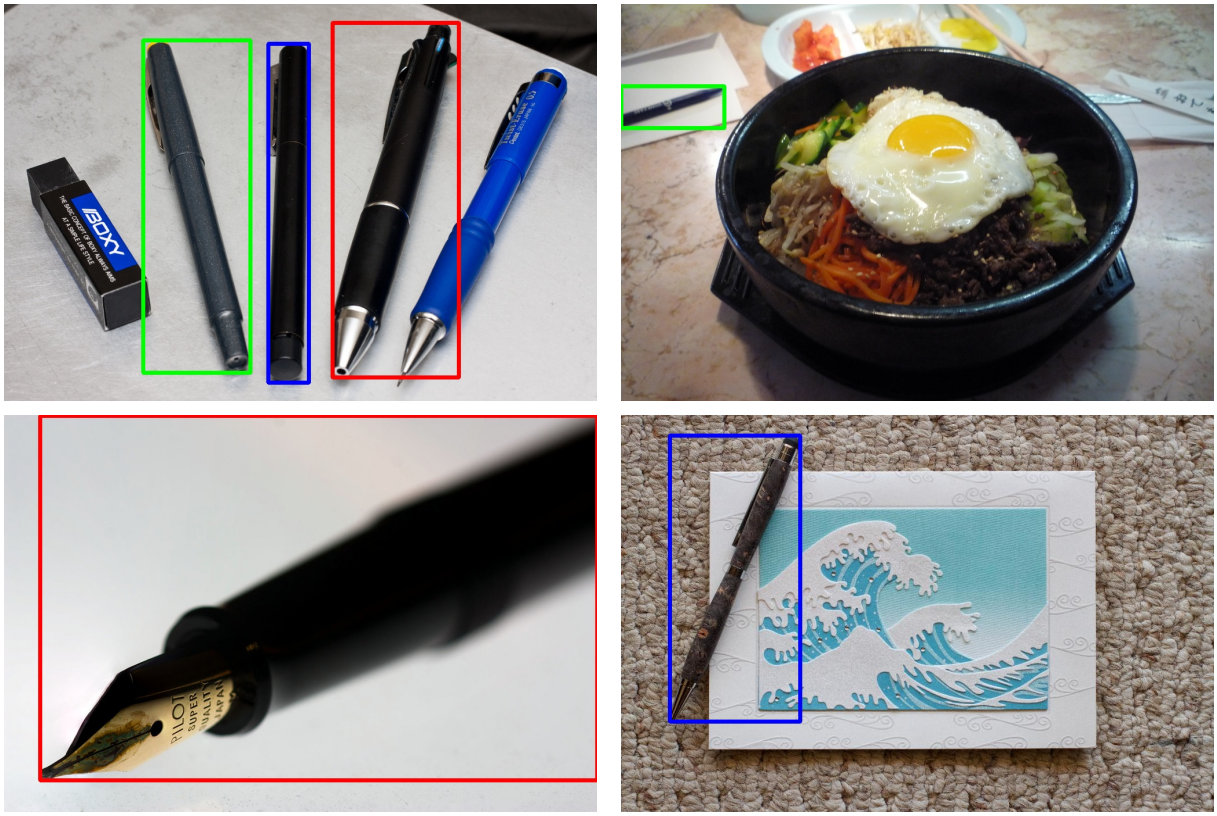


Figure 3.2: A sample of the images used to train the 1-class models with the respective ground-truth bounding boxes (Kuznetsova et al. 2018).

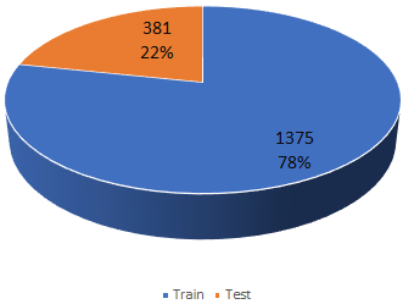
The images were divided into two groups, corresponding to the train and test partitions. The first received 159 images, annotated with 244 bounding boxes, while the test partition was composed of 39 images, corresponding to 49 object examples. This represents, approximately, a 4 to 1 proportion on images and a 5 to 1 proportion on bounding boxes.

3.3.3 Models trained on 4 classes

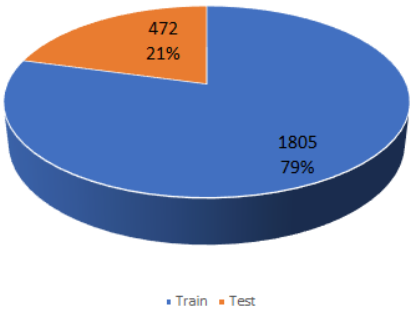
To further the knowledge on training and to analyze the impact of having multiple classes on a trained model, a second set of models were trained to detect four classes. Besides the class already chosen on Section 3.3.2, the new classes chosen were: *Glasses* (as in vision-aiding glasses), *Mobile phone* and *Mug*. These classes were chosen because they are commonly found items on a office or home tables and, combined with that, they have a comprehensive set of images in the Open Images Dataset.

At this stage, all the images available on the Open Images Dataset for each class were used. As before, the images were divided into train and test partitions. The first entailed 1375 images and the second got 381 images, with 1805 and 475 bounding boxes, respectively. This data can be observed

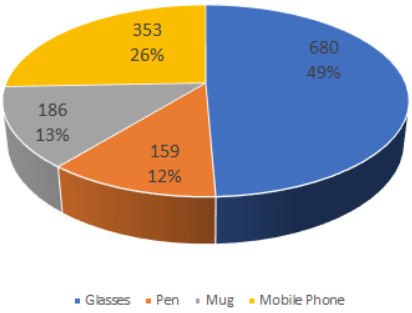
in Figure 3.3, where it is also possible to examine the proportion of each class in the partitions, both in images and in bounding boxes.



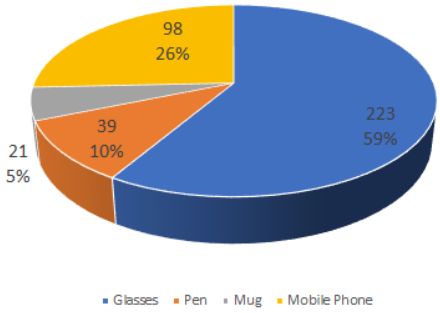
(a) Division of images into the train and test partitions.



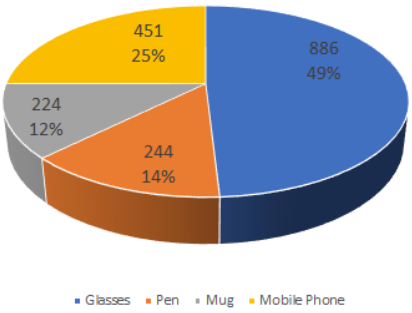
(b) Division of bounding boxes into the train and test partitions.



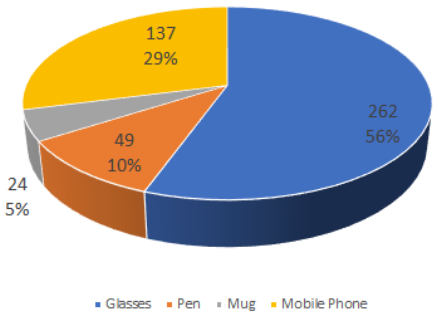
(c) Proportion of class images in the train partition.



(d) Proportion of class images in the test partition.



(e) Proportion of class bounding boxes in the train partition.



(f) Proportion of class bounding boxes in the test partition.

Figure 3.3: Characteristics of the set of images used to train the four-classes models.

When training the SSD model there was a necessity for some adjustments. Due to the limitations of some of the used equipment, namely the GPU running out of memory, the batch size had to be reduce, from the original 64 to 8 images per batch, and the dimensions of the images (width and height) had to be reduced by half. The latter was done using the *open-cv* function *resize* and the interpolation method INTER-AREA, that is, resampling using pixel area relation.

A sample of the images used for the three additional classes can be seen in Figure 3.4, with the annotated bounding boxes. In the Figure, it is possible to denote some potential problems, specially for the *Glasses* class. In fact, most of the examples for this class aren't images of the isolated object, but rather of the object on a human face. This could lead the model to expect some characteristics of a

human face around the object when identifying glasses, which isn't helpful on a tabletop environment. Furthermore, the class has many examples of different types of glasses, such as diving goggles or sunglasses, that are not necessarily of interest to our model. The other two classes are in general, better suited to training, with several examples of different types of mobile phones or mugs and different poses for those items.

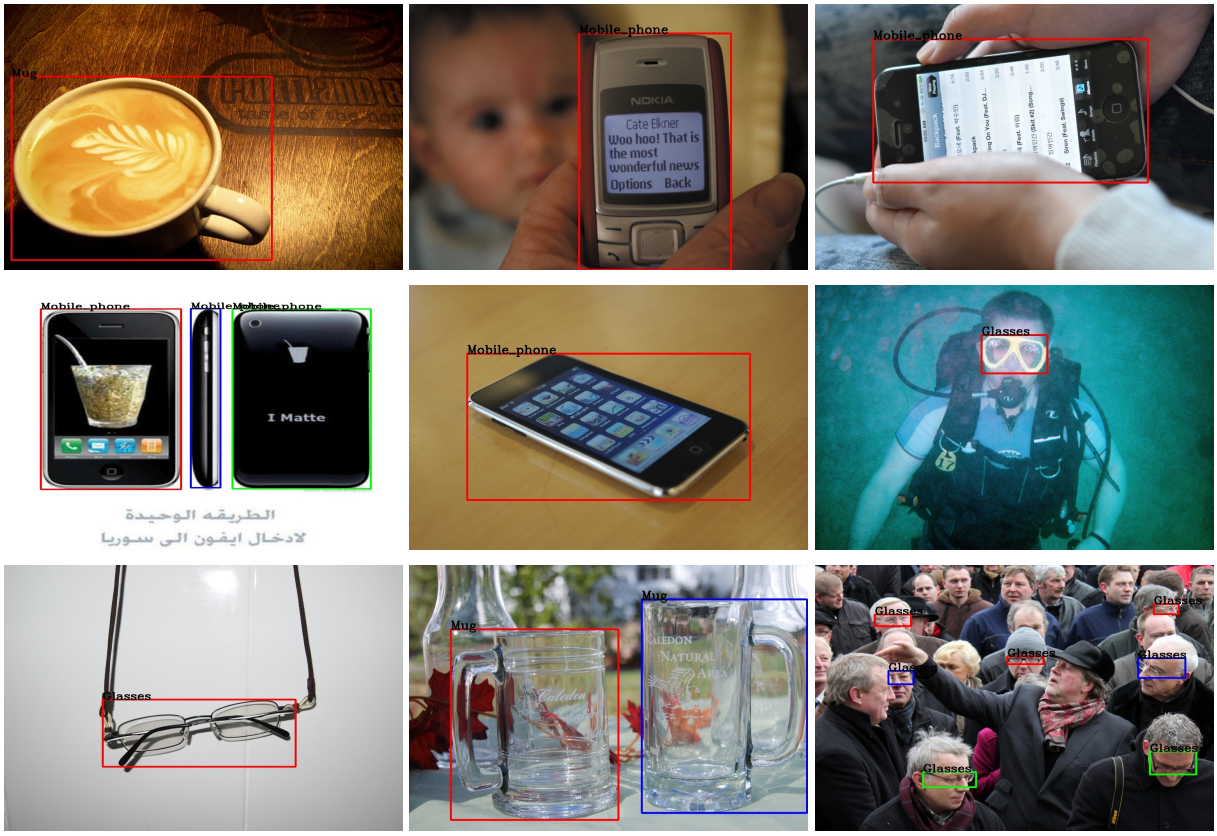


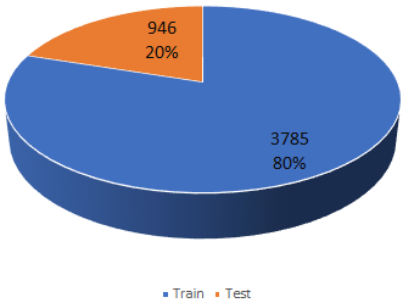
Figure 3.4: A sample of the images used to train the 4-class models with the respective ground-truth bounding boxes (Kuznetsova et al. 2018).

3.3.4 Models trained on 10 classes

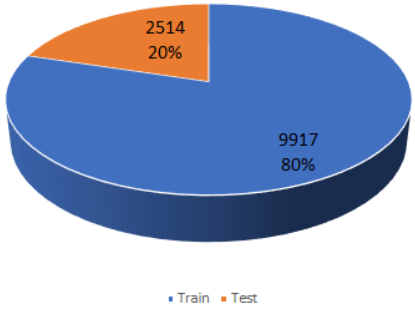
The final set of models for the first phase were trained on ten different classes. Following the criteria attended in Section 3.3.3, the six classes added were: *Book*, *Bottle*, *Coin*, *Headphones*, *Human Hand* and *Tin Can*. Despite not being an object found frequently on a table, the *Human Hand* class was also chosen because of the possible real-time in-frame movement of objects by an human subject.

The distribution of the images and bounding boxes used for training, comprised of all the data available for the classes in the Open Images Dataset is available in Figure 3.6. As can be seen, there is a big disparity in the availability of data for the different classes. That is, the dataset used is highly imbalanced, more specifically, there is a big class imbalance. Several works are dedicated to study the impact of imbalance and class imbalance [77–79]. The general conclusion is that class imbalance has a significantly negative impact on the performance of many object detection methods. To test such an hypothesis, the set was balanced, using two different approaches. First, images were randomly selected so that their number for each class in the train and test partition were about the same. Hence, the number of images and bounding boxes, as well the distribution of classes, for each class are exhibited in Figure 3.7. As can be observed, the aforementioned balancing doesn't produce a perfectly balanced set, seen as the number of bounding boxes for the *Book* class is far superior to the number of the rest of the classes. This happens because each image of the class usually entails a large number of bounding boxes, more so than any of the other objects. An example of this fact can be noticed in Figure 3.5. As is the case, for many of the images for this class, there are many instances of the object present, in this particular example 39. Having that in mind, a third dataset was produced. Images were randomly selected so that the number of bounding boxes for each class in the train and test partition was about the same. The information for this dataset can be visualize in Figure 3.8.

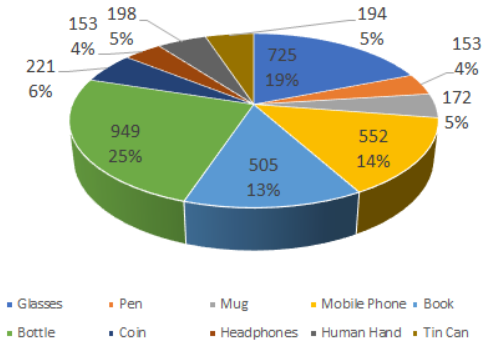
Figure 3.5: Example of an image annotated with the *Book* class (Kuznetsova et al. 2018).



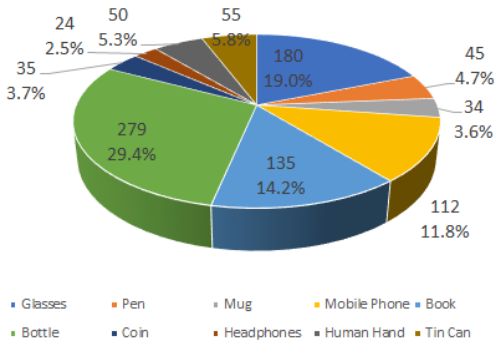
(a) Division of images into the train and test partitions.



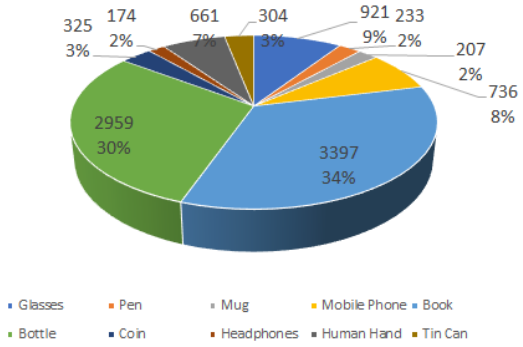
(b) Division of bounding boxes into the train and test partitions.



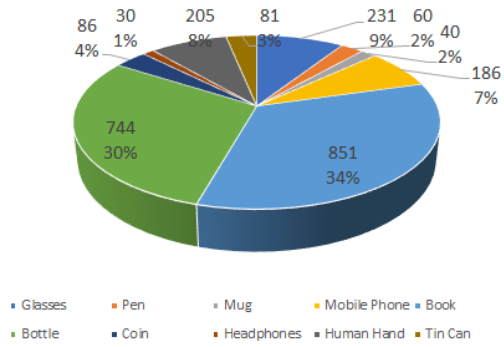
(c) Proportion of class images in the train partition.



(d) Proportion of class images in the test partition.



(e) Proportion of class bounding boxes in the train partition.

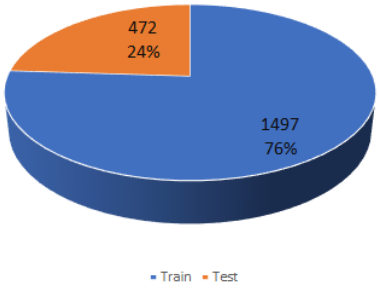


(f) Proportion of class bounding boxes in the test partition.

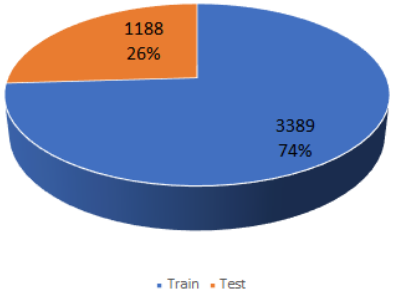
Figure 3.6: Statistics on the first set of images used to train the ten-classes models.

As was the case in Section 3.3.3, the batch size of the SSD model had to be reduced. The images used were also resized to half its size, using the same method as before.

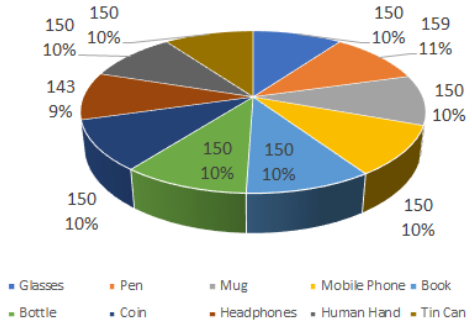
A sample of the images used during the training process can be seen in Figure 3.9. Several characteristics of the images and of the bounding boxes used are worth pointing out. As is to be expected, some images were annotated with bounding boxes of more than one class, however, it is possible to observe several instances where desired classes are present in the image and weren't annotated. This is very prevalent in the *Glasses* and *Human Hand* classes, categories that are present in many images and not always annotated. A further manual annotation of such examples could contribute to an improvement of the training set. There are also several bounding boxes that indicate a group of objects, that is, they are example of the Open Images Dataset attribute, *GroupOf*. For the environment in study, these may not be the best boxes for training the model. So, there are possibly some gains in eliminating



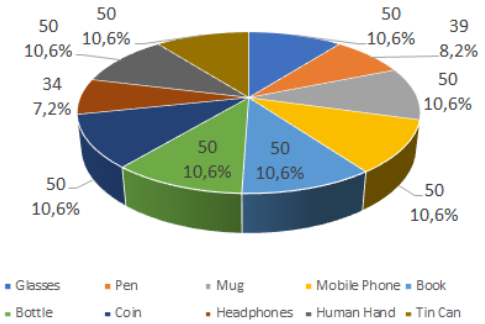
(a) Division of images into the train and test partitions.



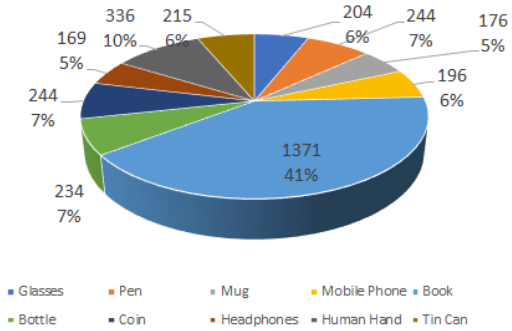
(b) Division of bounding boxes into the train and test partitions.



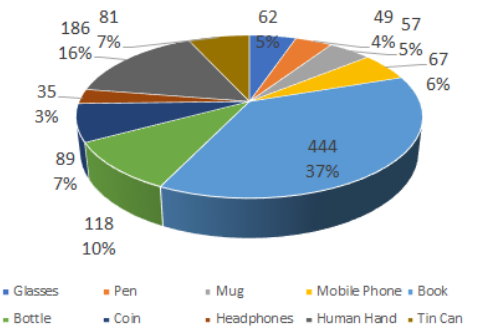
(c) Proportion of class images in the train partition.



(d) Proportion of class images in the test partition.



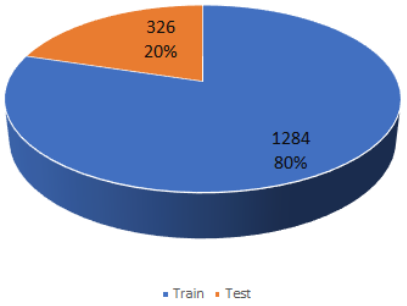
(e) Proportion of class bounding boxes in the train partition.



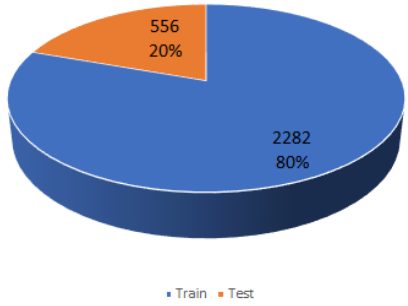
(f) Proportion of class bounding boxes in the test partition.

Figure 3.7: Statistics on the second set of images used to train the ten-classes models.

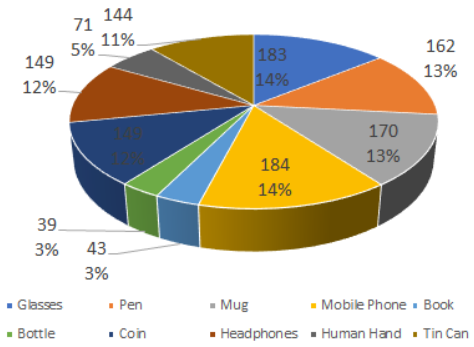
such example images.



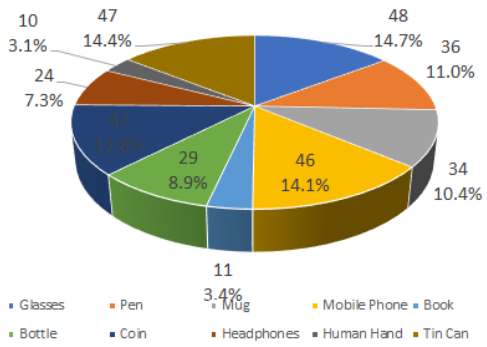
(a) Division of images into the train and test partitions.



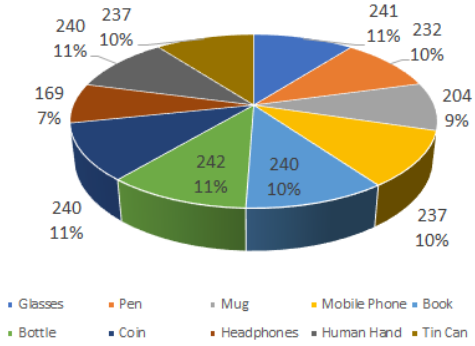
(b) Division of bounding boxes into the train and test partitions.



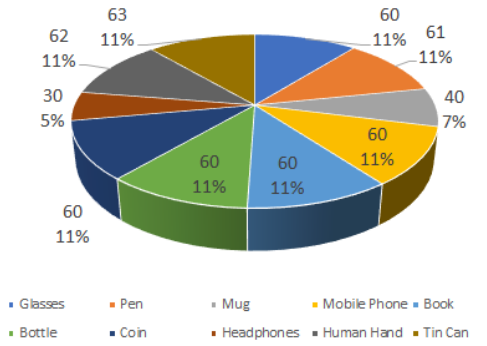
(c) Proportion of class images in the train partition.



