mathematics

MDPI

Article

Real-Time Detection of Unrecognized Objects in Logistics Warehouses Using Semantic Segmentation

Serban Vasile Carata [1,*], Marian Ghenescu [1,2] and Roxana Mihaescu [1]

[1] Softrust Vision Analytics, 107A, Oltenitei Avenue, 041303 Bucharest, Romania
[2] ISS—Institutul de Stiinte Spatiale, 409, Atomistilor Street, 077125 Magurele, Romania
* Correspondence: serban.carata@softrust.ro; Tel.: +40-737-222-781

Abstract: Pallet detection and tracking using computer vision is challenging due to the complexity of the object and its contents, lighting conditions, background clutter, and occlusions in industrial areas. Using semantic segmentation, this paper aims to detect pallets in a logistics warehouse. The proposed method examines changes in image segmentation from one frame to the next using semantic segmentation, taking into account the position and stationary behavior of newly introduced objects in the scene. The results indicate that the proposed method can detect pallets despite the complexity of the object and its contents. This demonstrates the utility of semantic segmentation for detecting unrecognized objects in real-world scenarios where a precise definition of the class cannot be given.

Keywords: object detection; semantic segmentation; UPerNet; convolutional neural networks; background subtraction; warehouse pallet

MSC: 65D19; 68T45

Citation: Carata, S.V.; Ghenescu, M.; Mihaescu, R. Real-Time Detection of Unrecognized Objects in Logistics Warehouses Using Semantic Segmentation. *Mathematics* **2023**, *11*, 2445. https://doi.org/10.3390/math11112445

Academic Editors: Vladimir V. Arlazarov, Konstantin Bulatov and Konstantin Kozlov

Received: 24 March 2023
Revised: 24 April 2023
Accepted: 24 May 2023
Published: 25 May 2023

1. Introduction

In recent years, the applications of machine learning and computer vision have extended to encompass a vast array of industries. Specifically, logistics warehouses have benefited considerably from the deployment of these technologies, as they enhance operating efficiency and safety. Unfortunately, despite their numerous benefits, these technologies still struggle to identify and detect unrecognized things in the actual world.

The potential hazard posed by abandoned or misplaced products is one of the most important concerns in logistics warehouses. These things can pose a serious risk to employees and impede the warehouse's general operations. Hence, it is imperative to detect these unidentified things swiftly and precisely.

To overcome this issue, we offer a novel semantic segmentation method for finding unknown items in images. Semantic segmentation is a technique that adds a semantic label to each pixel of an image, indicating the entity to which it belongs. Particularly in the realm of computer vision, this method has proven to be highly effective for picture recognition tasks.

Our suggested method identifies freshly introduced objects in a scene based on their position and their stationary behavior. Using the state-of-the-art UPerNet model, which is renowned for its remarkable performance in semantic segmentation tasks, we developed a semantic segmentation mask.

Establishing a baseline by assessing the scene devoid of any items is the first stage in our methodology. Once a baseline has been constructed, the scene is monitored for any changes to the semantic segmentation mask. If a new object is added into the scene, its position and behavior is used to identify it.

To further improve the precision of our system, we classify freshly identified objects using machine learning methods. This enables our technique to distinguish between various object kinds, such as pallets and boxes, and label them appropriately.

The results of testing our proposed strategy in a real-world scenario were really encouraging. Our technique was able to accurately detect and classify unknown objects in situations when the object class was not explicitly defined. Existing approaches frequently struggle to identify unidentified items; therefore, this is a huge advance.

In conclusion, our method illustrates the effectiveness of semantic segmentation in recognizing unknown objects in real-world circumstances. The combination of position, stationary behavior, and machine learning techniques enables our system to effectively detect and classify newly introduced objects, hence enhancing warehouse security and operating efficiency. We feel that our technology has a great application potential in businesses where the detection of unidentified objects is a major issue.

The paper's outline is as follows: We introduce the problem of pallet detection and its significance in the introduction. Section 2 then reviews the related work on pallet detection. In Section 3, we discuss the theory of neural network architectures and optimization techniques, including stochastic gradient descent (SGD), momentum method, and adaptive gradient descent (AdaGrad). Semantic segmentation and convolutional neural networks are discussed in this section as well. In Section 4, we present a method for detecting pallets that consists of background subtraction, tracking, and majority vote. Section 4 explains in detail our proposed algorithm. In Section 5, we discuss performance evaluation metrics, including the evaluation of semantic segmentation models, and provide dataset examples. There, we also present experimental results, and in Section 6, we conclude on the effectiveness of our proposed pallet detection method.

2. Related Work

2.1. Problem Overview

Today, logistics centers surround urban areas and employ a substantial number of individuals. The need to strengthen safety at these institutions is evident, but sadly, the expense of doing so is also a factor. Falling palettes from the shelves, lost palettes on passageways between shelves, and pallets stopping forklifts are some of the primary work security concerns in these locations.

Existing commercial hardware-based solutions for the first issue are based on infrared beams. They are extremely dependable but expensive, particularly for large distribution hubs.

We propose a technique that utilizes the existing monitoring system to detect all of these issues reliably. To do this, the suggested algorithm must be independent of perspective or lighting.