(d) Proportion of class images in the test partition.



(e) Proportion of class bounding boxes in the train partition.



(f) Proportion of class bounding boxes in the test partition.

Figure 3.8: Statistics on the third set of images used to train the ten-classes models.

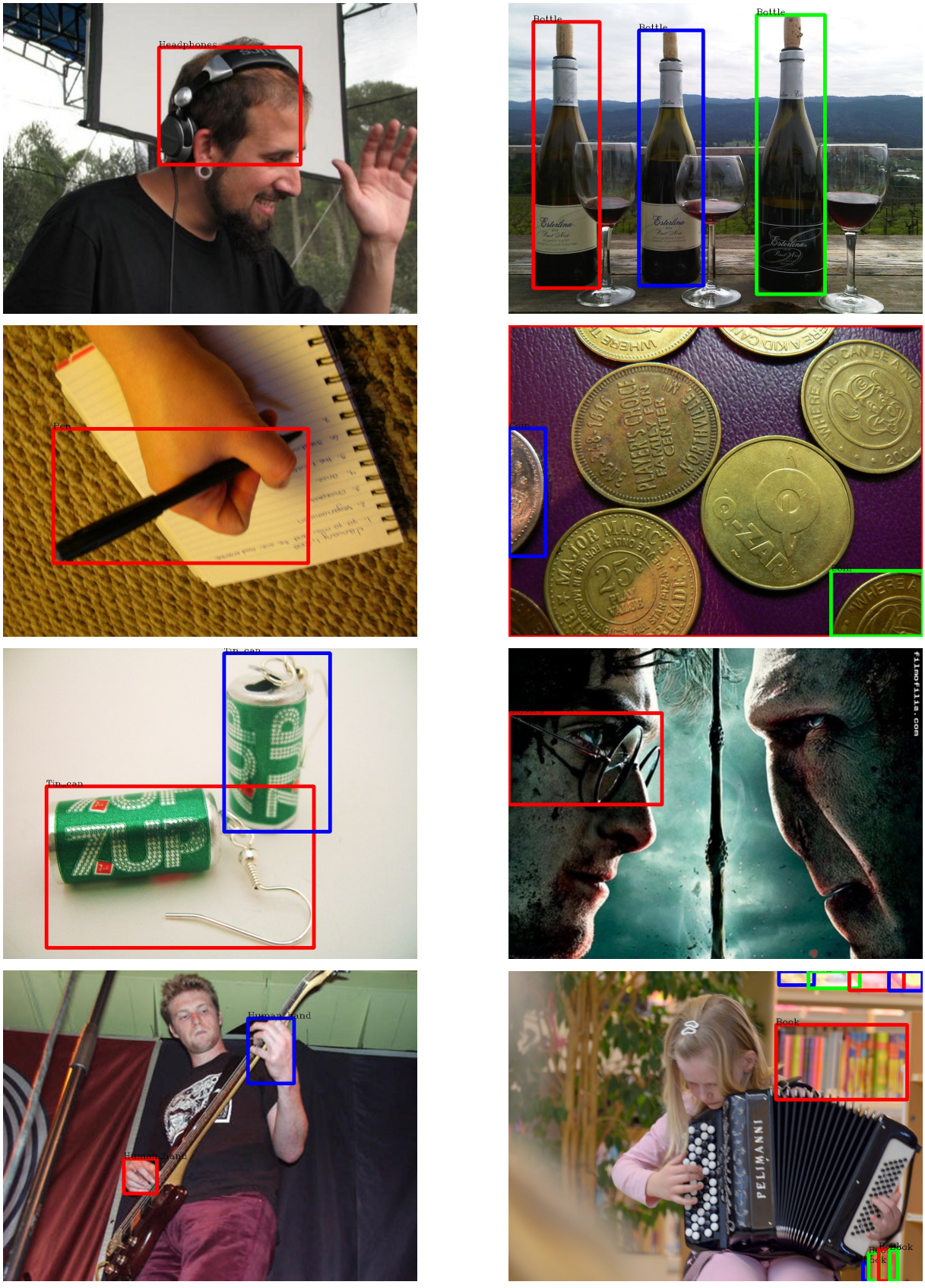


Figure 3.9: A sample of the images used to train the 10 classes models with the respective ground-truth bounding boxes (Kuznetsova et al. 2018).

3.4 Models trained with manually annotated images

There were several purposes in training models with manually annotated images. The first objective is to determine if it is possible to obtain a set of images and annotations, from a source other than a purposely curated database, good enough to train a workable model, seen as such databases are very rare and possess a relative small number of classes. The second purpose, related to the main objective of this work, is to reflect on and check if it would be possible for a program to be coded, that would allow training "at home", that is, the training of a model by an untrained end user of a possible product with the characteristics discussed in Section 1.3.

The choice of classes and general environment to be simulated were also object of consideration. The chosen classes were: *capacitor*, *potentiometer* and *resistor*. These point out to an electronics related environment such as an electronics workshop or an electronics laboratory. This decision greatly influences the workflow for the training of a model. For instance, the workflow for creating a database as the one in Section 3.3.1 becomes invalidated, seen as, there are not a significant number of images on *Flickr* for the described classes. Hence, these classes can be considered more "technical" or "bussiness related", in as much as they don't usually appear in general purpose images, as the ones shared on social networks. Thus, it is also important to test if such classes and such images can be used to construct good models.

The workflow of this phase can be encapsulated into three stages: obtaining and selecting images, annotating the images and training the models.

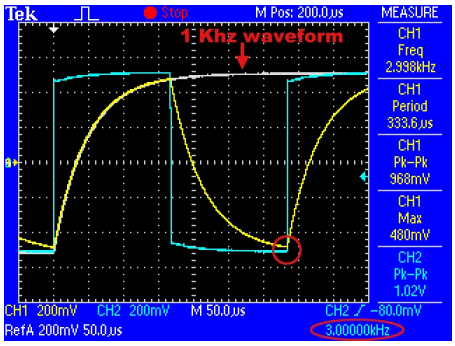
3.4.1 Obtaining and selecting images

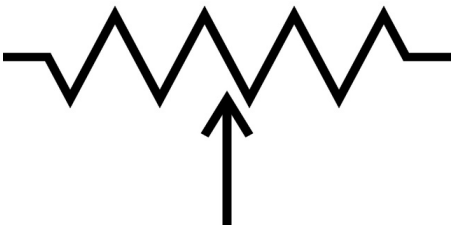
The images used in this phase were obtained through several query searches on *Google Images*. The queries used to procure the images were:

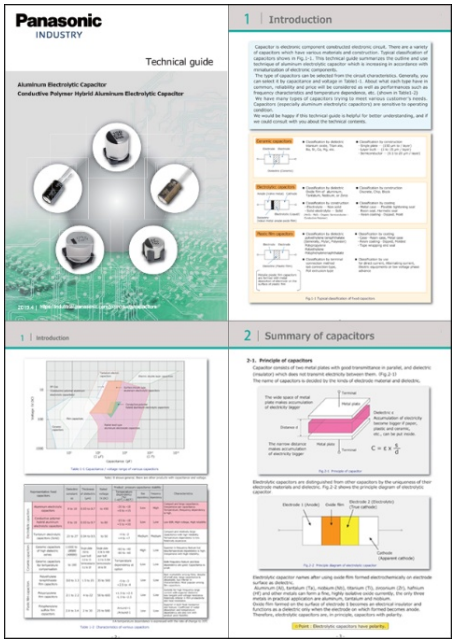
- **Capacitor:** *capacitor*, *capacitor breadboard*, *capacitor with background*, *ceramic capacitor* and *electrolytic capacitor*.
- **Potentiometer:** *potentiometer*, *potentiometer breadboard* and *potentiometer circuit board*.
- **Resistor:** *resistor*, *resistor background*, *resistor breadboard* and *resistor circuit board*.

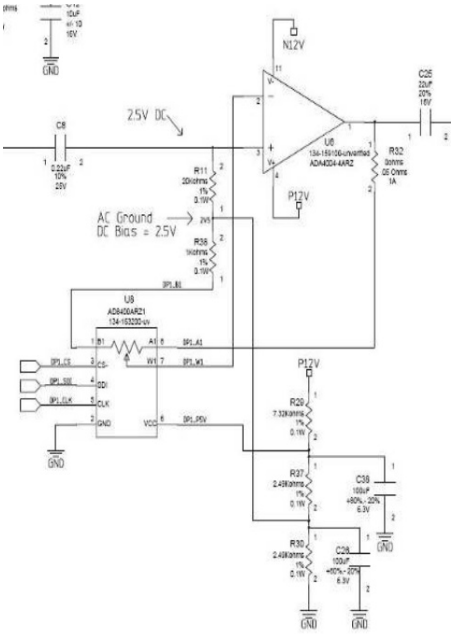
The images were then downloaded in bundle, that is, all the resulting images were downloaded, resulting in a total of 1175 images for the *capacitor* class, 1061 images for the *potentiometer* class and 1333 images for the *resistor* class.

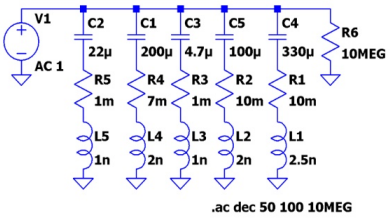
Next, the images had to be filtered, that is, a selection of the useful images had to be made. Images such as drawings, animations, schematics, graphics, pages from technical documentation, table and duplicate images were removed. Also removed, were images too strenuous to annotate, images where the objects were not visible and images that didn't correspond to the intended search. An example of the removed images can be seen in Figure 3.10. The resulting dataset from this process was comprised of 320 images for the *capacitor* class, 273 images for *potentiometer* and 313 for *resistor*.



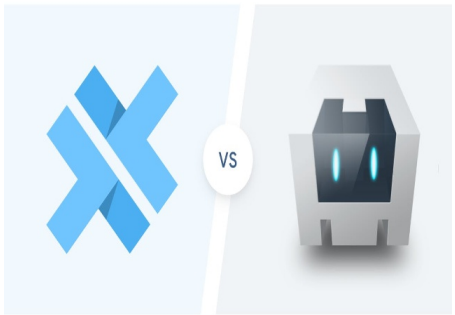








Series	HGX	HBX	PHV
Temperature range	-40℃ to +135℃	-40℃ to +125℃	-55℃ to +135℃
Lifetime	135℃/2000hrs, 3000hrs	125℃/3000hrs	135℃/2000hrs, 4000hrs
Voltage range	25V to 70V	250V to 290V	25V, 35V
Capacitance range	240μF to 6800μF	30μF to 56μF	47μF to 330μF
Size	12.5X20 to 18X30	10X30 to 12.5X30/16X20	6.3X6.1 to 10X10.5



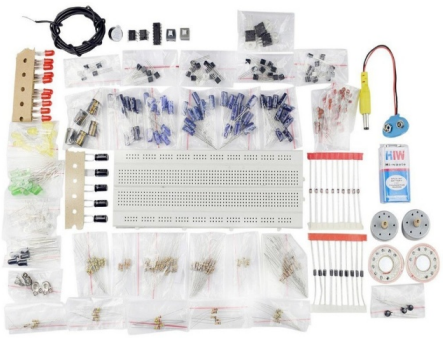


Figure 3.10: A sample of the images removed while selecting the appropriate images for training.

3.4.2 Annotating the images

The resulting dataset from Section 3.4.1 had to be annotated, that is, indicate the location of the objects in each image by means of bounding boxes. A program was produced, where the annotators indicated the top-left and bottom-right corners of the relevant objects in the image, using mouse clicks, and the classes of such objects, by the ways of keyboard selection. After a final review of the annotated objects and acceptance of the results, the program produced two annotations files accordingly to the specifications of the frameworks later used for training - a *.txt* file for YOLO training and a *.csv* file for the Tensorflow API. The workflow of the program can be seen in Figure 3.11

(a) Original Image.

(b) Class selected.

(c) Top-left corner selected.

(d) Bottom-right corner selected.

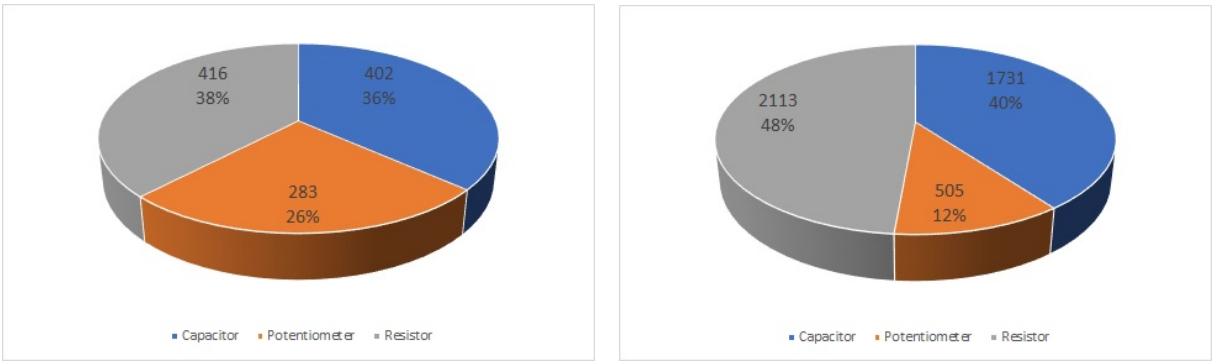
(e) Final result, ready to be confirmed.

Figure 3.11: Workflow of the program produced to annotate images.

As indicated by Kuznetsova et al. [75], a perfect box is the smallest possible box that contains all visible parts of the object. Although simple at first sight, this definition has a few caveats, the main of which is: "what is and isn't part of the object?". This question lead to the decision of not including the metal pin of capacitors and resistors as part of the objects, due to several facts:

- These are often not presented in its entirety in the collected images.
- Their composition and dimensions lead many times to a difficulty in their recognition. For instance, shiny metals often face a difficulty in identification related to light flares.
- Considering the general objective of this work relates to a pick and place robot, the pins are irrelevant to the task, seen as it is more difficult to pick the objects by the pins.

The features of the resulting dataset, with bounding boxes annotations, are detailed in Figure 3.12.



(a) Number of images that contain each of the classes.

(b) Number of bounding boxes for each class.

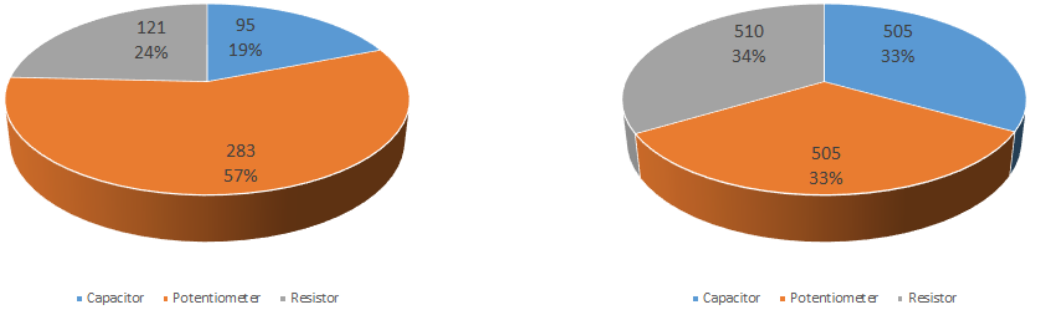
Figure 3.12: Features of the annotated dataset.

Finally, it is important to denote that the annotation process is quite strenuous. It took two annotators a combined time of about 25 hours to produce the annotations for the 906 images of the dataset. Seen as the total number of bounding boxes is 4349, the time spent corresponds to 20s per bounding box, which is quite acceptable. Hence, one can assume that any dataset with a sufficient size will take a considerable amount of resources to annotate.

3.4.3 Training the models

In order to further investigate the impact of class imbalance, the decision was made to use two different datasets. The first one was comprised of all the collected and selected images, with the characteristics detailed on Figure 3.12. The second one was a balanced dataset, where images were randomly selected so that the number of bounding boxes for each class was the same and equal to the maximum possible, 505. Hence, the second dataset possessed the characteristic detailed in Figure 3.13.

A sample of the images used and the corresponding labels can be seen in Figure 3.14. Analysing the images, it is possible to make some considerations:



(a) Number of images that contain each of the classes.

(b) Number of bounding boxes for each class.

Figure 3.13: Features of the annotated and balanced dataset.

- The items are specially focused in the images and occupy large portions of them. Considering that those items are very small in real life (around 1 cm for resistors, for example) this may not be the best representations for a real-world situation.
- The object are very few times presented in a natural ambient, that is, the images seem to have been computational enhanced and their backgrounds are most of the times artificial.
- The images have, in general, several different objects and they are located in the same regions. Although the two previous points are generally detrimental when training an object detector, this characteristic can be helpful, specially when detecting objects in cluttered environments.

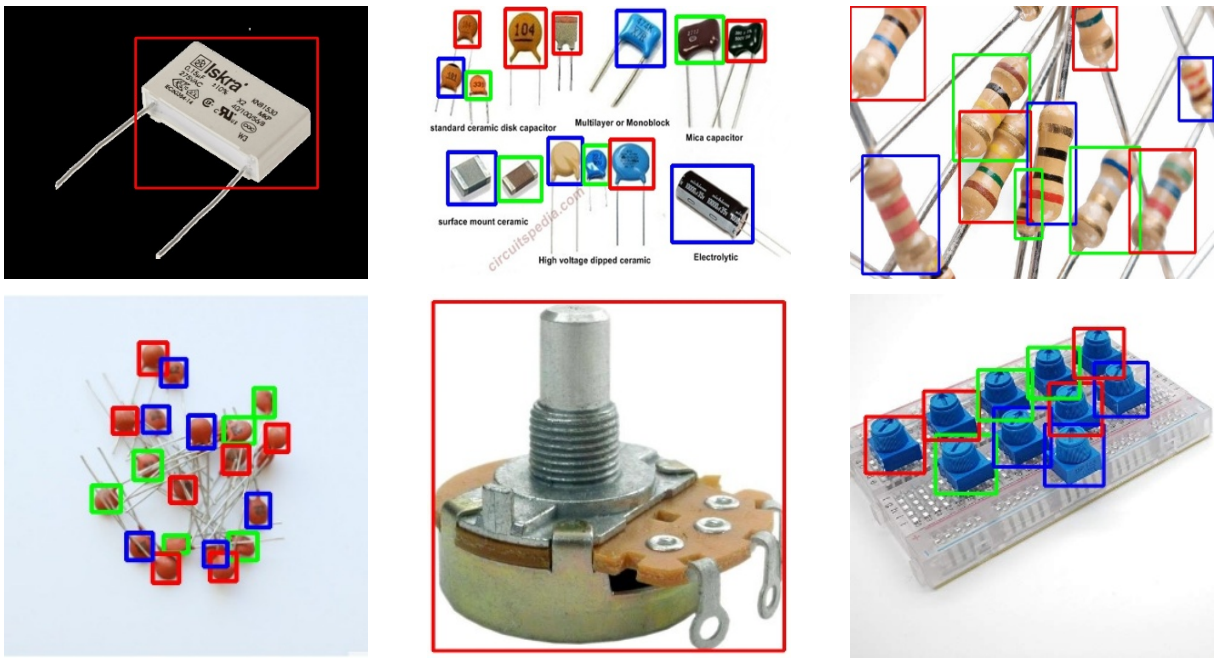


Figure 3.14: Manually annotated images used to train the models with the respective ground-truth bounding boxes.

Chapter 4

Results

Taking into consideration that the main goal of this work is to be able to perform real-time or near real-time object detection on video feeds, several sample videos were taken, in order to evaluate the resulting models. The videos were capture with a simple webcam of model SelecLine PPW-10, with resolution 640×480. The installation can be seen in Figure 4.1. The purpose of using such a device is to evaluate the possibility of adapting the trained models to a household item, readily available at a low price. These videos were implicitly different in factors such as:

- Illumination.
- Distance to the objects intended to be identified.
- Background color.
- The pose of the objects to be detected.
- The presence or absence of movement.
- The area of the frame where objects were present.
- Occlusion of the objects to be detected.

From these videos, several frames were then randomly selected. These were then manually annotated using the same method as in Section 3.4.2 in order to obtain the ground-truth bounding boxes. The comparison of such bounding boxes with the ones predicted by each model enables the computation of the mAP.

The other factor used in the evaluation of the models relates the the time needed for the model to make a prediction. The used metric was the rate of prediction in FPS, that is, the number of frames the model could evaluate in a second. In order to obtain the results for this metric, it was simply measured the time between predictions on the video feed of the webcam. It is important to denote that the best result possible for this metric is 30 FPS, seen as this is the rate of capture for the webcam feed.

In the following sections, we show the obtained results for each trained models, using such metrics and analyse some details of those results.

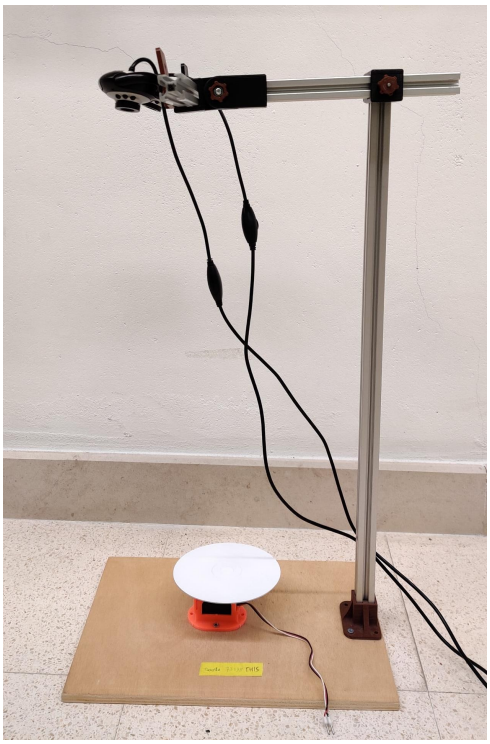


Figure 4.1: Webcam installation used for results evaluation.

4.1 Models Trained Using a Curated Dataset

4.1.1 Models trained on 1 class

The verification set obtained for these models, using the process described above, was composed by 89 images with 141 ground-truth boxes for the *Pen* class. The obtained results for the models trained in this section can be seen on Table 4.1.

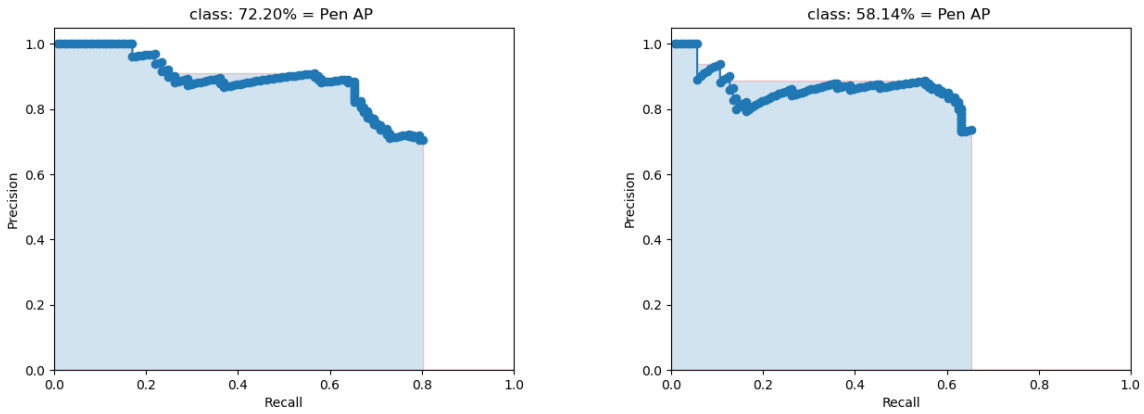
Model	mAP(%)	Rate (FPS)
Faster R-CNN	**72.20**	6
SSD	12.58	**23**
YOLO	58.14	**23**

Table 4.1: Results for the models trained on 1 class.

The first conclusions that can be extracted from the analysis of Table 4.1 are:

- The Faster R-CNN model performs the best in terms of accuracy while being significantly slower when predicting the results.
- The SSD model matches the YOLO model has the fastest model but performs very poorly in terms of accuracy.
- The YOLO model performs very well in terms of speed but lags behind the R-CNN model when accuracy is concerned.

However there are some insights very important to compare the Faster R-CNN and YOLO models that aren't visible in Table 4.1. In order to have a deeper understanding of the performance of both models we have to look at their precision-recall curves. These are shown in Figure 4.2.



(a) R-CNN Model

(b) YOLO Model

Figure 4.2: Precision-recall curves of the R-CNN and YOLO models trained on 1 class.

For this test, it is possible to observe that the R-CNN model has a bigger trade-off between the precision and recall numbers, that is, although the final value of recall is higher than the one obtained for the YOLO model, it comes at the expense of precision, which is lower than in (b). This becomes clear when analyzing the true and false positive numbers available in Figure 4.3. Depending on the final application of the model, that is, if it is more important to have the highest precision or the highest recall, the choice of model can fall on the second model or, preferably, the first model but with a higher confidence threshold for prediction.

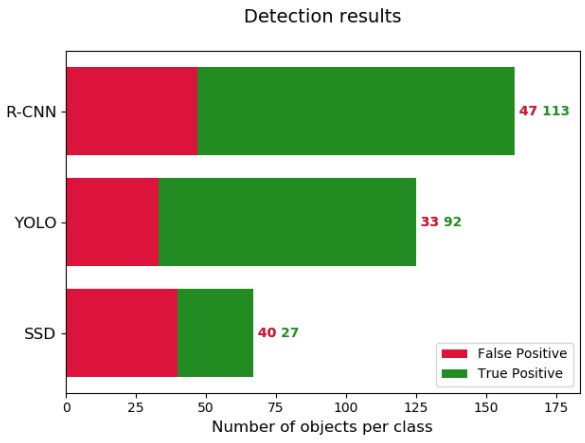


Figure 4.3: Number of true positives and false positives for the models trained on 1 class.

Figure 4.4 encompasses a deeper analysis of the R-CNN model as it is the best performing model in terms of accuracy. The ground-truth boxes are marked in blue or pink, respectively, if they have a correspondent prediction or not. Equally, the prediction bounding boxes are green or red, if they are classified as true positives or false positives.

- It is visible that the model performs worse when luminosity is low. That is evidenced by (a) as opposed to (b). Despite that the model seems to be resistant to variations in luminosity to a certain degree.
- It is also apparent that the model has better results when the objects are closer to the camera, as demonstrated by (c) as opposed to (d).
- The model is resistant to occlusion and movement to a certain degree, as can be seen in (e) and (f). Even (g) can be seen as a proof of good result, as far as occlusion is concerned. Despite the prediction being classified as a false positive, as its IOU with the ground-truth bounding box is below 0.5, it is true that it corresponds to part of a pen.
- The region of the image where the object appears doesn't seem to affect the model. In spite of that, when an object is encountered in the edge of the image and that is combined with other adversarial factors, such as occlusion (h) or a greater distance to the objects (i), the model seems to behave worse.
- Items similar to pens, such as a test light screwdriver (d,i,j), seem to easily fool the model.

Hence, despite some drawbacks, the R-CNN model seems to produce good results, close to the state-of-the-art results given in Table 2.1.

4.1.2 Models trained on 4 classes

The composition of the verification set can be seen in Figure 4.5. None of the images used in Section 4.1.1 was used in this section. The different proportion of bounding boxes between classes results from the real situation in study. Specifically, in the real world, it is expectable that the occurrence of the class *Pen* exceeds the occurrence of the other classes and so on. The obtained results for the models of this section can be seen in Table 4.2.

Model	mAP(%)	Rate (FPS)
Faster R-CNN	**59.36**	6
SSD	16.79	**20**
YOLO	45.99	**20**

Table 4.2: Results for the models trained on 4 classes.

The conclusions that can be drawn from Table 4.2 are:

- The accuracy values tend to decrease from the models trained on 1 class, indicating the models perform worse when detecting the classes that aren't *Pen*.
- For the R-CNN model the speed doesn't seem to be affected when the total number of classes the model can detect is increased, while for the other models the speed is slightly lower than before.
- The Faster R-CNN model continues to be the more accurate, while the others continue to lead on speed.

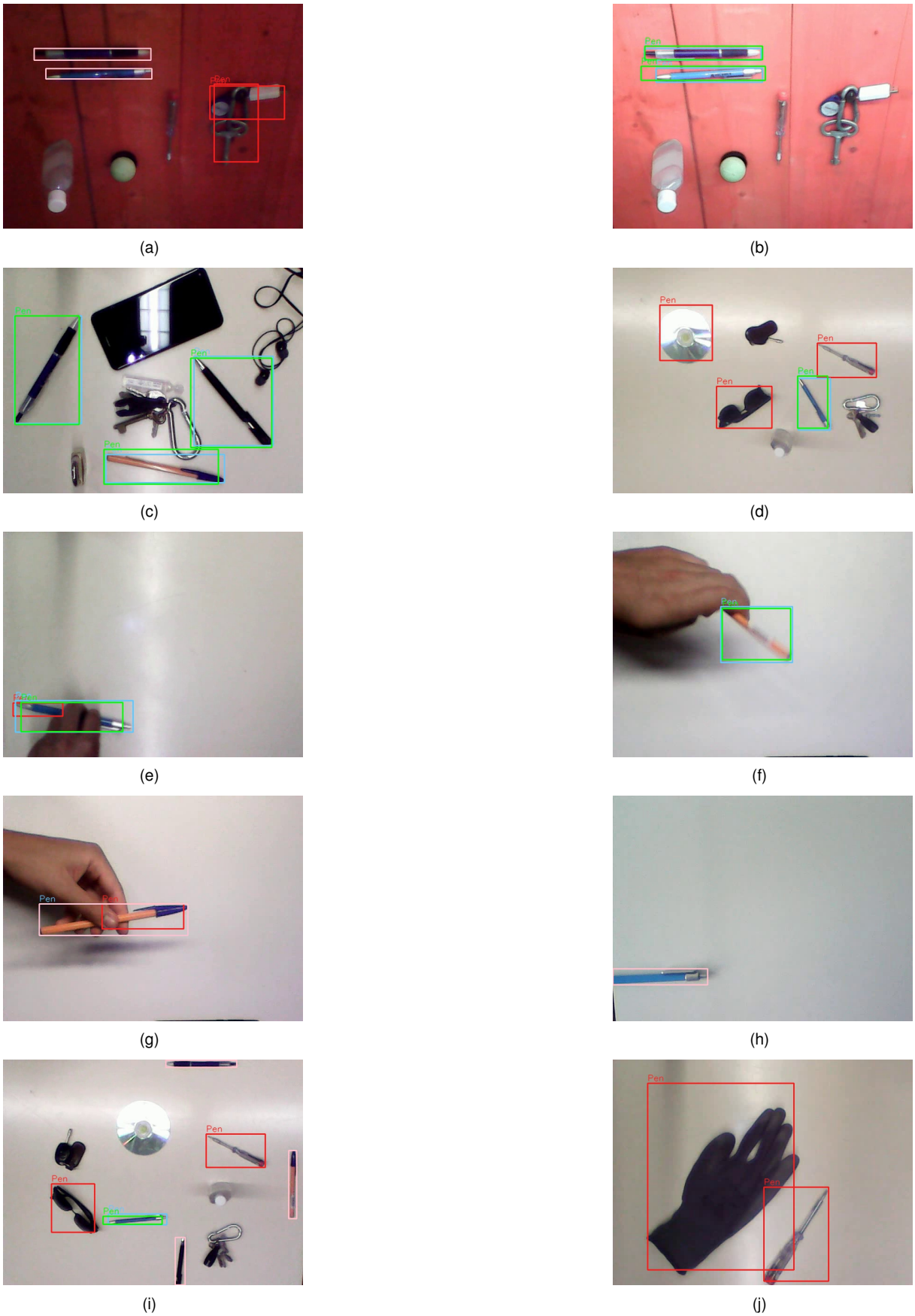


Figure 4.4: Predictions of the R-CNN model trained on 1 class.

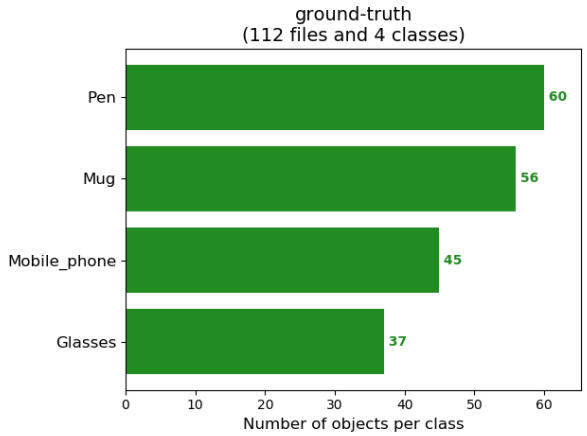
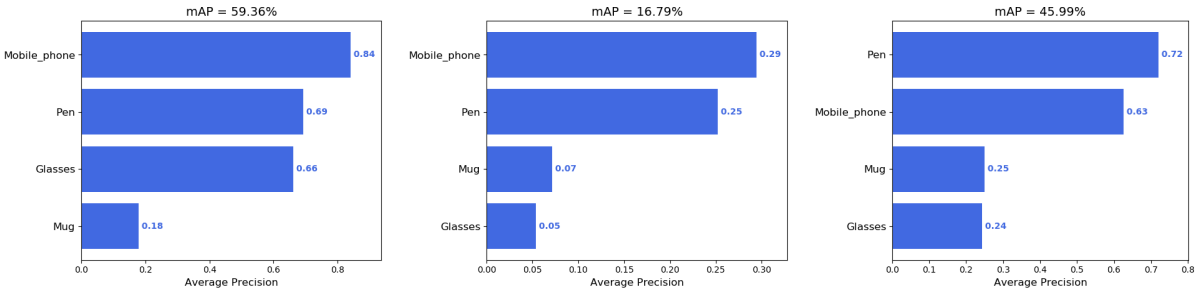


Figure 4.5: Verification set composition for the 4 classes models.

Delving deeper into the computation of mAP for each model, we obtain Figure 4.6. The results for the AP of the *Pen* class are different from the ones in Section 4.1.1 because of two factors: the verification sets are different and the models that make the predictions are also different. It is possible to see in Figure 4.6 that the presence of the *Mug* class greatly decreases the value of mAP for the models. In fact, if the class was to be disregarded, the value of mAP for the R-CNN model would be equal to 73.19%, higher than the one obtained on Section 4.1.1. The same happens with the other models. If the 2 worst performing classes were to be disregarded for the YOLO model, its mAP would be 67.32%.



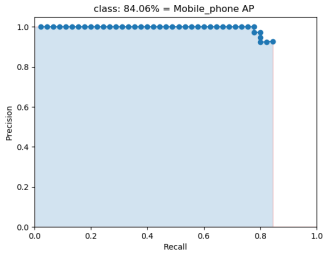
(a) R-CNN Model (b) SSD Model (c) YOLO Model

Figure 4.6: Comparison of the AP by class for the models trained on 4 classes.

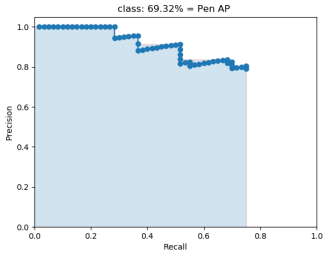
Having a deeper look into the R-CNN model, it is important to take into consideration the precision-recall curve and the absolute values of predictions. This information is present in Figure 4.7. Some information that can be withdrawn from the figure is:

- The *Mobile phone* class has a high value of value of AP due to a consistently high value of precision.
- The *Pen* class, despite worse and the previous one, still presents good results.
- The *Glasses* class has a big trade-off between precision and recall. Depending on the objective of application or the preference of the user the confidence threshold value should be adjusted to have higher precision or higher recall.

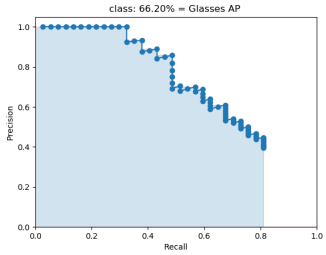
- Despite poor results, when analysing mAP, the *Mug* class has high values for precision, which means that each time the model predicts there is a mug in the image, it is highly probable the prediction is true, despite not many mugs being identified. A possible cause for this is that, in the verification set, the mugs used were white and, many times, in a white background. This can be seen in Appendix A. Furthermore, one of those was of small dimensions and seems to be predicted less times than the other. This may indicate the model learned certain dimensions for the *Mug* class that aren't compatible with the first mug.



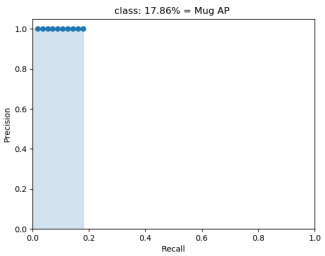
(a) PR curve for *Mobile Phone* class



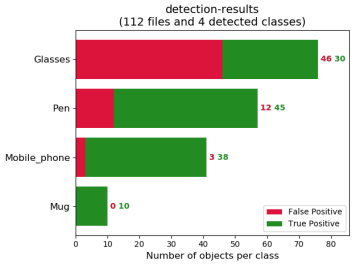
(b) PR curve for *Pen* class



(c) PR curve for Glasses class



(d) PR curve for *Mug* class



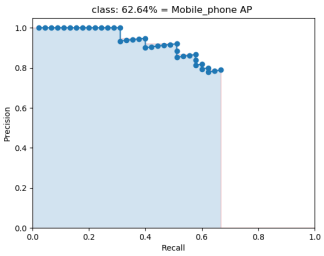
(e) Absolute values for predictions

Figure 4.7: Precision-recall curves and predictions distribution for the R-CNN model trained on 4 Classes.

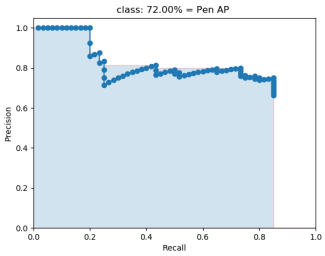
The same information is shown in Figure 4.8 for the YOLO model. Some points that are apparent in the figure are:

- The *Mobile phone* class performs worse than in the R-CNN model. It has a bigger trade-off between precision and recall.
- The *Pen* class performs better than in the R-CNN model, but the final value of precision is worse, evidenced by the greater number of false positives. The confidence threshold value for this class should be higher.
- The *Mug* and *Glasses* have low values of mAP, but high precision values. For the *Mug* class the explanation may be the same as the given above. For the *Glasses* class, the model only seems to detect it when the object are on a certain position and at a certain distance, as can be seen in Appendix A.

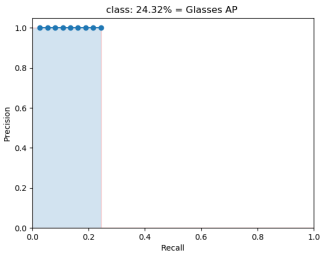
Taking the previous information into consideration, it is visible the R-CNN model is again the best performing model and produces good results, specially if we exclude the worst performing class.



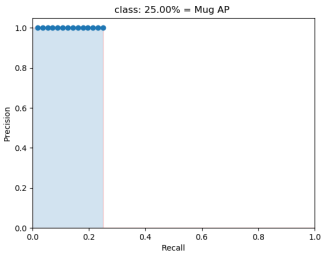
(a) PR curve for *Mobile Phone* class



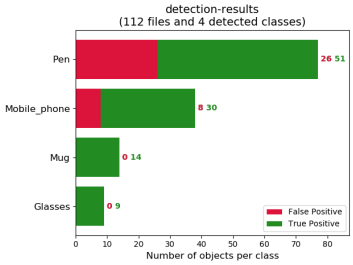
(b) PR curve for *Pen* class



(c) PR curve for Glasses class



(d) PR curve for *Mug* class



(e) Absolute values for predictions

Figure 4.8: Precision-recall curves and predictions distribution for the YOLO model trained on 4 Classes.

4.1.3 Models trained on 10 classes

The composition of the verification set used for this phase can be seen in Figure 4.9. Equally, the obtained results for the models can be seen in Table 4.3. The references (a), (b) and (c) correspond, respectively, to the models trained with an imbalanced dataset, a dataset balanced in the number of images and a dataset balanced in the number of bounding boxes.

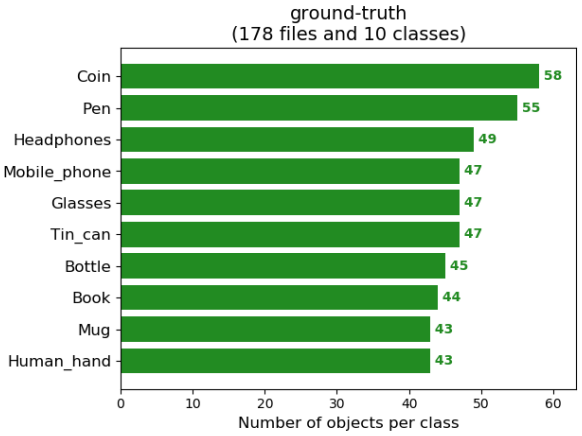


Figure 4.9: Verification set composition for the 10 classes models.

The increase in the number of classes to detect seems to have an effect on the accuracy of the models. Analyzing the detections, the problem looks to be located on images with cluttered environments, that is, with several objects in an image, such as Figure 4.10. In this kind of image, the models seem to correctly predict one or two classes, but fail to make predictions on the remaining objects of the images. This appears to be confirmed by the accuracy results when the verification set is reduced to images with 1 or 2 objects. These results are shown in Table 4.4. The table clearly shows the models perform much

Model	mAP(%)			Rate (FPS)
	(a)	(b)	(c)	
Faster R-CNN	**38.44**	33.66	20.99	6
SSD	11.00	6.83	4.88	19
YOLO	26.68	18.50	22.50	**20**

Table 4.3: Results for the models trained on 10 classes.

better with less objects to predict per image. In fact, except for the SSD model with training dataset (b), all the models have better results in this verification set, with the best performing ones approaching the performance of the models of Section 4.1.2. As was previously referred in Section 3.3.4, the dataset used for training of the models usually focus each image for a specific class, which may be an explanation for this kind of behaviour.

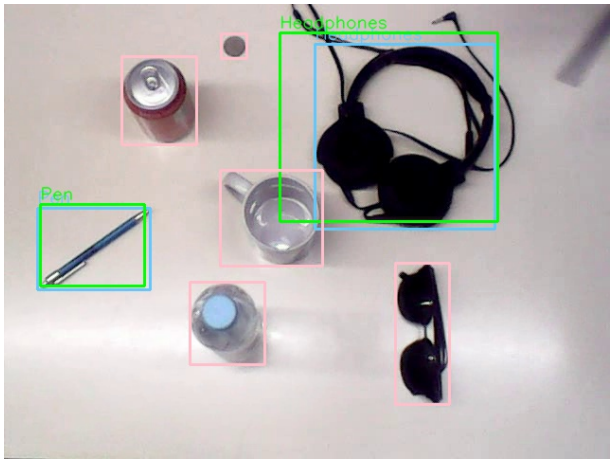


Figure 4.10: Example of image with cluttered environment.

Model	mAP(%)		
	(a)	(b)	(c)
Faster R-CNN	**58.54**	47.25	30.50
SSD	18.03	4.37	7.37
YOLO	44.32	19.34	25.81

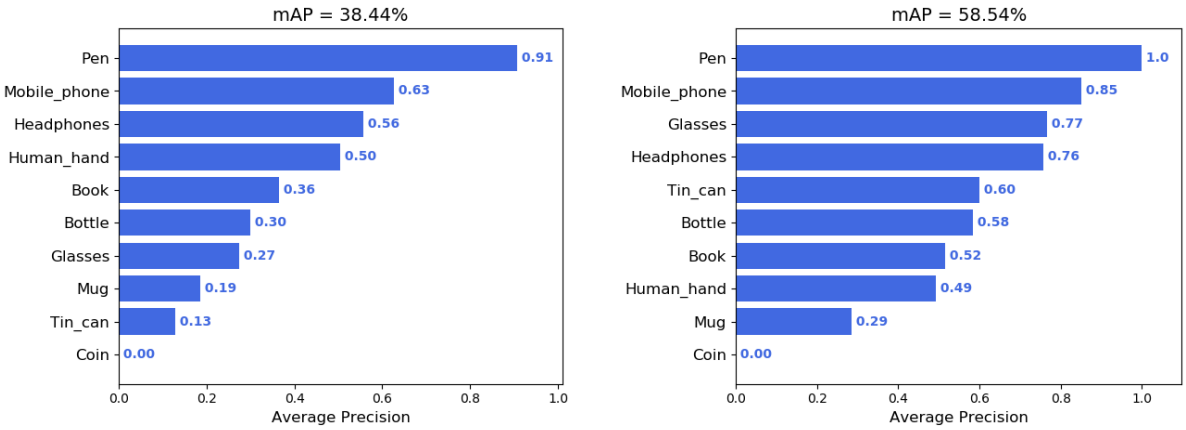
Table 4.4: Results for the models trained on 10 classes using images with 2 objects or less.

In what concerns speed, the SSD model seems to be somewhat affected with the increase of the number of classes, while the other methods keep approximately the values from previous sections. This factor, allied with the fact that YOLO has an higher mAP, seems to suggest that, when speed of prediction is an important factor, the YOLO algorithm has an edge over the SSD models.

Another consideration that is possible to take for the results relates to the balancing of the training dataset. It is quite clearly that the efforts made in that area didn't produce good results. Indeed, every model performs worse when trained with the balanced dataset than with the original dataset. However, the effect of the balancing isn't straightforward. Actually, some classes present an increase in AP in

consequence of the balancing. For example, the *Coin* class has an AP of 16.83% for the Faster R-CNN model trained with dataset (b), compared to the 0% AP obtained with dataset (a). Similarly, the *Headphones* class has an AP of 35.37% for the YOLO model trained with dataset (b) and of 20.49% when dataset (a) is used. Although, this effect isn't equal for all the classes, the classes with less examples in dataset (a) seem to be benefited by balancing while the ones with the most examples seem to be hindered.

Exploring the best performing model, the Faster R-CNN model trained with dataset (a), Figure 4.11 shows the AP for each class. As is the case in Section 4.1.2, the models struggles with the *Mug* class. Additionally, the *Glasses* class also performs worse in this phase. However, it is important to point out that the AP of this class greatly increases when the verification set is comprised of images with 2 or less objects. In such conditions the AP for this class is 76.75%. It is quite clear that the *Coin* gravely impacts the overall performance of the model. The reason for this problem may be in the fact that the images for this class in the training set are quite close to the target, making the objects rather big, as can be seen in Figure 3.9. Oppositely, the images in the verification set make the objects seem relatively small. If we exclude such a class, the obtained mAP would be 42.07% for the entire verification dataset and 65.05% for the smaller verification dataset. If we go further and eliminate also the influence of the *Mug* class, the value for the second verification set would be 69.61%. Although not state-of-the-art, this value approaches the ones reported by Zhao et al. in Table 2.1.



(a) Entire verification set

(b) Verification set with 2 object or less per image

Figure 4.11: Average precision by class for the Faster R-CNN model trained with dataset (a).

4.2 Models trained with manually annotated images

As stated in Section 3.4.3 two dataset - balanced and imbalanced - were produced for training models in this phase of the work. The general results obtained for the consequently 6 models can be seen in Table 4.5. Furthermore the composition of the verification set by class can be seen in Figure 4.12

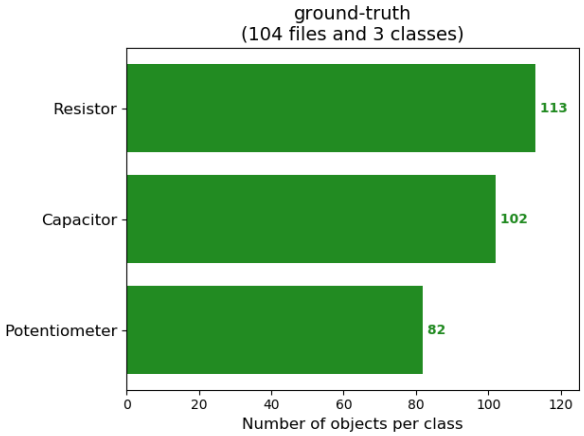


Figure 4.12: Verification set composition for the 3 classes models.

Model	mAP(%)		Rate (FPS)
	Balanced	Imbalanced	
Faster R-CNN	36.83	**37.50**	6
SSD	11.01	4.68	**20**
YOLO	19.52	24.69	**20**

Table 4.5: Results for the models trained with manually annotated images.

Some indications that can be withdrawn from Table 4.5 are:

- The Faster R-CNN model continues to be the best performing model in terms of accuracy, while the others have an edge when speed is concerned.
- The balancing of the dataset seems to produce either virtually no difference or a negative one, when talking about accuracy.
- The performance of the models, in terms of accuracy, greatly decreases from the models of Section 4.1 and from the results reported for each algorithms in the literature.

The last statement represents an important remark for this work. The dip in performance may be due to several factors. First, it was expected that the images used in this phase were not as appropriate for the task of object detection as images from a curated dataset. Some of the reasons for that are stated in Section 3.4.3. Second, the size of the objects in the verification set is quite small. This leads to several problems:

- There is a big difference between the images on the dataset used for training and the images on the verification set. As referred in Section 3.4.3, the images used for training are specially focused

in the objects in questions and, for that reason, very different from a real-world situation we can imagine. Possibly, if the images used for training were of simulated real scenarios, taken purposely for this application, the results would be better. However this bring a application problem for the product in analysis: it is impossible to simulate all the scenarios where a commercial available product can be used and, as such, if the data used for training can't generalize to most situations, the product is rendered useless.

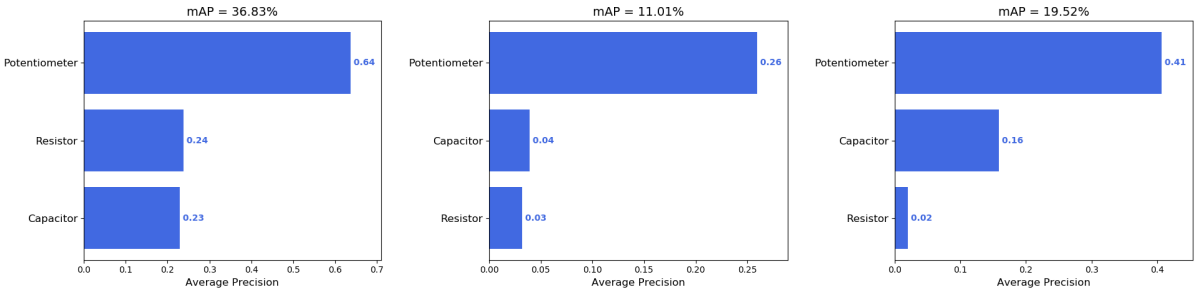
- There is a bigger influence of the IOU criteria used for the mAP computation, that is, seen as the objects are very small in the image, a small deviation in pixels of the predicted bounding box can mean a great deviation in IOU. Although this is a factor, its influence seems to be very limited because, when we lower the requirement of IOU for a prediction to be considered true to 30%, the improvements in mAP are marginal, as can be seen in Table 4.6.

Model	$mAP^{IOU=.30}$(%)	
	Balanced	Imbalanced
Faster R-CNN	38.41	**38.47**
SSD	11.01	4.68
YOLO	19.84	25.20

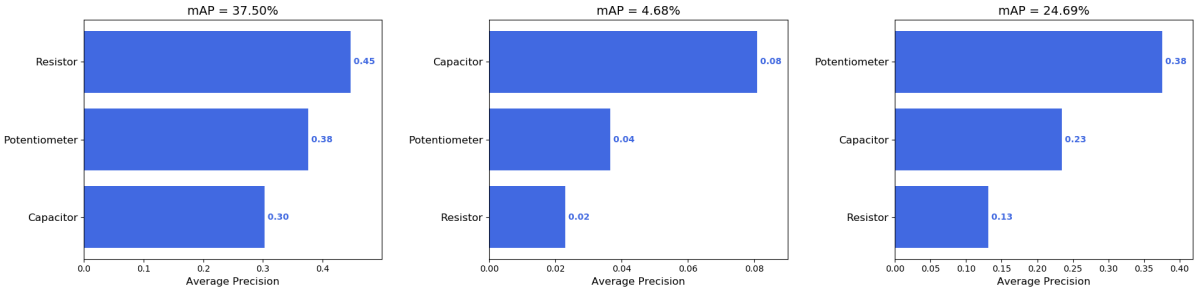
Table 4.6: mAP for the models trained with manually annotated images with IOU=30%.

When analysing the results in further detailed, it is possible to see that the previous statement regarding the influence of balancing a dataset can be misleading. In fact, it is possible to see in Figure 4.13 that the balancing the dataset greatly improves the performance of the detection of the class that has fewer examples in the imbalanced dataset, the *Potentiometer* class, across all trained models. This may indicate that reducing the number of example from other classes isn't the best approach and that balancing can be beneficial using, however, other techniques.

Finally, it is important to point out that the biggest problem of the models trained in this Section is the fact that there is a very low number of predictions for each class, that is, although precision for each model is relatively high, recall is very low. This seems to be confirmed by Figure 4.14. Comparing with Figure 4.12, it is possible to see that the number of prediction made by the models is very low when comparing with the number of examples in the set. In fact, except for one class in one of the models (the *Potentiometer* class in the Faster R-CNN model trained with the balanced dataset), no model is capable of accurately predicting even 50% of one class. This seems to be related to the point argued before concerning the distinction between the training and the verification sets.



(a) R-CNN model with balanced dataset (b) SSD model with balanced dataset (c) YOLO model with balanced dataset



(d) R-CNN model with imbalanced dataset (e) SSD model with imbalanced dataset (f) YOLO model with imbalanced dataset

Figure 4.13: Comparison of the AP by class for the models trained with manually annotated images.

(a) R-CNN model with balanced dataset (b) SSD model with balanced dataset (c) YOLO model with balanced dataset

(d) R-CNN model with imbalanced dataset (e) SSD model with imbalanced dataset (f) YOLO model with imbalanced dataset

Figure 4.14: True positive and false positive prediction for each model.

Chapter 5

Conclusions

Regarding the main general objective, the study of the input sensory system of a tabletop pick and place robot, we can see from the work developed that it is possible to devise a general system capable of recognizing to a good extend everyday objects and possible to be applied to commercial use. Comparing with other solutions published [80, 81], the accuracy metric on those solutions is better. However, such solutions, and by consequence their verification sets, are designed for very limited scenarios. In this work, the purpose of the product is more general and, despite lower accuracy, the solution found can be satisfactory. However, it is fairly clear that such a product has several limitation and should be design with those limitations in mind.

The first of the constraints to be considered is the fact that it is pretty explicit that some classes of objects are more suitable to this purpose than others. For instance, small objects seem to be harder to identify and locate than others of superior dimensions. On the other hand, if the objective of the system was to recognize smaller objects, it could be design such that the objects to be detected could be closer to the sensor, that is, such that they would better resemble the training examples used. The contrary could also lead to an improvement in performance, namely, the training examples could be chosen so that they better resemble the final objective images. This means the training of the image recognition algorithm can't be indiscriminate and dissociated from the objective of the system to be projected.

The second constraint relates to the detection of objects in cluttered environments. In such environments, the algorithms suffer a dip in accuracy performance, which means that any device using these algorithms should be design having that in consideration. That is, it should avert situations where the objects are closely together. Alternatively, it could also take advantage from the fact that, despite having a dip in performance, the detector seems to still predict correctly some of the object in the image. For instance, in a pick and place robot, it is possible to imagine a situation that the detector rightly predicts objects that are then removed from the working environment, decluttering it and improving the detector performance in the next moment of detection.

The third consideration necessary is the fact that the best performing algorithms are resistant to the majority of the hindering factors referred in the beginning of Chapter 4, but not to all. Low illumination can sometimes decrease the performance of the algorithms. Further study is necessary to determine

how much it contributes to such decrease. The distance to the objects to be identified also can be a problem, but seems to only affect small objects as previously discussed. Background color also appears to be a factor with importance but its effects were only seen in limited cases.

Relating to the specific situations studied in this work, it can be concluded that the choice of algorithm to perform object detection should fall onto Faster R-CNN when accuracy is the top requirement for a situation. On the other hand, if speed is preferred, the YOLO algorithm should be favored. Regarding the SSD algorithm, the results obtained are very distant, in terms of accuracy, from the ones reported by the authors and by other sources. This could mean that a series of blunders may have affected the training of such models, affecting the results obtained.

Another conclusion regarding the implementation of this work concerns the balancing of the training datasets. It is possible to comprehend that the balancing of the datasets in this work wasn't successful and, in fact, lowered the performance of the models. The most probable cause of such a situation was the fact that the method used for the balancing, subsampling, may not be the most appropriate for this task. Other methods such as oversampling, data augmentation or SMOTE [82] can be used to test the effects of data unbalancing and understand if it is possible to improve performance.

Regarding the proposition stated in Section 3.4, regarding the possible annotation of images by an end user, it may be fair to say that such a process would be very difficult to implement. That is justified by two arguments. The biggest hurdle to such a process would be the difficulties that would be presented to an untrained annotator. As is referred in Section 3.4.2, the annotating process is very time consuming and tedious. Furthermore, it is abundantly explicit throughout this work that some images are more appropriate to train a model than other. As an end user wouldn't be alert to this fact, there is no guarantee that he images chosen for training would be suitable. Indeed, the same argument is valid for the quality of the annotations, of which there also isn't any guarantee. However, it is possible to imagine that a trained group of people could expand the capabilities of products already in use, accompanying the evolution of datasets such as the Google Open Images Dataset or otherwise creating new and purposely built datasets.

Finally, it is important to mention that the models studied in this work can also be applied to many other situations. This unspecialization and possibility for general use is one the best feature of the object detection algorithms studied. In light of this, the proven feasibility of the use of this technology allows one to imagine several applications of it. It is possible to imagine the use of the technology in devices that would collaborate with an human in an work platform, a electronics stand, an instrumentation booth, a kitchen, and in inconceivable amount of other applications.

5.1 Future Work

In the future, it is essential to increase the capacity of the implemented models by expanding the database used for training and the number of classes they are able to recognize. It is also important, the study of future or recently published algorithms for object detection such as YOLOv4 [83], YOLOv5 or End-to-End Object Detection with Transformers [84]. Equally, it should be important to accompany

developments in the databases with object detections tasks. The Open Images Dataset was analysed in this work using its fourth version. It is now already in its sixth and continues to grow. Regarding the applications related to pick and place robots, it would also be important to add information on the depth aspect of images. Using cameras with such capabilities, it may be important for this task to perform object detection on RGB-D images, reviewing the works already published by some authors [85–87]. Regarding the device suggested in this work, it would be beneficial to evaluate the possibility of implementing the studied algorithms in an embedded system or a single board computer such as a Raspberry Pi or the more specialized NVIDIA Jetson, Google Coral or Khadas VIM series. In the future, there are also several possibilities for the operation of the suggested device. For instance, a process can be devise to “deconstruct scenes”, that is, withdrawn from the working space the recognized objects, leaving just the unknown ones. From there, a semi-automatic process of training could be constructed, where a model would provide the location information, maybe following recent works in class agnostic object detection [88], and the user would provide the name for the class.

Bibliography

[1] K. Čapek. *R.U.R. - Rossum's Universal Robots*. Aventinum, 1920.

[2] A. Bauer, D. Wollher, and M. Buss. Human-robot collaboration: a survey. *International Journal of Humanoid Robotics*, Dec. 2007.

[3] B. J. Grosz. Collaborative systems: AAAI–94 presidential address. *AI Magazine*, 17(2):67–85, 1996.

[4] J. E. Colgate, J. Edward, M. A. Peshkin, and W. Wannasuphoprasit. Cobots: Robots for collaboration with human operators, 1996.

[5] J. Broekens, M. Heerink, and H. Rosendal. Assistive social robots in elderly care: a review. *Gerontechnology*, pages 94–103, 2009.

[6] C. A. Avizzano. *Human-Robot Interactions in Future Military Operations*. Routledge, 2016.

[7] R. R. Murphy. Human-robot interaction in rescue robotics. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 34(2):138–154, May 2004.

[8] N. Syll, V. Bonnet, F. Colledani, and P. Fraisse. Ergonomic contribution of able exoskeleton in automotive industry. *International Journal of Industrial Ergonomics*, 44(4):475–481, July 2004.

[9] A. de Santis, B. Siciliano, A. de Luca, and A. Bicchi. An atlas of physical human–robot interaction. *Mechanism and Machine Theory*, 43(3):253–270, March 2008.

[10] MarketsandMarkets. Collaborative robot market size, growth, trend and forecast to 2025, Oct 2018.

[11] Z. A. Khan. Attitudes towards intelligent service robots. S-100 44 STOCKHOLM, Sweden, Aug 1998. Royal Institure of Technology (KTH).

[12] M. Wongphati, Y. Matsuda, H. Osawa, and M. Imai. Where do you want to use a robotic arm? and what do you want from the robot? In *Proceedings - IEEE International Workshop on Robot and Human Interactive Communication*, pages 322–327, 09 2012. ISBN 978-1-4673-4604-7. doi: 10.1109/ROMAN.2012.6343773.

[13] D. Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. Henry Holt and Co., Inc., New York, NY, USA, 1982. ISBN 0716715678.

[14] A. Andreopoulos and J. K. Tsotsos. 50 years of object recognition, directions forward. *Computer Vision and Image Understanding*, 117(8):827–891, August 2013.

[15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[16] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.

[17] L. Roberts. *Machine Perception of Three-Dimensional Solids*. Garland Publishing, New York, 01 1963. ISBN 0-8240-4427-4.

[18] S. Papert. The summer vision project. 1966.

[19] M. A. Fischler and R. A. Elschlager. The representation and matching of pictorial structures. *IEEE Trans. Comput.*, 22(1):67–92, 01 1973. ISSN 0018-9340.

[20] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[21] R. A. Brooks and T. O. Binford. Interpretive vision and restriction graphs. In *Proceedings of the First AAAI Conference on Artificial Intelligence*, AAAI'80, pages 21–27, 1980.

[22] D. G. Lowe. Lowe, d.g.: Three-dimensional object recognition from single two-dimensional images. artif. intell. 31, 355-395. *Artificial Intelligence*, 31:355–395, 03 1987. doi: 10.1016/0004-3702(87)90070-1.

[23] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. volume 1, pages I–511, 02 2001. ISBN 0-7695-1272-0. doi: 10.1109/CVPR.2001.990517.

[24] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, June 2005. doi: 10.1109/CVPR.2005.177.

[25] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. volume 2, pages 2169 – 2178, 02 2006. ISBN 0-7695-2597-0. doi: 10.1109/CVPR.2006.68.

[26] P. F. Felzenszwalb, R. Girshick, and D. Mcallester. Visual object detection with deformable part models. volume 56, pages 2241–2248, 06 2010. doi: 10.1109/CVPR.2010.5539906.

[27] M. Everingham, A. Zisserman, C. K. I. Williams, L. V. Gool, M. Allan, C. M. Bishop, O. Chapelle, N. Dalal, T. Deselaers, G. Dorkó, S. Duffner, J. Eichhorn, J. D. R. Farquhar, M. Fritz, C. Garcia, T. Griffiths, F. Jurie, D. Keysers, M. Koskela, J. Laaksonen, D. Larlus, B. Leibe, H. Meng, H. Ney,

B. Schiele, C. Schmid, E. Seemann, J. Shawe-taylor, A. Storkey, O. Szedmak, B. Triggs, I. Ulusoy, V. Viitaniemi, and J. Zhang. The 2005 pascal visual object classes challenge, 2006.

[28] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009.

[29] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. volume 8693, 04 2014. doi: 10.1007/978-3-319-10602-1_48.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012. doi: 10.1145/3065386.

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

[32] Y. Le Cun. Generalization and network design strategies. *Connectionism in Perspective*, 01 1989.

[33] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[34] Y. Lecun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Handwritten digit recognition with a back-propagation network. *Neural Information Processing Systems*, 2, 11 1997.

[35] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science (New York, N.Y.)*, 313:504–7, 08 2006.

[36] A.-r. Mohamed, G. E. Dahl, and G. Hinton. Acoustic modeling using deep belief networks. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20:14 – 22, 02 2012.

[37] J. Wang and S. Li. Comparing the influence of depth and width of deep neural network based on fixed number of parameters for audio event detection. 04 2018. doi: 10.1109/ICASSP.2018.8461713.

[38] A. N. Gorban, E. M. Mirkes, and I. Y. Tyukin. How deep should be the depth of convolutional neural networks: a backyard dog case study. *Cognitive Computation*, 12:388–397, 2019.

[39] J. Ba and R. Caruana. Do deep nets really need to be deep? *ArXiv*, abs/1312.6184, 2014.

[40] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang. The expressive power of neural networks: A view from the width. In *NIPS*, 2017.

[41] M. Thoma. Analysis and optimization of convolutional neural network architectures. *ArXiv*, abs/1707.09725, 2017.

[42] S. Solla and Y. Lecun. *Constrained neural networks for pattern recognition*, volume IV, pages 142–161. Prentice Hall, 1991.

[43] E. Barnard. Optimization for training neural nets. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 3:232–40, 02 1992.

[44] A.-L. Cauchy. Methode generale pour la resolution des systemes d'equations simultanees. *C.R. Acad. Sci. Paris*, 25:536–538, 1847.

[45] B. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4:1–17, 12 1964.

[46] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, arXiv 1409.1556, 09 2014.

[47] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[48] M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, abs/1312.4400, 12 2013.

[49] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016. doi: 10.1109/CVPR.2016.90.

[50] Z.-Q. Zhao, P. Zheng, S. tao Xu, and X. Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 2018.

[51] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104:154–171, 2013.

[52] B. Alexe, T. Deselaers, and V. Ferrari. Measuring the objectness of image windows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2189–2202, Nov 2012. doi: 10.1109/TPAMI.2012.28.

[53] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[54] R. B. Girshick. Fast r-cnn. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 04 2015.

[55] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39: 1137–1149, 06 2015.

[56] J. Dai, Y. Li, K. He, and J. Sun. R-fcn: Object detection via region-based fully convolutional networks. In *NIPS*, 2016.

[57] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie. Feature pyramid networks for object detection. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017.

[58] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2155–2162, 2014.

[59] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[60] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016.

[61] J. Cartucho, R. Ventura, and M. Veloso. Robust object recognition through symbiotic deep learning in mobile robots. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2336–2341, 2018.

[62] E. S. Olivas, J. D. M. Guerrero, M. M. Sober, J. R. M. Benedito, and A. J. S. Lopez. *Handbook Of Research On Machine Learning Applications and Trends: Algorithms, Methods and Techniques - 2 Volumes*. Information Science Reference - Imprint of: IGI Publishing, Hershey, PA, 2009. ISBN 1605667668.

[63] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, 2017.

[64] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *ArXiv*, abs/1804.02767, 2018.

[65] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[66] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[67] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, Y. Bengio, A. Bergeron, J. Bergstra, V. Bisson, J. B. Snyder, N. Bouchard, N. Boulanger-Lewandowski, X. Bouthillier, A. de Brébisson, O. Breuleux, P.-L. Carrier, K. Cho, J. Chorowski, P. Christiano, T. Cooijmans, M.-A. Côté, M. Côté, A. Courville, Y. N. Dauphin, O. Delalleau, J. Demouth, G. Desjardins, S. Dieleman, L. Dinh, M. Ducoffe, V. Dumoulin, S. E. Kahou,

D. Erhan, Z. Fan, O. Firat, M. Germain, X. Glorot, I. Goodfellow, M. Graham, C. Gulcehre, P. Hamel, I. Harlouchet, J.-P. Heng, B. Hidasi, S. Honari, A. Jain, S. Jean, K. Jia, M. Korobov, V. Kulkarni, A. Lamb, P. Lamblin, E. Larsen, C. Laurent, S. Lee, S. Lefrancois, S. Lemieux, N. Léonard, Z. Lin, J. A. Livezey, C. Lorenz, J. Lowin, Q. Ma, P.-A. Manzagol, O. Mastropietro, R. T. McGibbon, R. Memisevic, B. van Merriënboer, V. Michalski, M. Mirza, A. Orlandi, C. Pal, R. Pascanu, M. Pezeshki, C. Raffel, D. Renshaw, M. Rocklin, A. Romero, M. Roth, P. Sadowski, J. Salvatier, F. Savard, J. Schlüter, J. Schulman, G. Schwartz, I. V. Serban, D. Serdyuk, S. Shabanian, Étienne Simon, S. Spieckermann, S. R. Subramanyam, J. Sygnowski, J. Tanguay, G. van Tulder, J. Turian, S. Urban, P. Vincent, F. Visin, H. de Vries, D. Warde-Farley, D. J. Webb, M. Willson, K. Xu, L. Xue, L. Yao, S. Zhang, and Y. Zhang. Theano: A python framework for fast computation of mathematical expressions, 2016.

[68] F. Chollet. keras. `https://github.com/fchollet/keras`, 2015.

[69] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding, 2014.

[70] F. Chollet. *Deep Learning with Python*. Manning Publications Co., USA, 1st edition, 2017. ISBN 1617294438.

[71] A. Gron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 1st edition, 2017. ISBN 1491962291.

[72] A. Rosebrock. *Deep Learning for Computer Vision with Python: Starter Bundle*. PyImageSearch, 2017.

[73] J. Huang, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. 11 2016.

[74] G. Jocher, Y. Kwon, guigarfr, perry0418, J. Veitch-Michaelis, Ttayu, D. Suess, F. Baltacı, G. Bianconi, IlyaOvodov, Marc, C. Lee, D. Kendall, Falak, F. Reveriano, FuLin, GoogleWiki, J. Nataprawira, J. Hu, LinCoce, LukeAI, N. Zarrabi, O. Reda, P. Skalski, S. Song, T. Havlik, T. M. Shead, and W. Xinyu. ultralytics/yolov3: v8 - Final Darknet Compatible Release, Nov. 2020. URL `https://doi.org/10.5281/zenodo.4279923`.

[75] A. Kuznetsova, H. Rom, N. Alldrin, J. R. R. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, T. Duerig, and V. Ferrari. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *ArXiv*, abs/1811.00982, 2018.

[76] A. Vittorio. Toolkit to download and visualize single or multiple classes from the huge open images v4 dataset. `https://github.com/EscVM/OIDv4_ToolKit`, 2018.

[77] F. Pulgar, A. Rivera Rivas, F. Charte, and M. J. Del Jesus. On the impact of imbalanced data in convolutional neural networks performance. pages 220–232, 06 2017. ISBN 978-3-319-59649-5. doi: 10.1007/978-3-319-59650-1_19.

[78] K. Oksuz, B. C. Cam, S. Kalkan, and E. Akbas. Imbalance problems in object detection: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2020.

[79] L. Zhang, C. Zhang, H. Xiao, S. Quan, G. Kuang, and L. Liu. A class imbalance loss for imbalanced object recognition. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, PP:1–1, 05 2020.

[80] R. Kumar, S. Lal, S. Kumar, and P. Chand. Object detection and recognition for a pick and place robot. In *Asia-Pacific World Congress on Computer Science and Engineering*, pages 1–7, 2014.

[81] S. Kumar, A. Majumder, S. Dutta, R. Raja, S. Jotawar, A. Kumar, M. Soni, V. Raju, O. Kundu, E. Behera, K. Venkatesh, and R. Sinha. Design and development of an automated robotic pick & stow system for an e-commerce warehouse. 03 2017.

[82] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Intell. Res. (JAIR)*, 16:321–357, 06 2002.

[83] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[84] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers, 2020.

[85] I. Ward, H. Laga, and M. Bennamoun. *RGB-D Image-Based Object Detection: From Traditional Methods to Deep Learning Techniques*, pages 169–201. 10 2019. ISBN 978-3-030-28602-6. doi: 10.1007/978-3-030-28603-3_8.

[86] M. Takahashi, A. Moro, Y. Ji, and K. Umeda. Expandable yolo: 3d object detection from rgb-d images, 2020.

[87] S. Zia, B. Yüksel, D. Yüret, and Y. Yemez. Rgb-d object recognition using deep convolutional neural networks. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 887–894, 2017.

[88] A. Jaiswal, Y. Wu, P. Natarajan, and P. Natarajan. Class-agnostic object detection, 2020. URL `https://arxiv.org/abs/2011.14204`.

Appendix A

Verification set and predictions for the best performing models

A.1 Models trained on 1 class

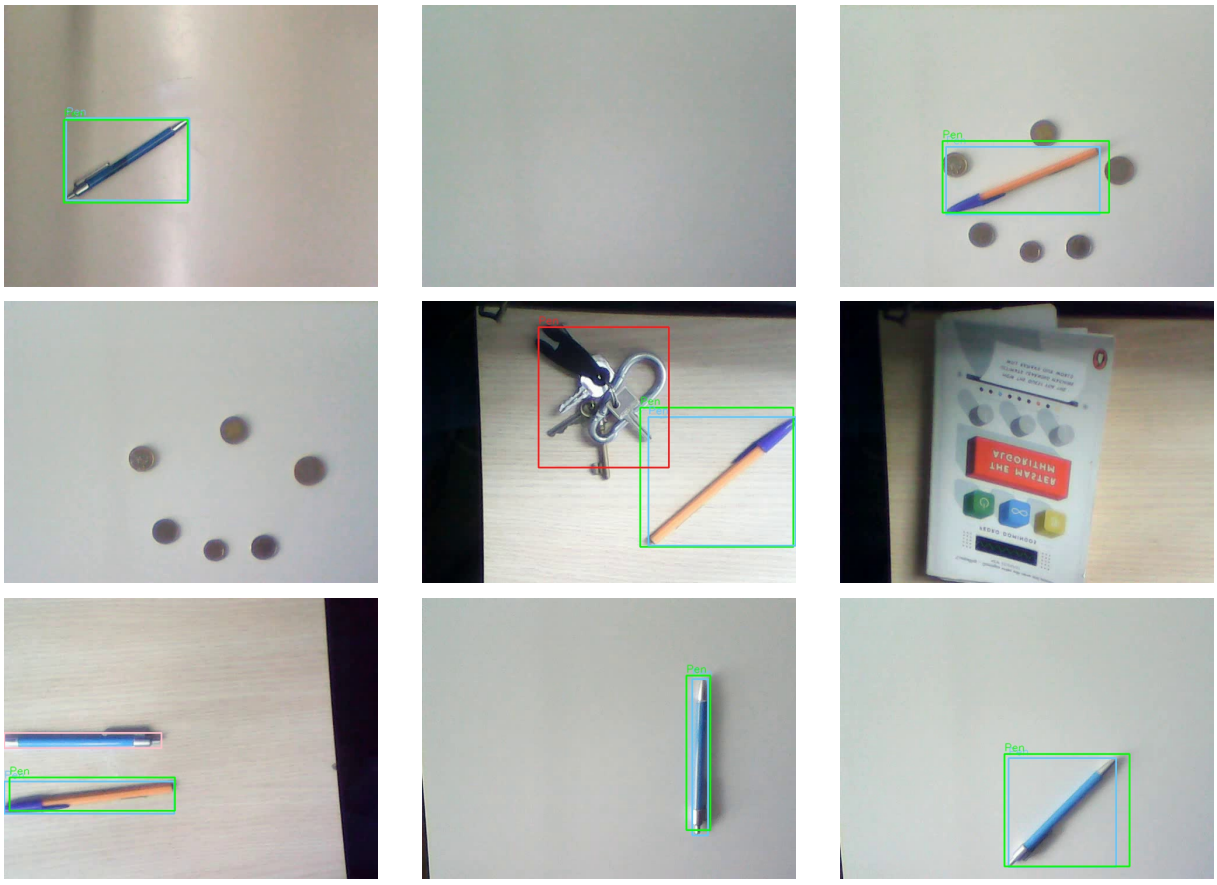


Figure A.1: Verification set and predictions for the R-CNN model (part 1 of 6).

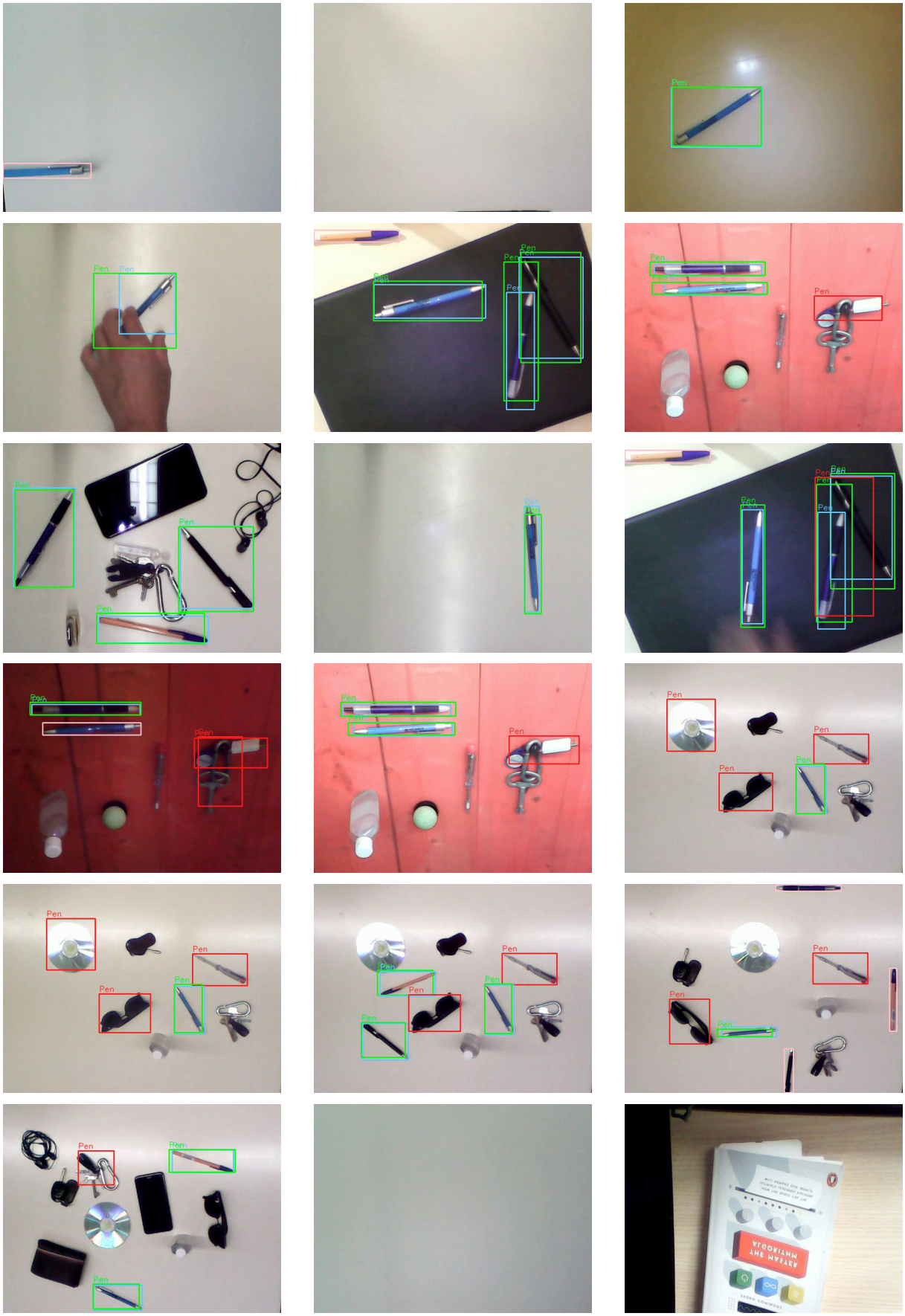


Figure A.2: Verification set and predictions for the R-CNN model (part 2 of 6).

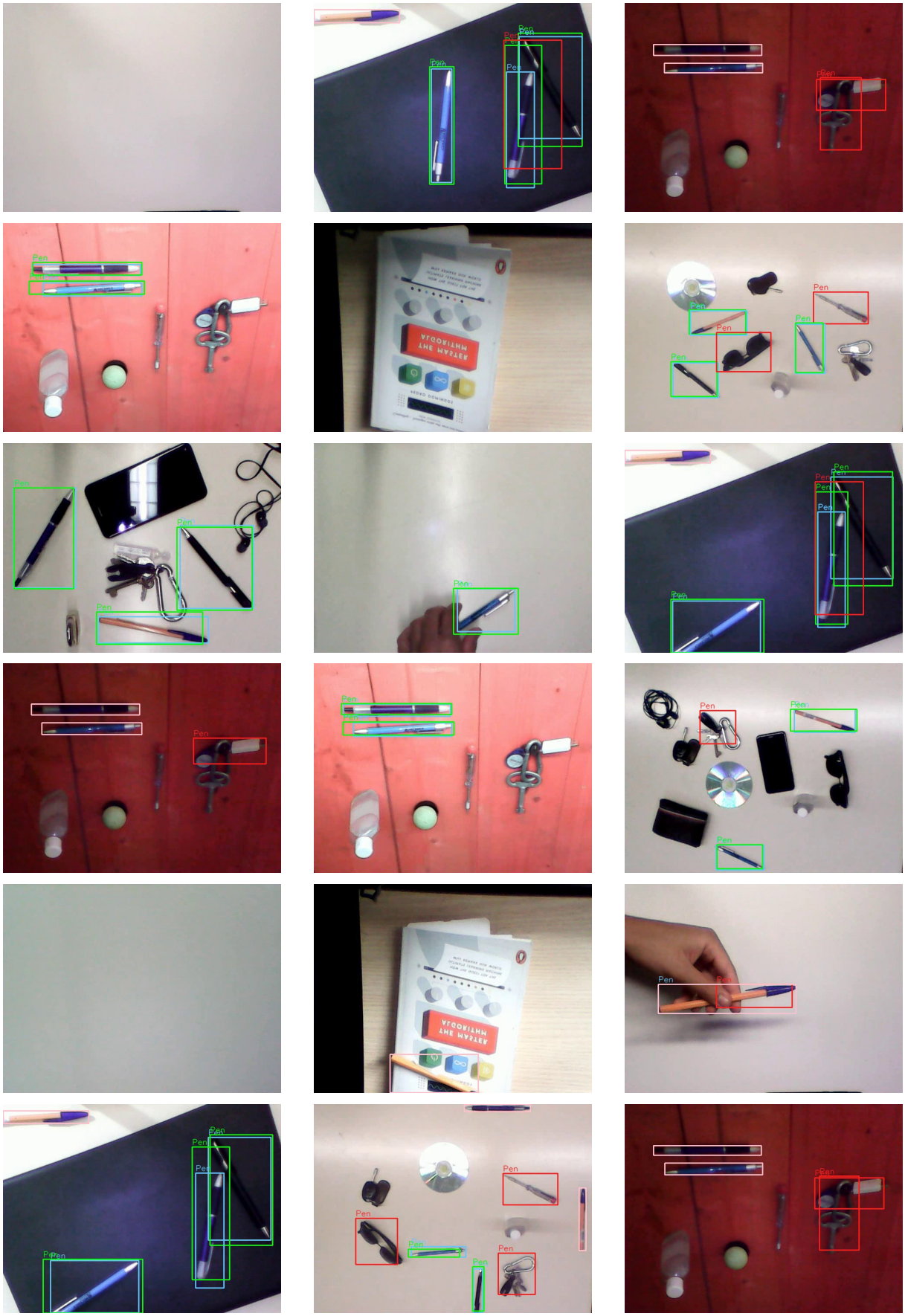


Figure A.3: Verification set and predictions for the R-CNN model (part 3 of 6).

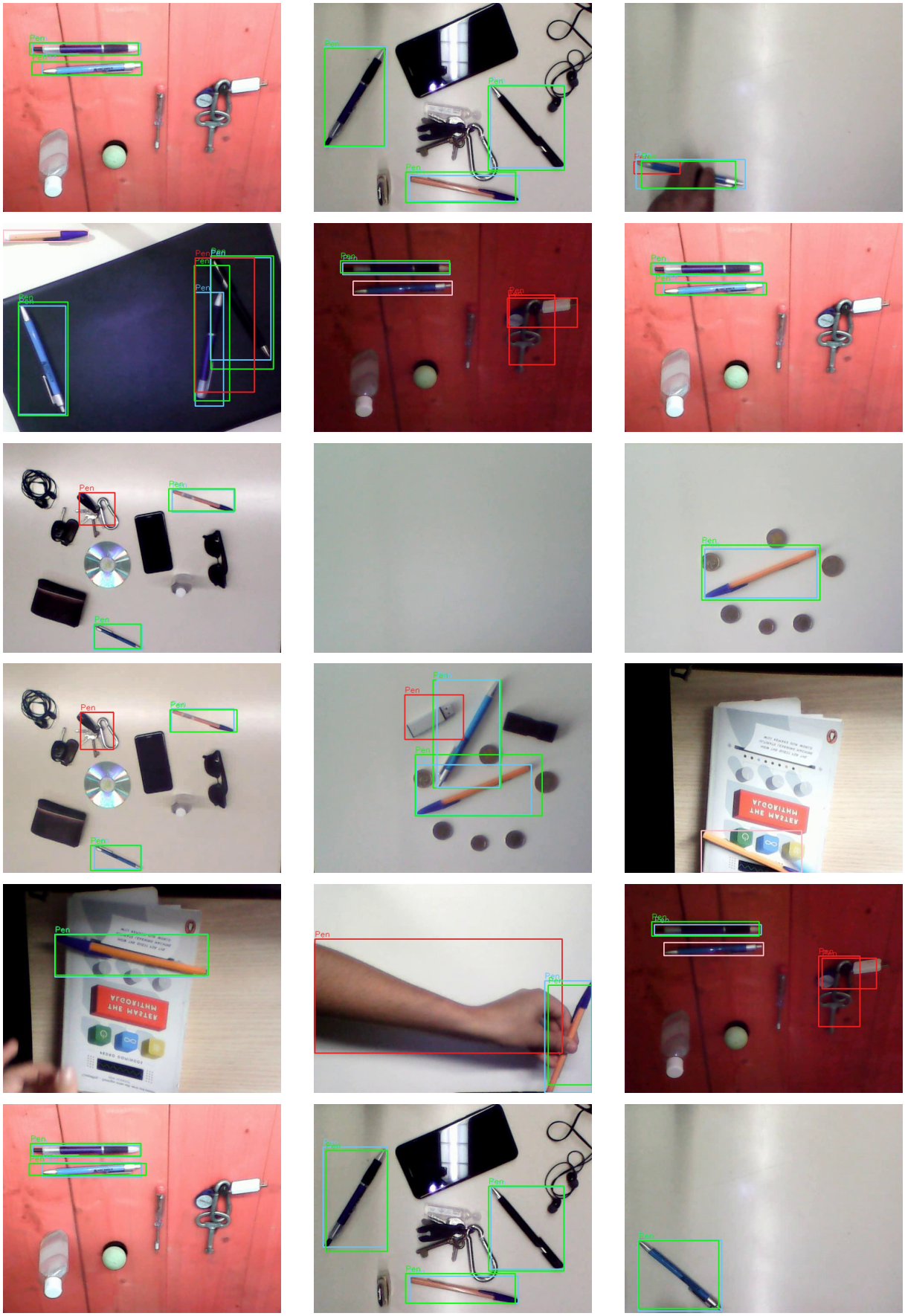


Figure A.4: Verification set and predictions for the R-CNN model (part 4 of 6).

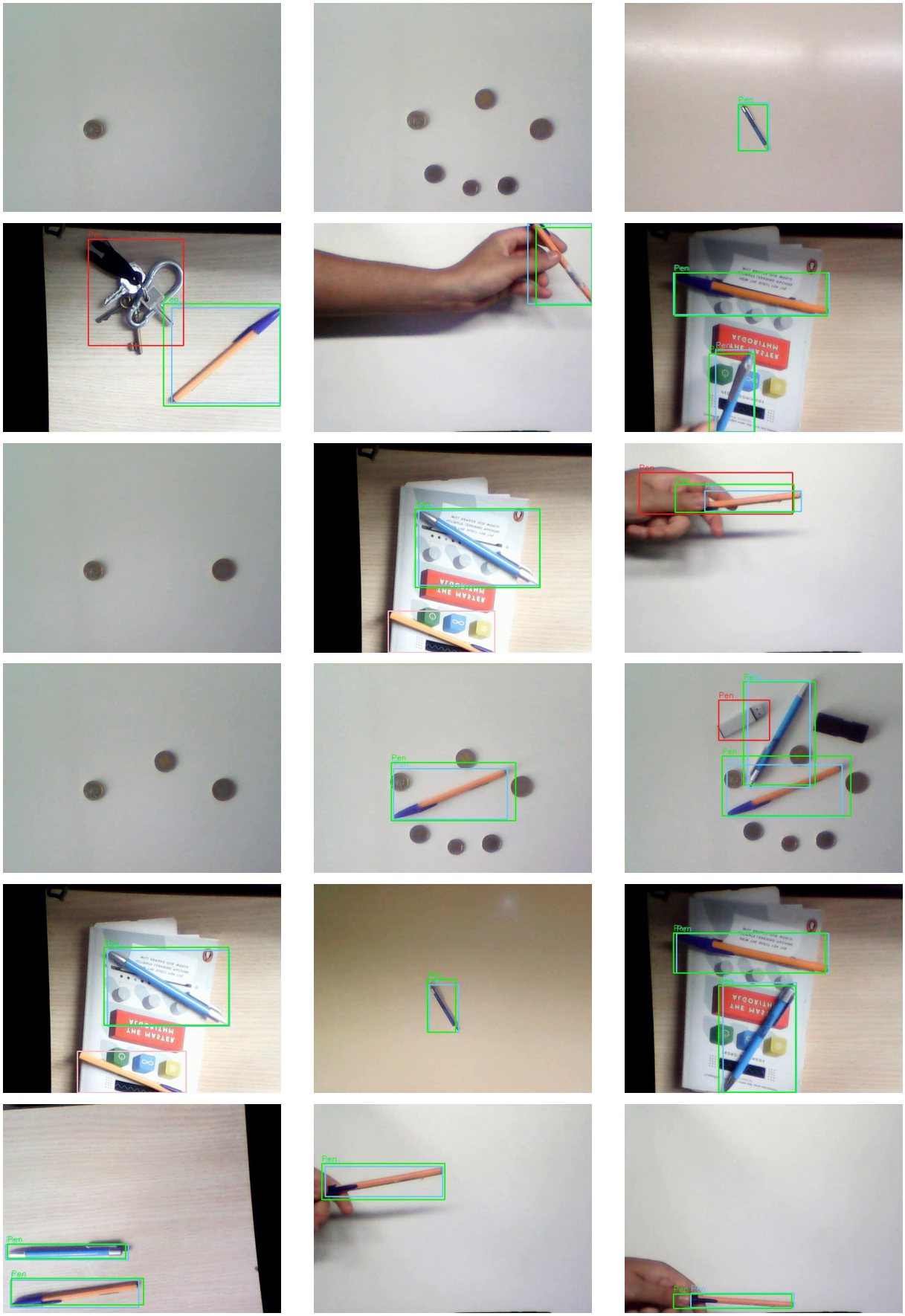


Figure A.5: Verification set and predictions for the R-CNN model (part 5 of 6).

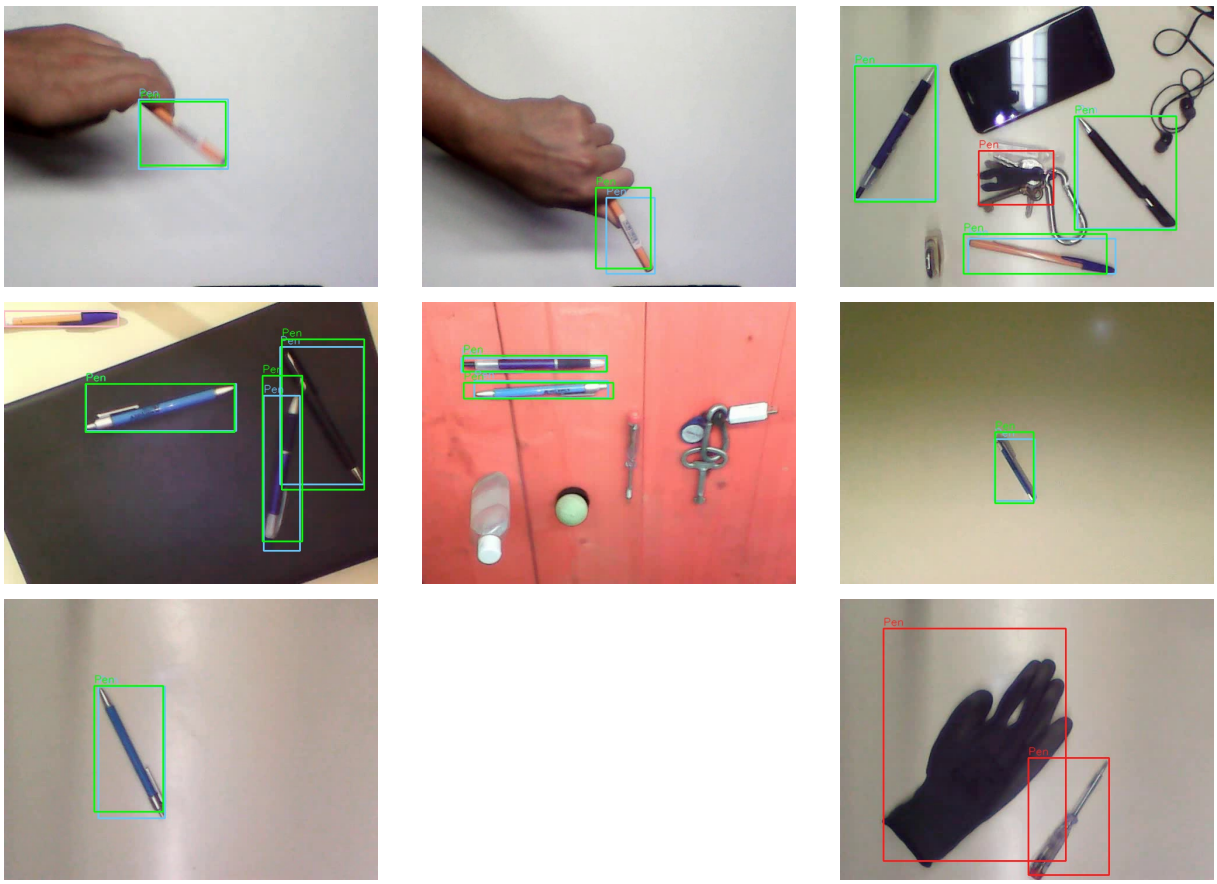


Figure A.6: Verification set and predictions for the R-CNN model (part 6 of 6).

A.2 Models trained on 4 class

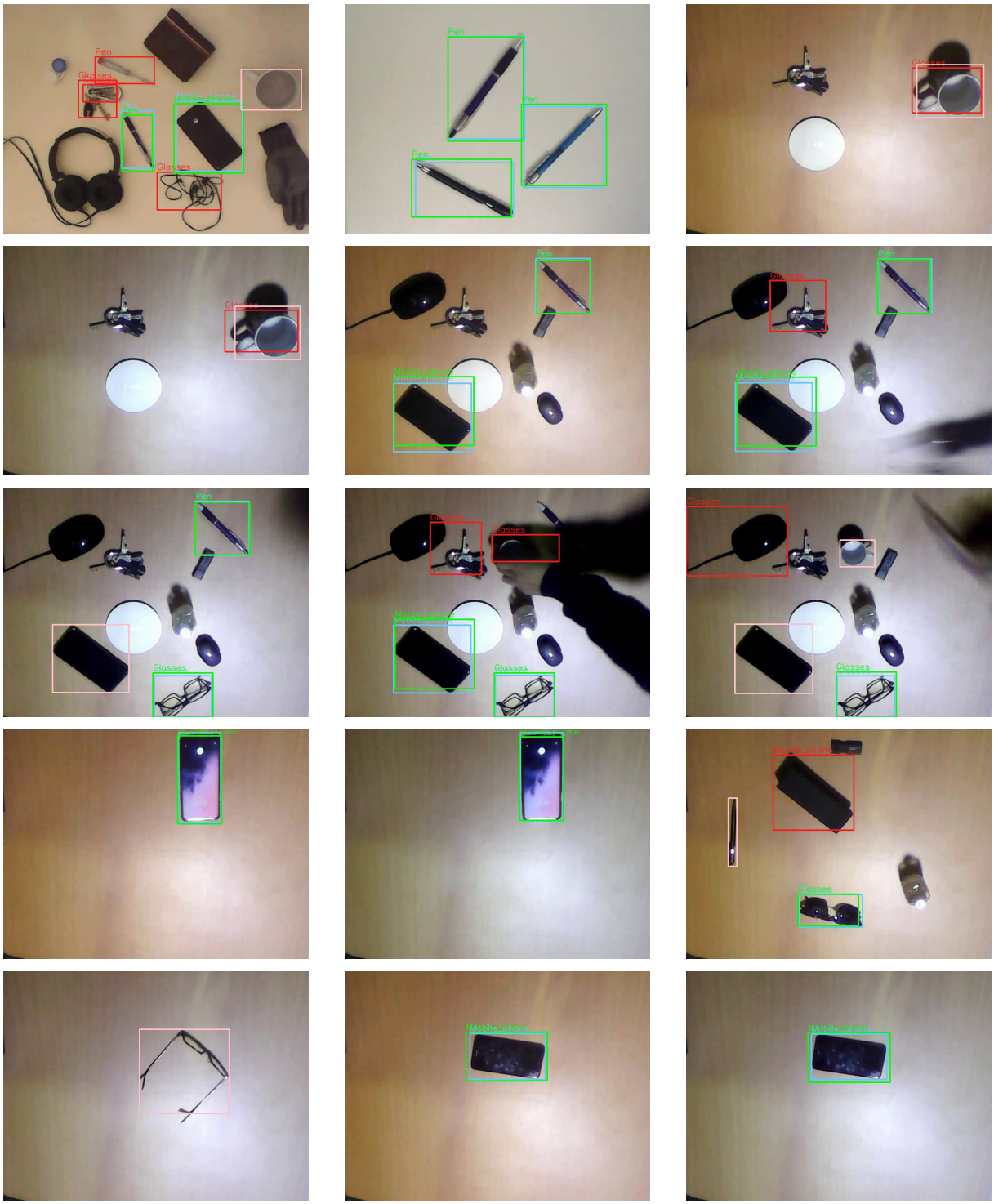


Figure A.7: Verification set and predictions for the R-CNN model (part 1 of 7).

Figure A.8: Verification set and predictions for the R-CNN model (part 2 of 7).

Figure A.9: Verification set and predictions for the R-CNN model (part 3 of 7).

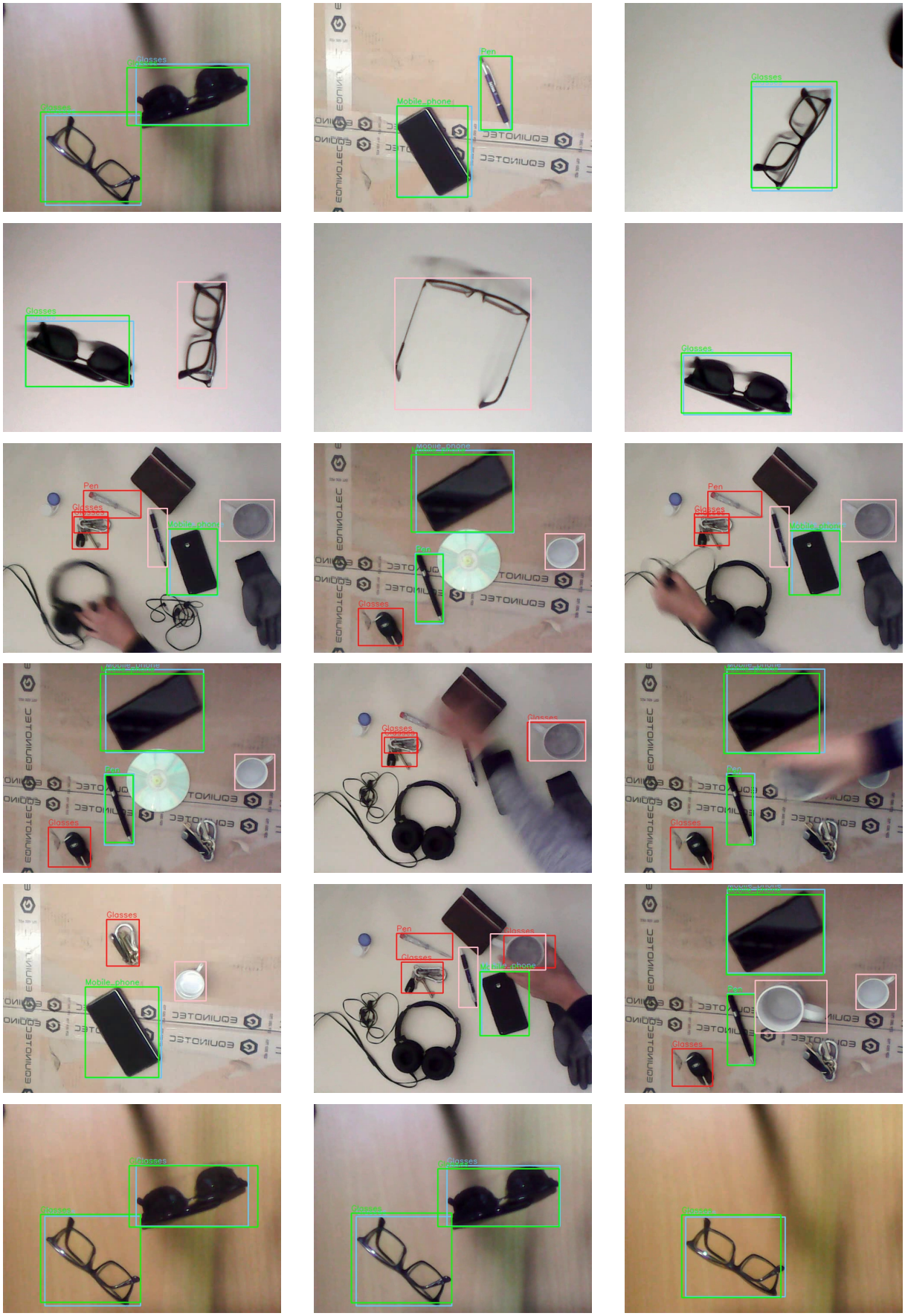


Figure A.10: Verification set and predictions for the R-CNN model (part 4 of 7).

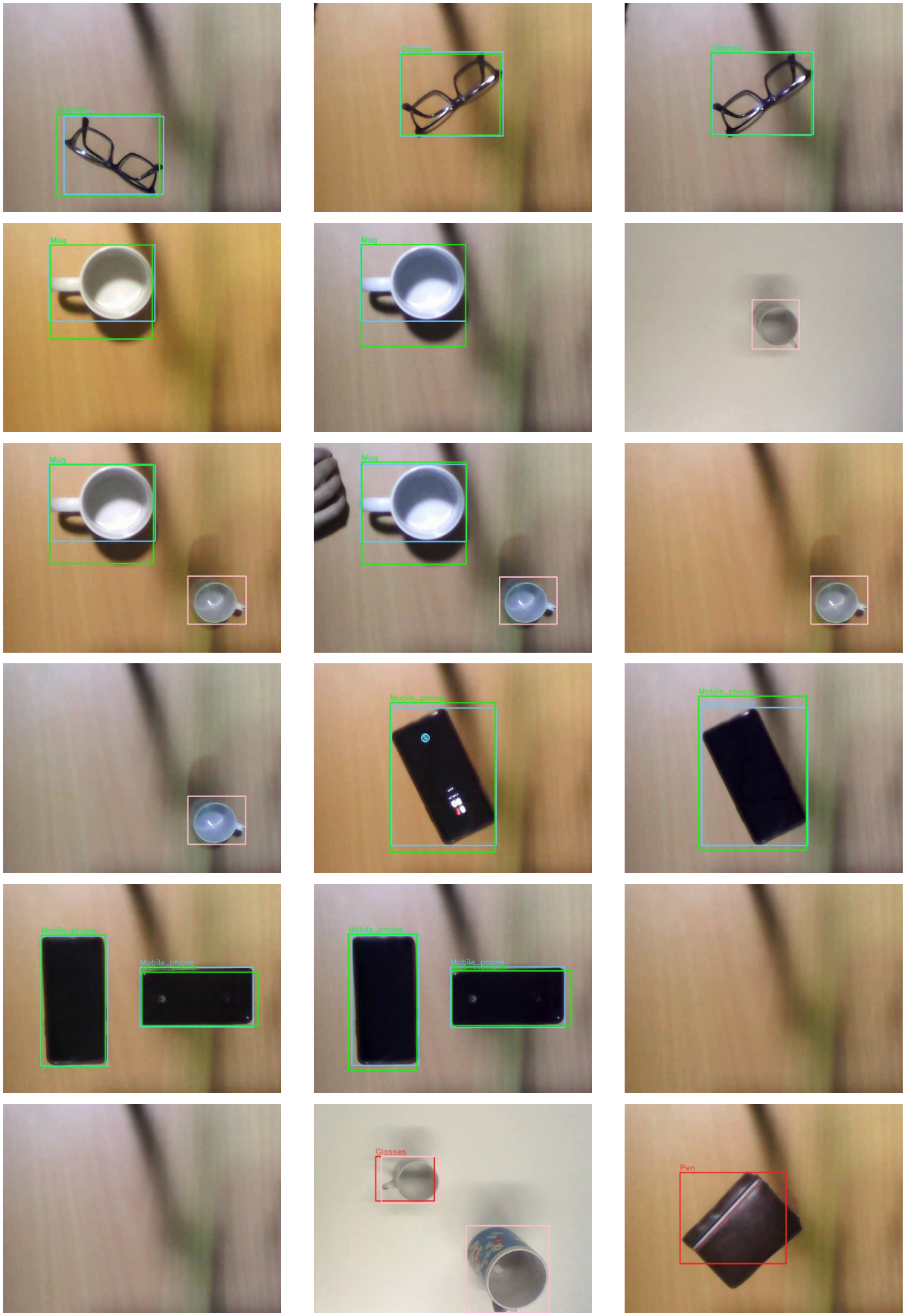


Figure A.11: Verification set and predictions for the R-CNN model (part 5 of 7).

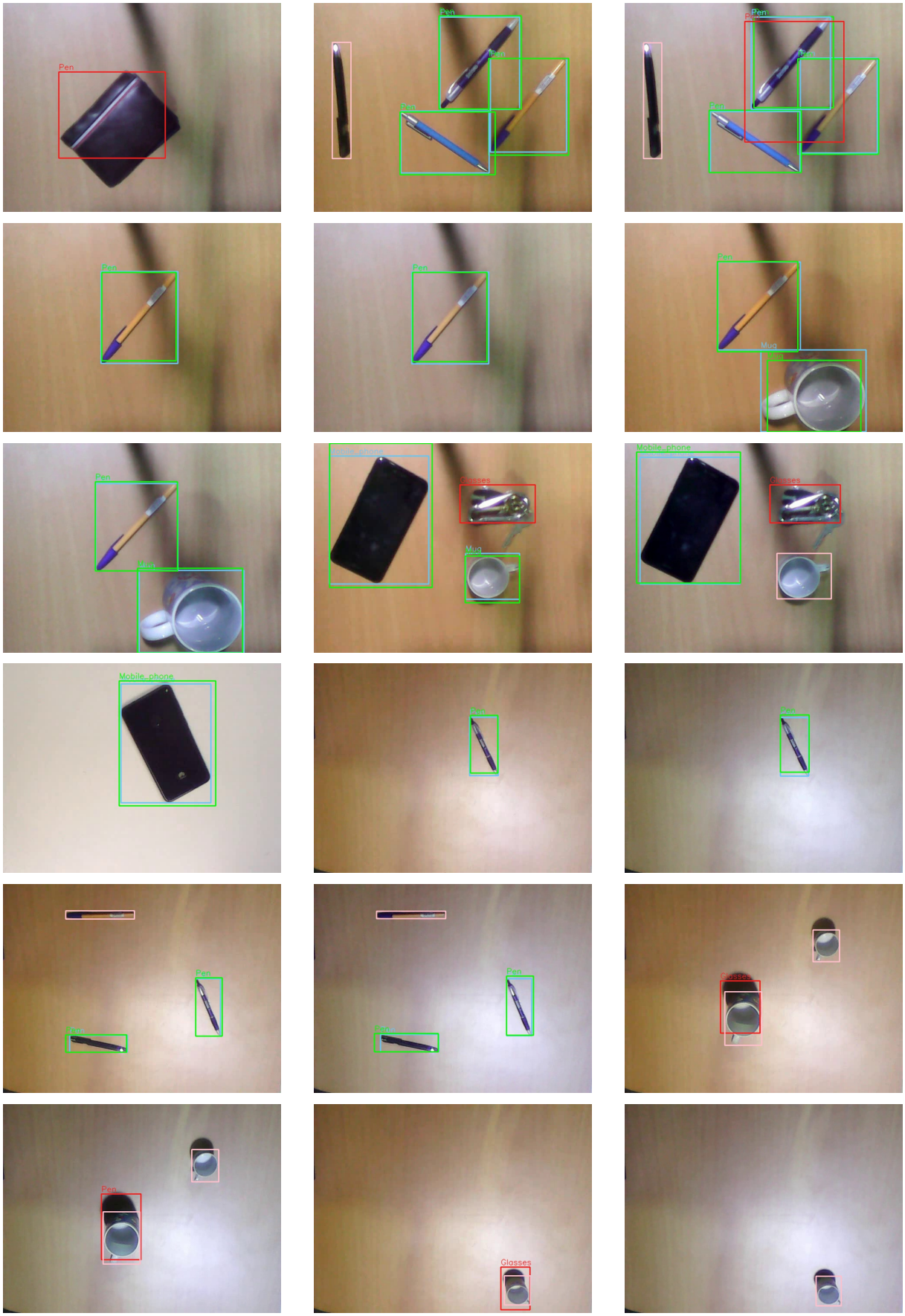


Figure A.12: Verification set and predictions for the R-CNN model (part 6 of 7).

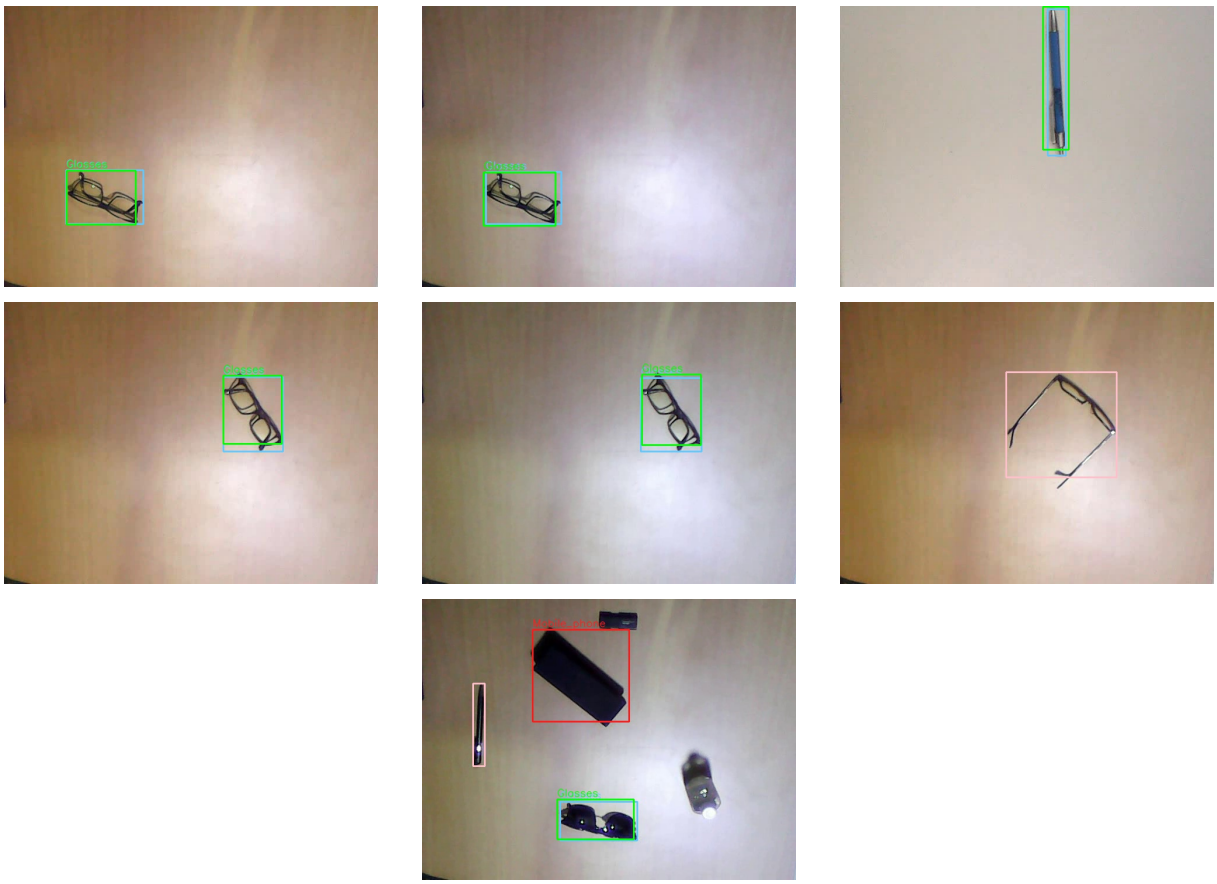


Figure A.13: Verification set and predictions for the R-CNN model (part 7 of 7).

A.3 Models trained on 10 class

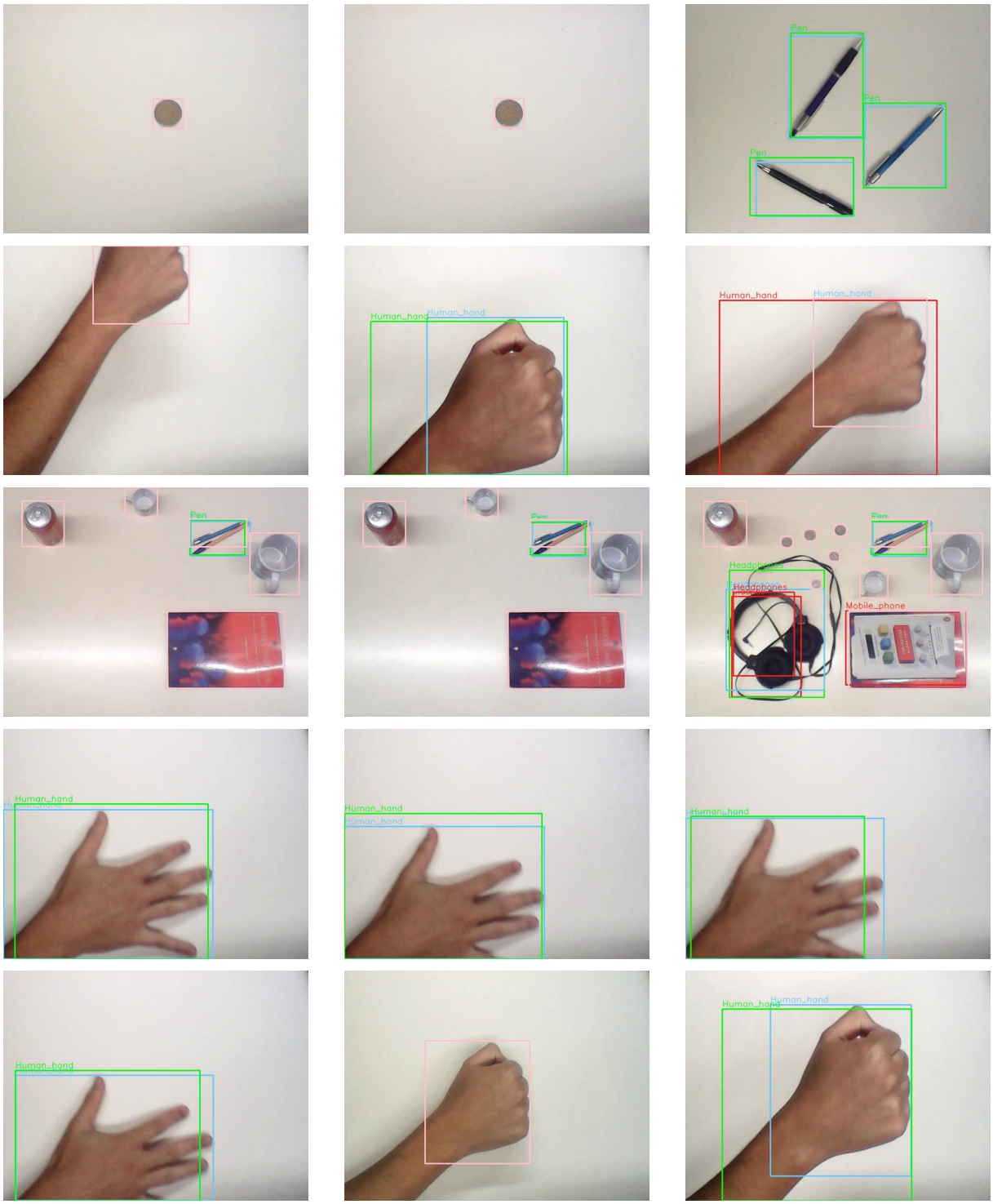


Figure A.14: Verification set and predictions for the R-CNN model trained with imbalanced dataset (part 1 of 10).

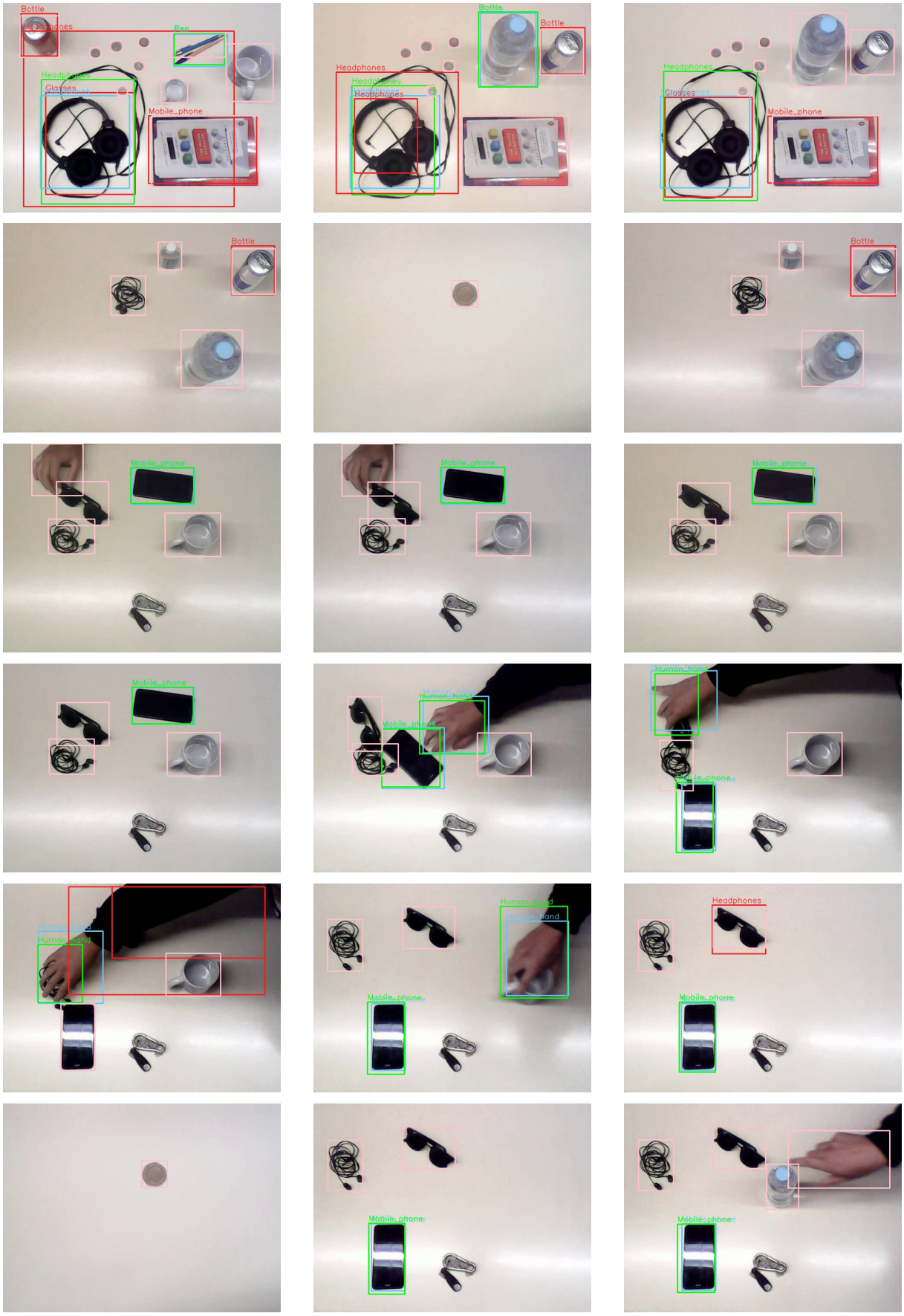


Figure A.15: Verification set and predictions for the R-CNN model trained with imbalanced dataset (part 2 of 10).

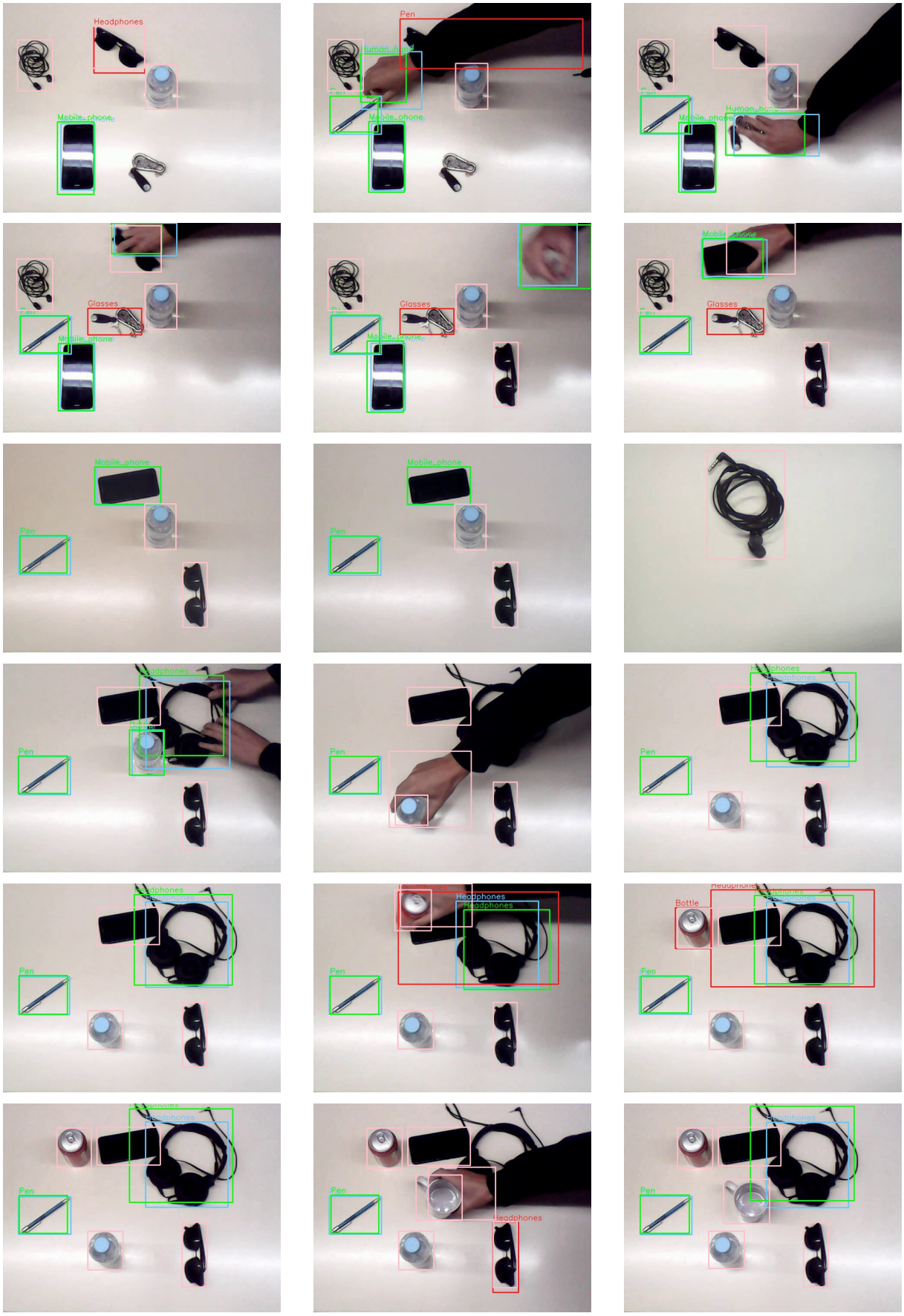


Figure A.16: Verification set and predictions for the R-CNN model trained with imbalanced dataset (part 3 of 10).

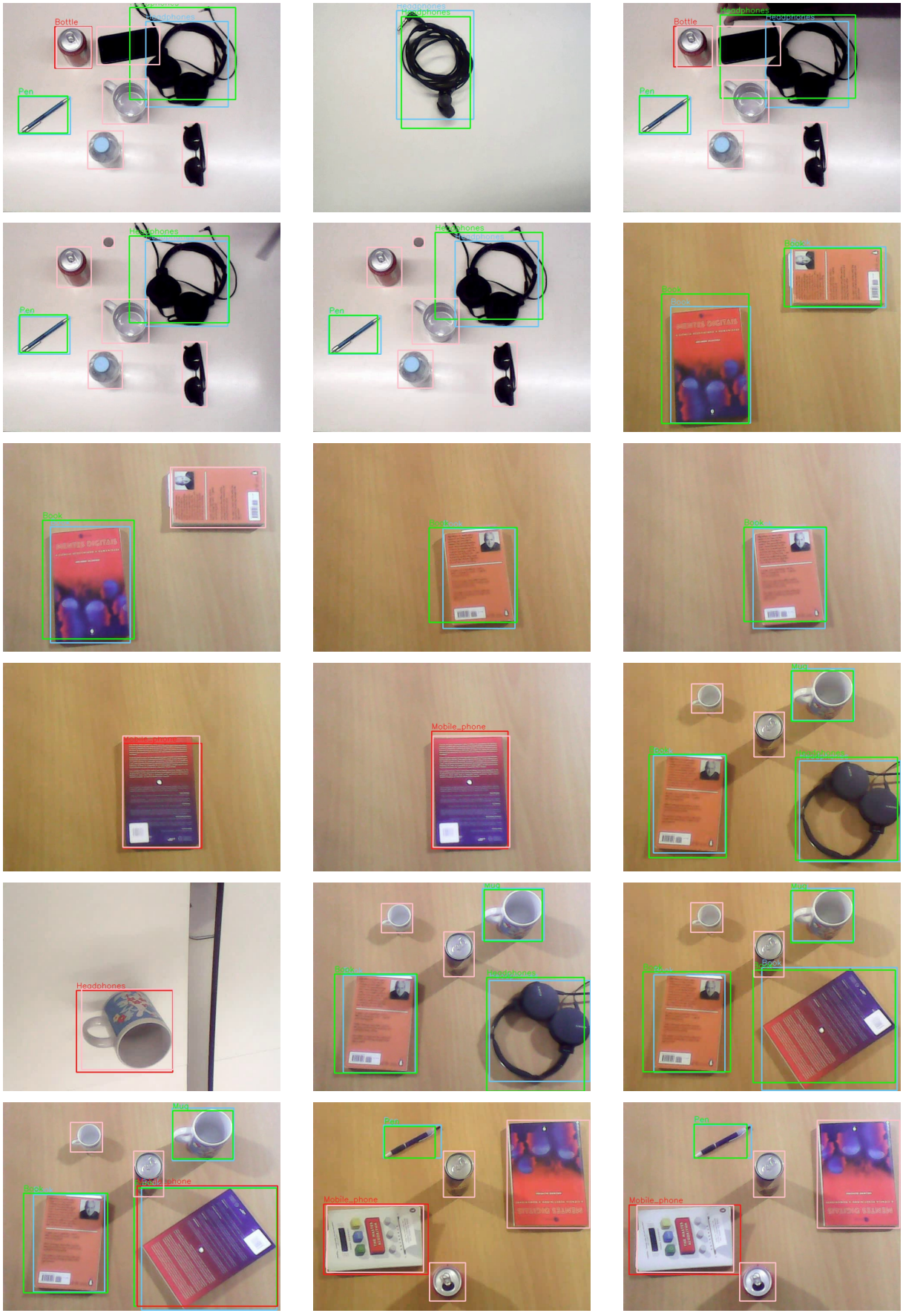


Figure A.17: Verification set and predictions for the R-CNN model trained with imbalanced dataset (part 4 of 10).

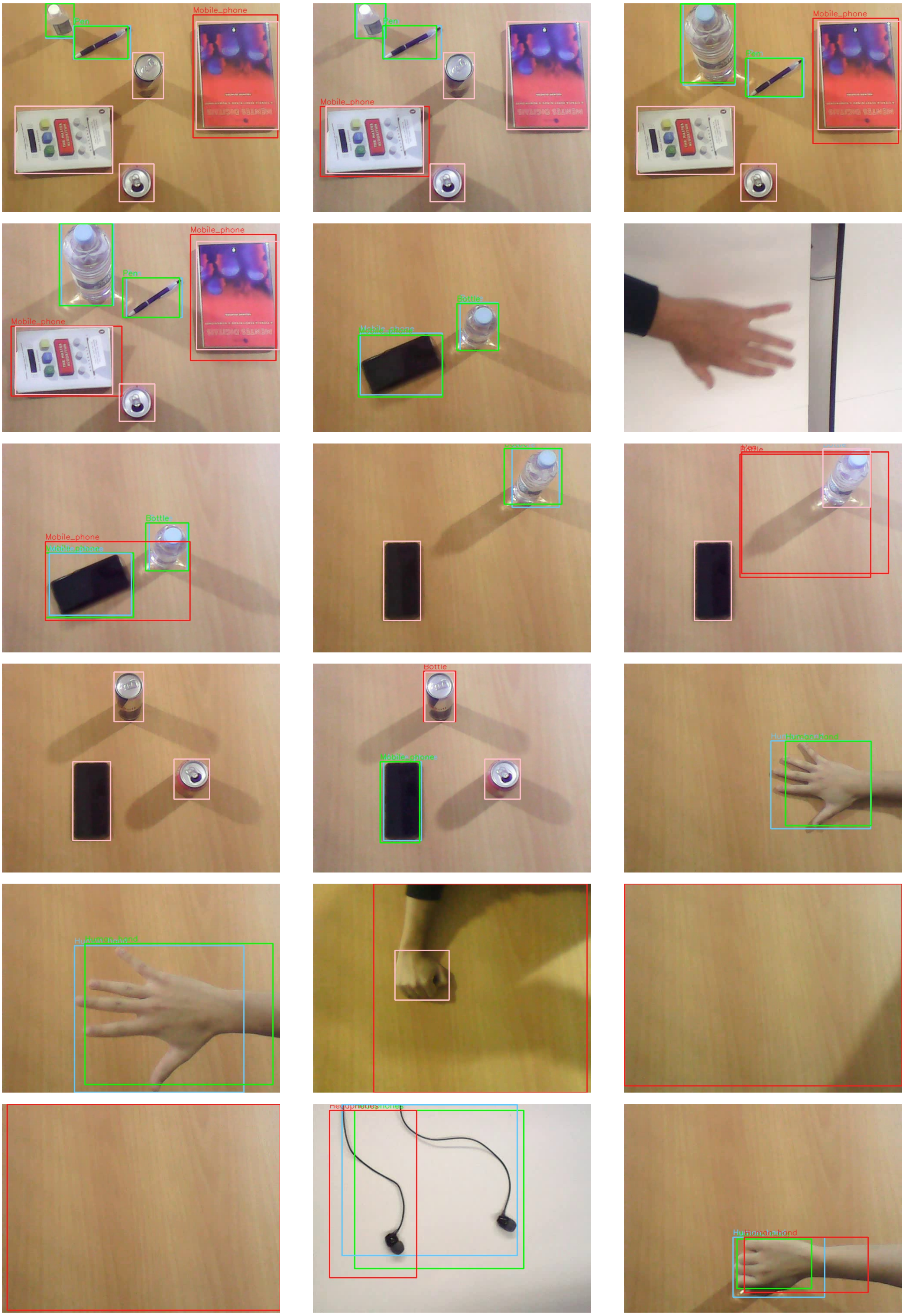


Figure A.18: Verification set and predictions for the R-CNN model trained with imbalanced dataset (part 5 of 10).

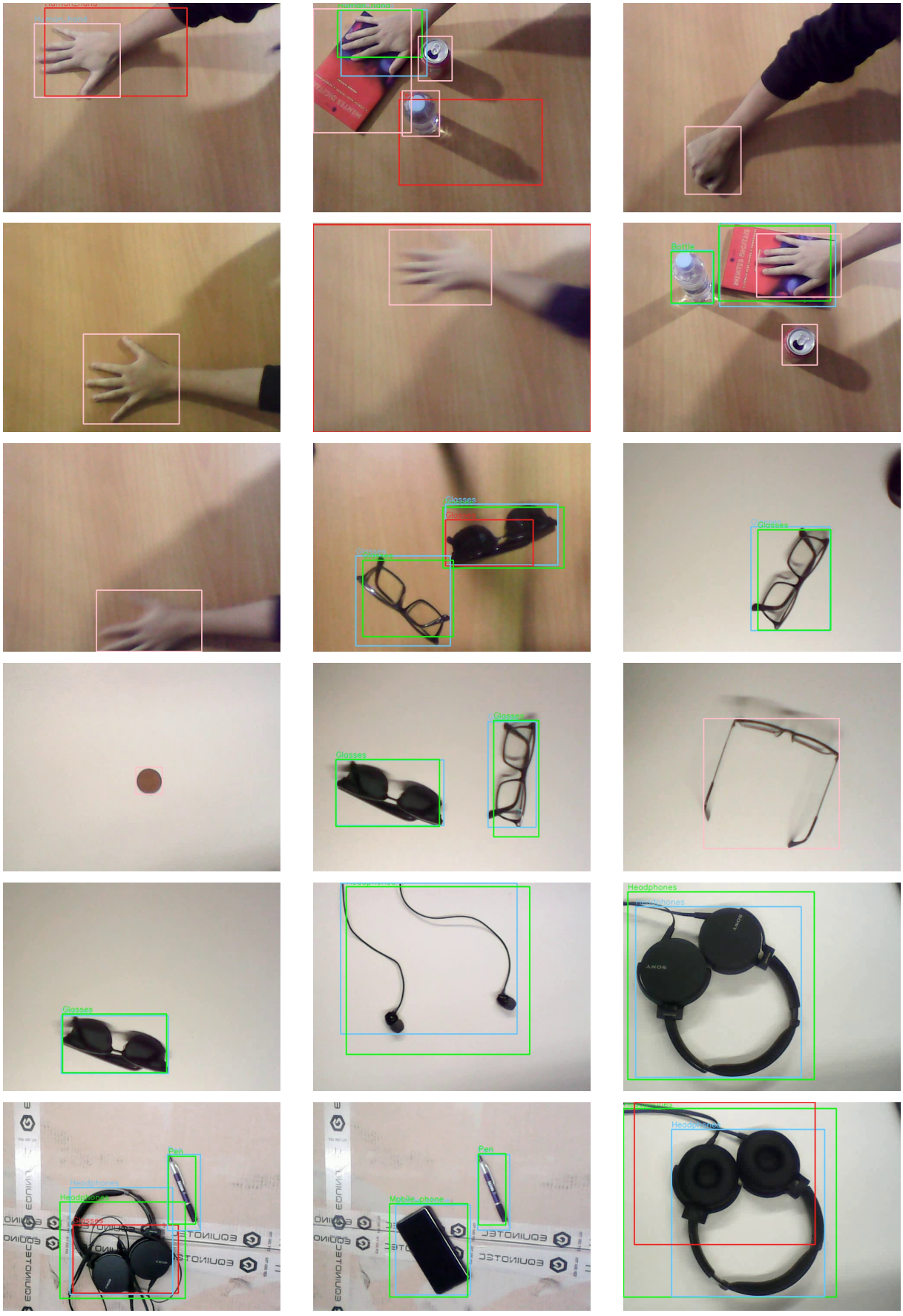


Figure A.19: Verification set and predictions for the R-CNN model trained with imbalanced dataset (part 6 of 10).

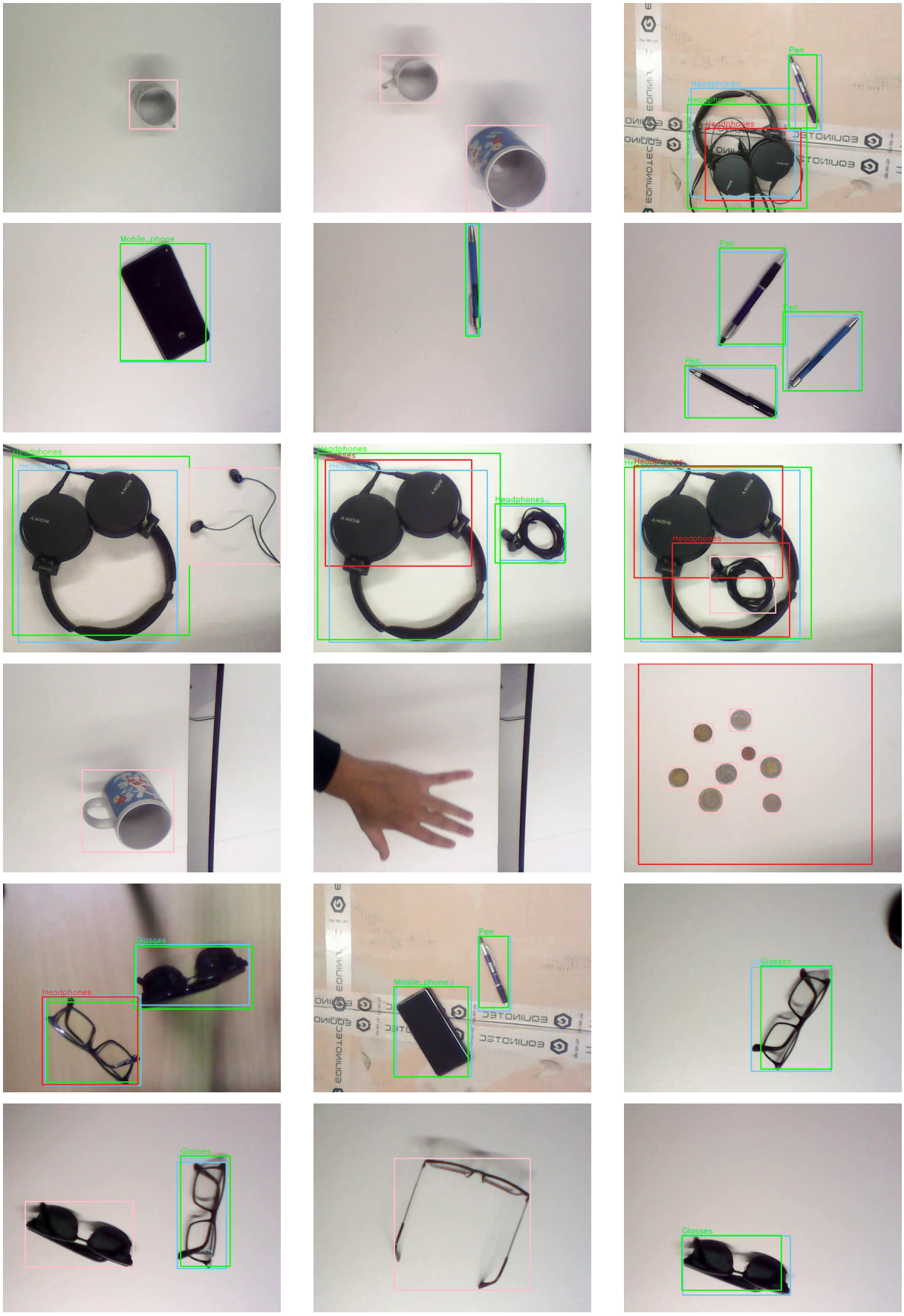


Figure A.20: Verification set and predictions for the R-CNN model trained with imbalanced dataset (part 7 of 10).

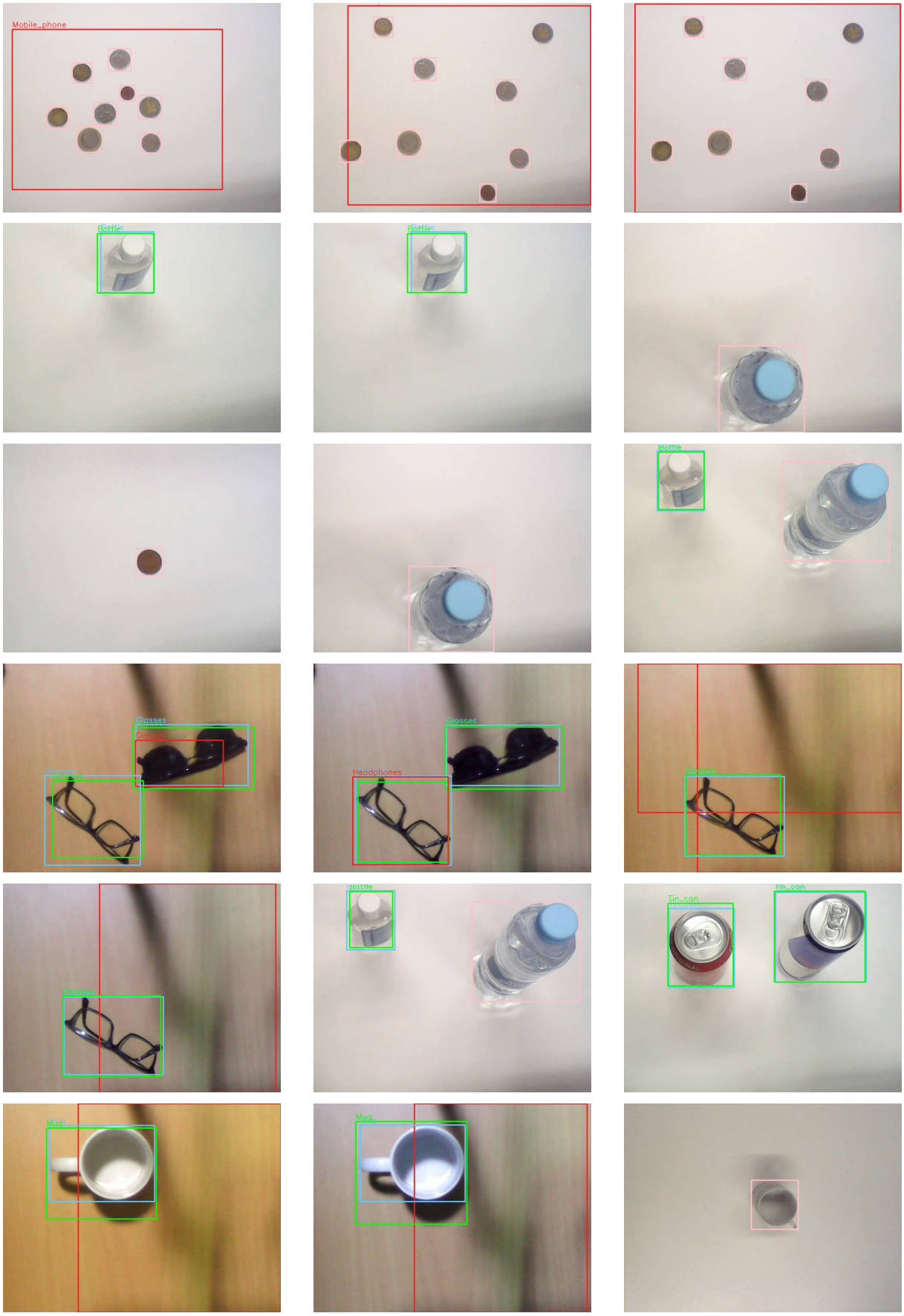


Figure A.21: Verification set and predictions for the R-CNN model trained with imbalanced dataset (part 8 of 10).

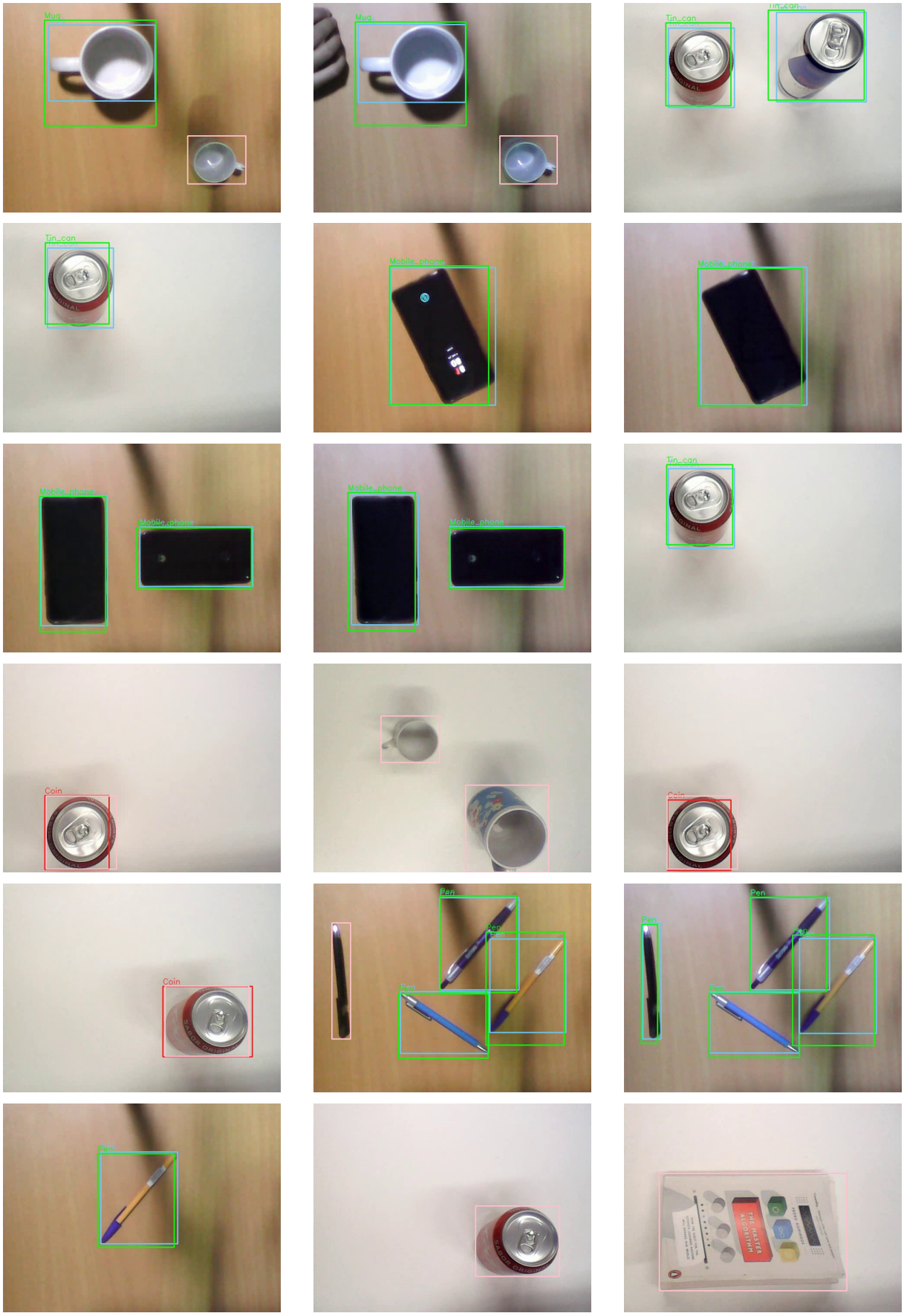


Figure A.22: Verification set and predictions for the R-CNN model trained with imbalanced dataset (part 9 of 10).

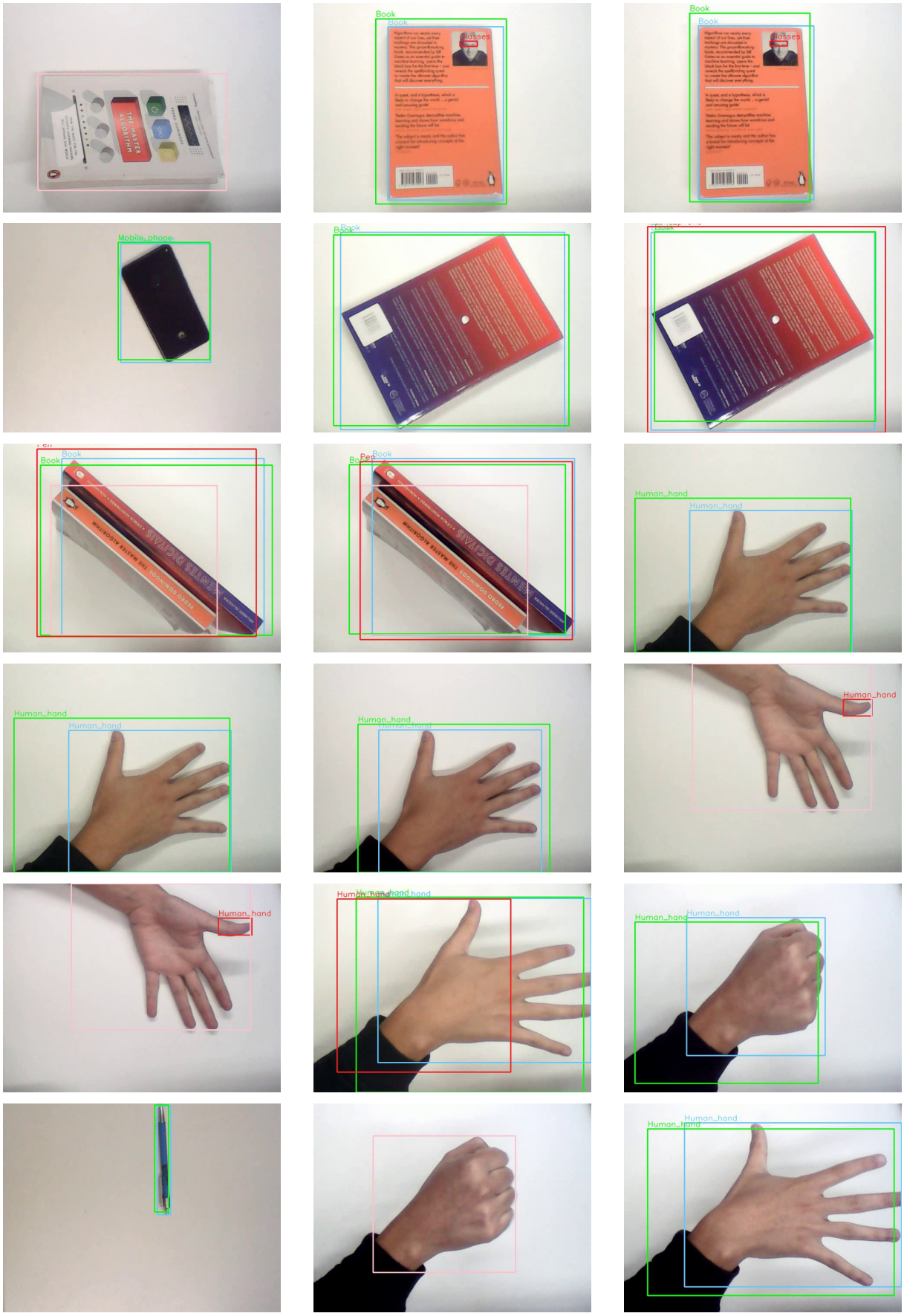


Figure A.23: Verification set and predictions for the R-CNN model trained with imbalanced dataset (part 10 of 10).

A.4 Models trained with manually annotated images

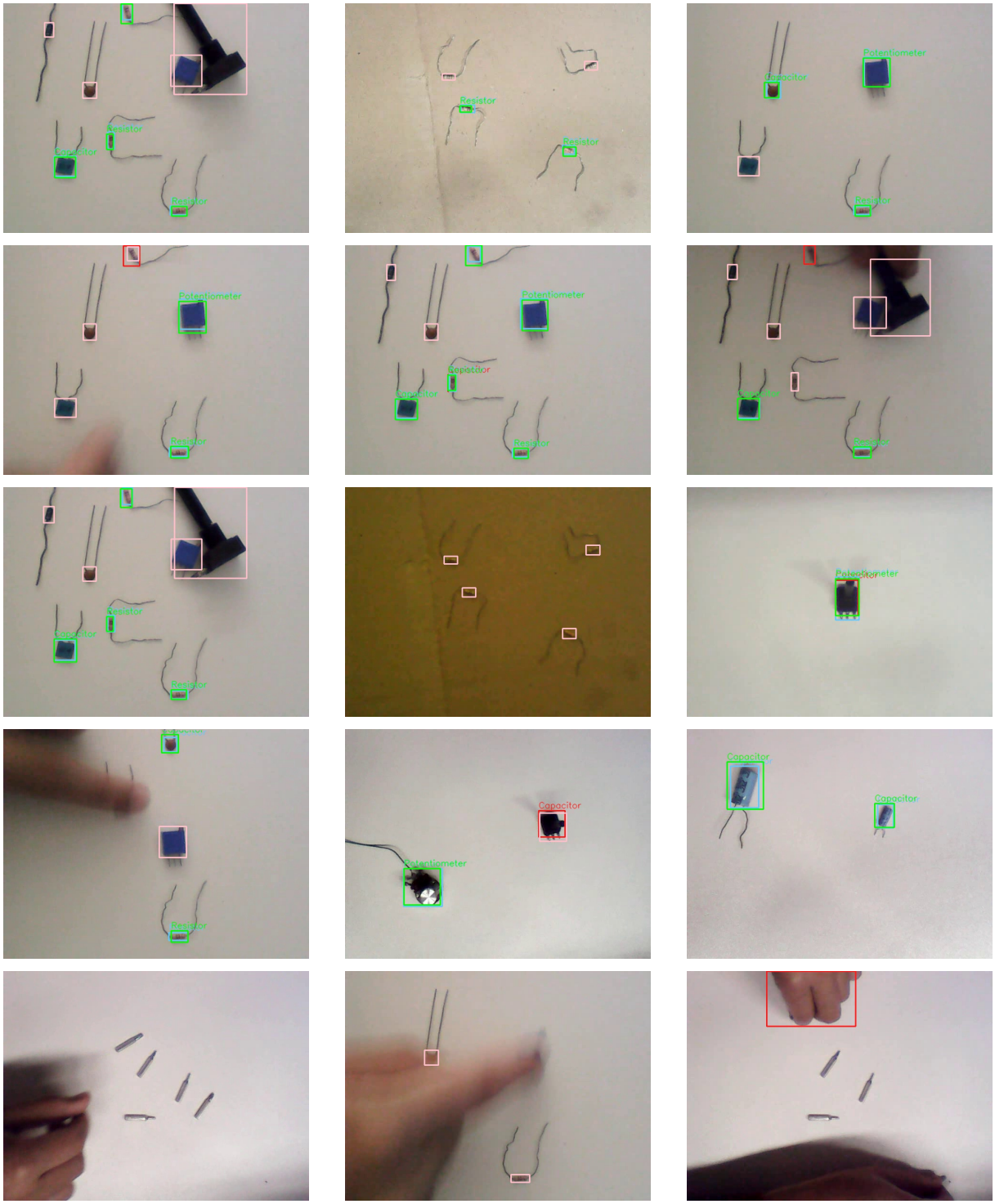


Figure A.24: Verification set and predictions for the R-CNN model trained with imbalanced dataset (part 1 of 6).

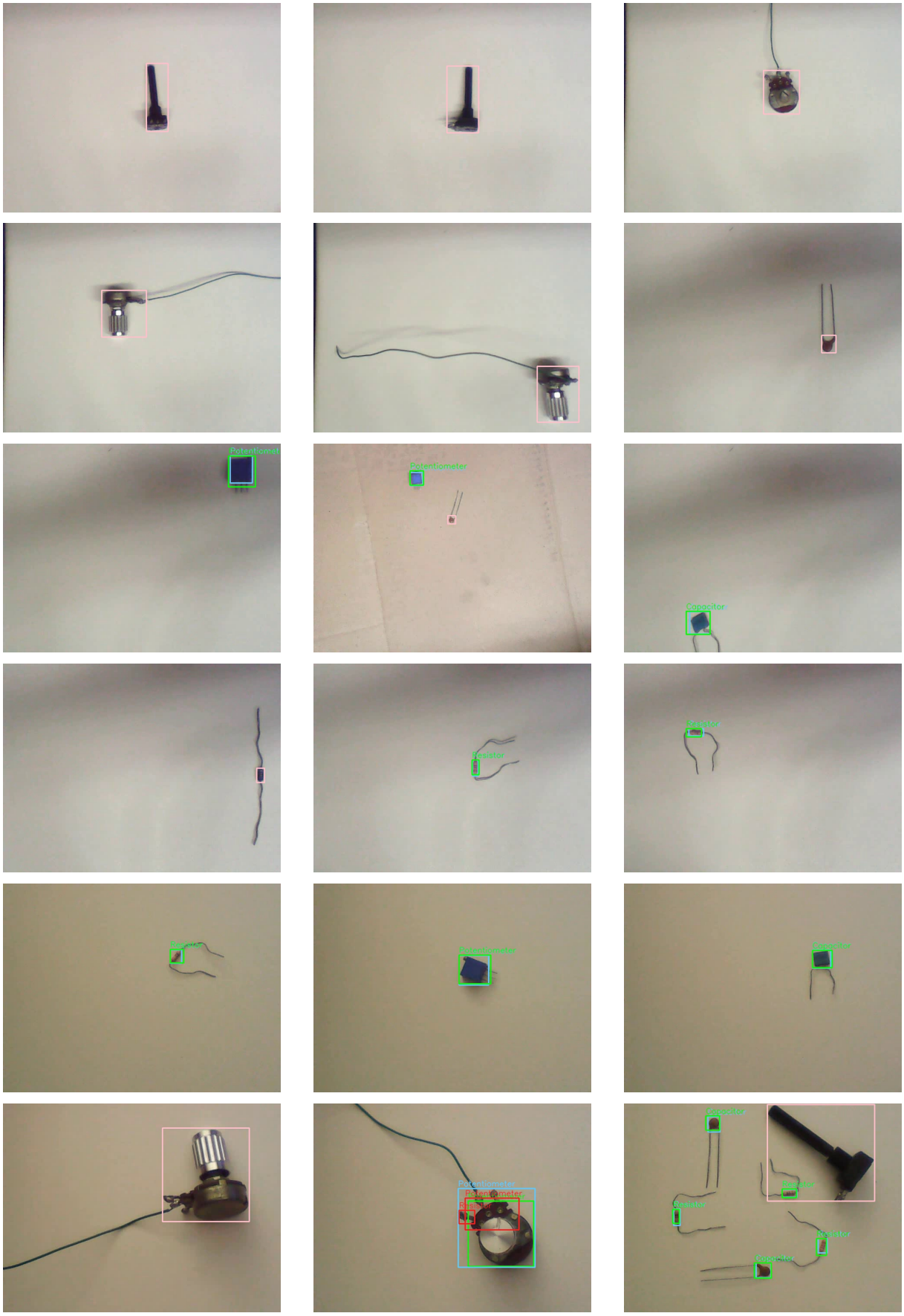


Figure A.25: Verification set and predictions for the R-CNN model trained with imbalanced dataset (part 2 of 6).

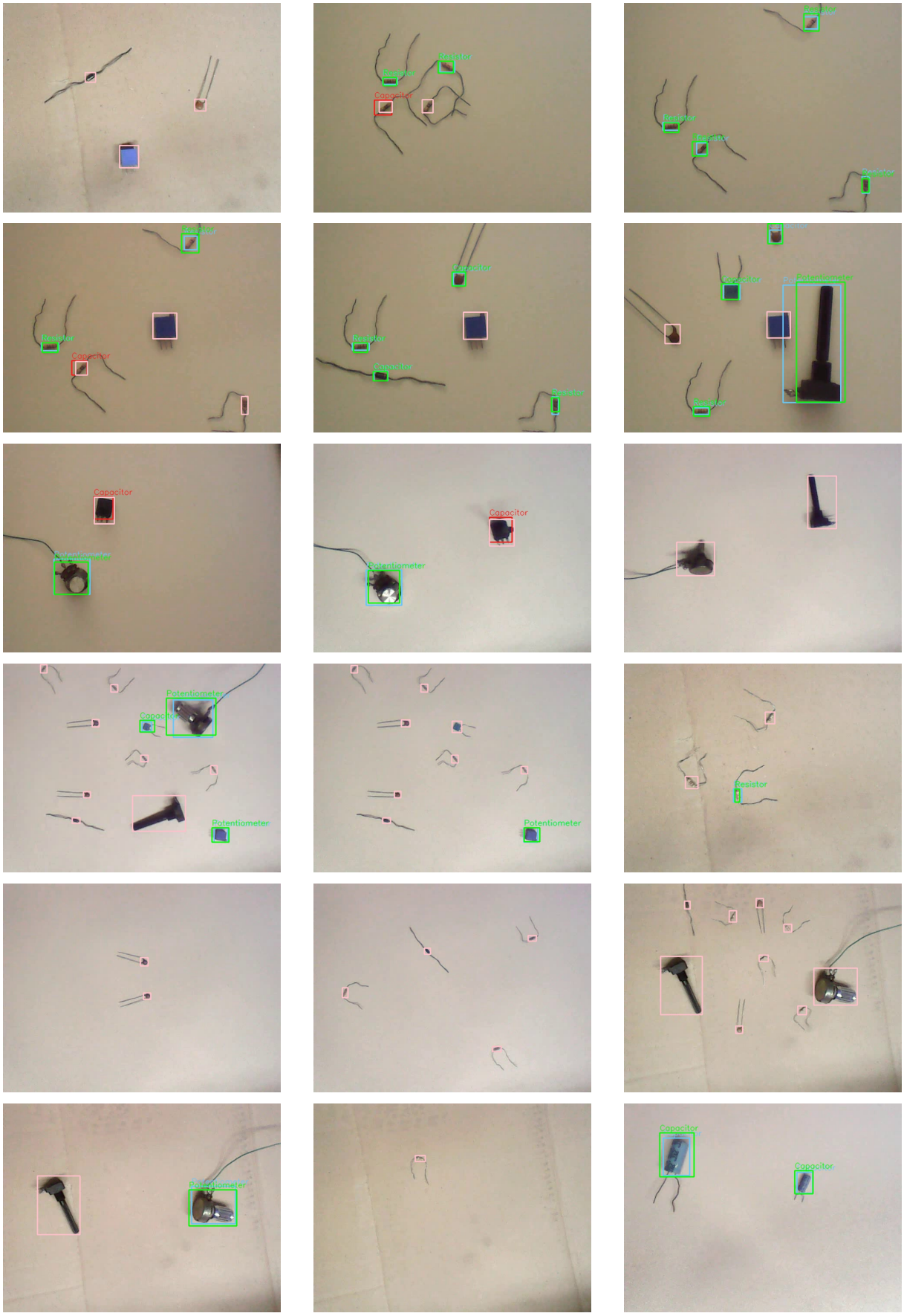


Figure A.26: Verification set and predictions for the R-CNN model trained with imbalanced dataset (part 3 of 6).

Figure A.27: Verification set and predictions for the R-CNN model trained with imbalanced dataset (part 4 of 6).

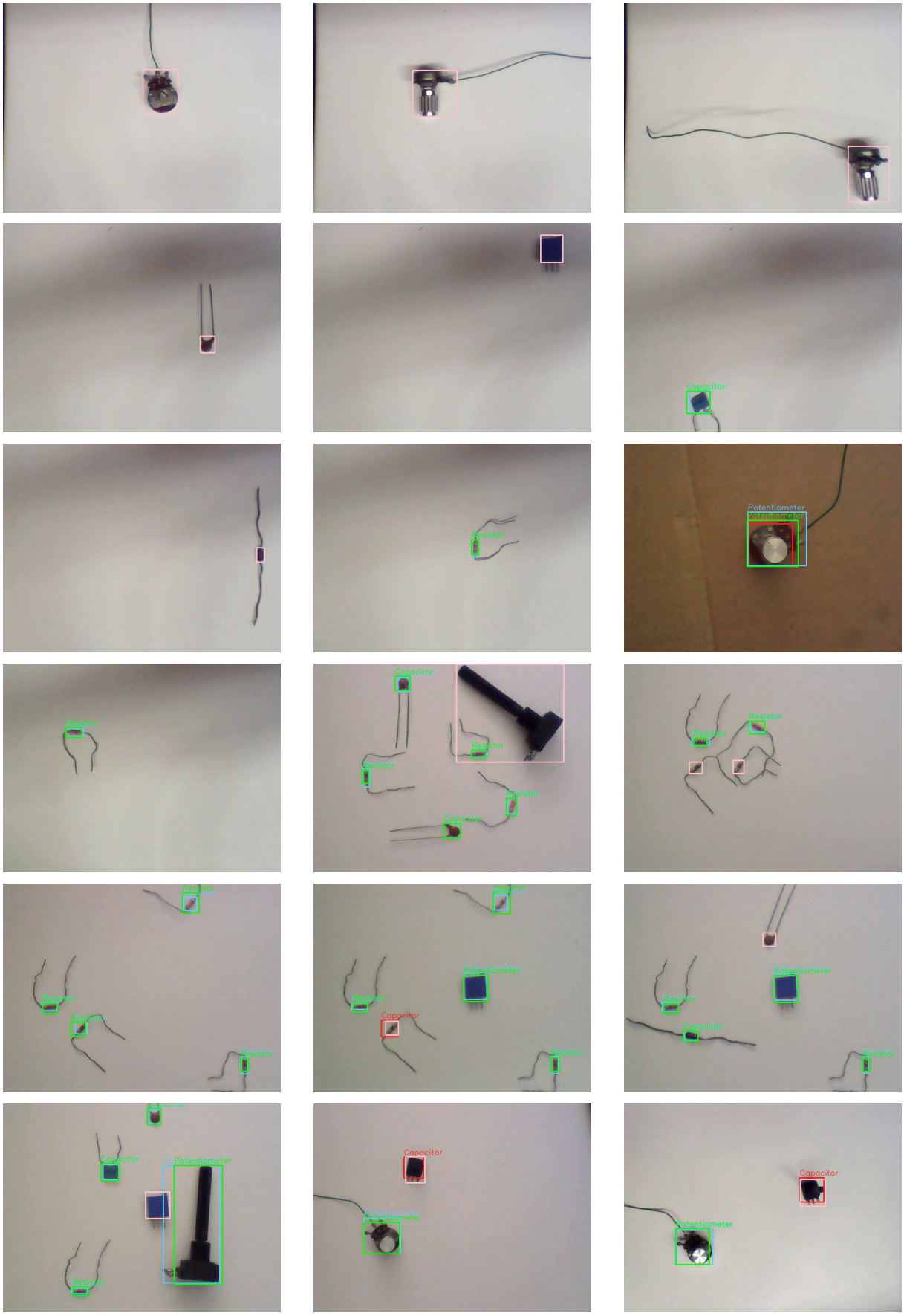


Figure A.28: Verification set and predictions for the R-CNN model trained with imbalanced dataset (part 5 of 6).

Figure A.29: Verification set and predictions for the R-CNN model trained with imbalanced dataset (part 6 of 6).