2.2. Dataset Problem

Due to the delicate and highly competitive nature of the logistics industry, public datasets are not readily available. The majority of research in this sector is conducted under strong nondisclosure agreements (NDAs), so photographs that could expose sensitive information are not released to the public. As a result, we had to independently acquire and annotate images. Regrettably, we were also bound by a stringent NDA.

In order to address this issue, we conducted the initial training for the semantic segmentation on a general purpose dataset and the fine-tuning on a smaller sample from our own dataset. For the initial step, we chose the ADE20k [1].

The ADE20k dataset is a large-scale dataset for scene parsing consisting of over 20,000 images with more than 150 object categories labeled at the pixel level. The images in the dataset depict a variety of indoor and outdoor settings including offices, bedrooms, streets, and parks, among others. The dataset is intended for computer vision tasks requiring scene comprehension, including object detection, semantic segmentation, and image captioning. The images in the dataset were gathered from multiple sources, including Flickr and Google Images, and annotated by humans using an interactive segmentation tool. Annotations consisting of object labels, object parts, and scene attributes provide rich and detailed scene information. The ADE20k dataset has been extensively utilized

in computer vision research, leading to the creation of cutting-edge algorithms for scene parsing and related tasks.

2.3. Existing Methods

Classic object classification methods such as decision trees, hidden Markov chains, and support vector machines have been widely used in object classification tasks. Nevertheless, they cannot be applied in the scenario proposed in this paper. The first issue is the unknown identity of the objects. The primary purpose of the algorithm is to identify any object that may represent a risk in the work environment, not just fallen pallets. The classic methods could not identify any unknown object, regardless of its shape and structure, or without seeing it before. Further, there are many environmental changes, both regarding the background movements and the lighting conditions, which lead to unsatisfactory results in the case of the classic classification methods mentioned previously.

As shown in [2], several algorithms have been implemented and trained to detect pallets in industrial warehouses. This paper compares three convolutional neural network architectures to address the pallet detection task.

Faster R-CNN [3] is based on a region proposal concept, which was first introduced with R-CNN [4]. In 2015, Girshick introduced Fast R-CNN, which generated region proposals directly on a feature map computed on the whole image. Ren et al. introduced the region proposal network (RPN), which used a fully convolutional network for feature extraction that output a feature map of the input image. After ROI pooling, a classifier determined the class.

SSD [5] and YOLO [6] propose a one-step object detection network that does not require region proposals. SSD is comprised of a first convolutional extraction network of the feature map, several convolutional layers that acquire multiscale feature maps, and a final component that generates the estimated offset and confidence for each class. Redmon et al. introduced YOLO in 2016, which improved the detection accuracy, but its primary weakness was its inability to detect small objects.

A CNN evaluation of pallets and pallet pockets was used to select the final pallet proposals by employing a decision block, as shown in Figure 1. The heuristic rules utilized by the decision-making phase were as follows: if a detected pallet's front side exceeds a threshold area, a pallet proposal is created only if the pallet's front side contains exactly two pockets; otherwise, the pallet's front side is discarded. If the area threshold is fixed to 150×103 pixels, the decision block always accepts the image as a palette, regardless of whether or not it contains two compartments. This decision rule was motivated by the necessity of identifying all pallet components in order to perform secure pallet forking operations on close and approximately frontal pallets. Additionally, it was necessary to model all potential pallets for AGV navigation.

Figure 1. CNN detects the front sides of pallets (yellow boxes) and the pallet pockets (red boxes) [2].

Work in this field has also been done more recently, as highlighted in [7], also based on SSD and YOLO models.

The solutions now available in the literature are geared toward assisting forklift operators and even autonomous forklifts to recognize and manipulate palettes. Our objective is to detect abandoned, misplaced, or fallen palettes in order to enhance security.

The approaches currently being investigated for palette detection are inapplicable to our needs, as they rely on clearly detecting and observing the palette structure. Our objective is to discover the palettes with the surveillance system. This shift in camera position and perspective necessitates a radical adjustment to the strategy and algorithms employed.

2.4. Challenges in Pallet Detection Using Traditional Methods

Due to the complexity of the objects stacked on top of pallets, traditional object detection techniques such as you only look once (YOLO) [6] model and region-based convolutional neural network (R-CNN) [8] are incapable of accurately recognizing pallets from above. Preliminary investigations demonstrated that these networks did not converge when trained on such datasets. As shown in Figure 2, pallets are frequently covered with a variety of objects, such as boxes, barrels, bottles, motorcycles, and more. These objects can be of varying sizes, shapes, and textures, making it difficult for object detection algorithms to classify them accurately. In addition, pallets can be partially obscured by other objects or their surroundings, making it more difficult for object detection techniques to identify them. As a result, traditional methods may fail to accurately recognize pallets, making their monitoring and tracking in logistics warehouses difficult.

Figure 2. Pallets with different loads: (**a**) stacked bags; (**b**) large bag; (**c**) stacked boxes; (**d**) barrels; (**e**) stacked bottle cases; (**f**) stacked water bottles; (**g**) motorcycle bodies.

The high density of pallets in logistics centers presents a difficulty in detecting [9] misplaced or tipped-over pallets. Even if conventional object detection methods could

accurately detect pallets, it is challenging to distinguish between pallets in their designated locations and those that are not. This emphasizes the need for a specialized algorithm capable of distinguishing between misplaced or fallen-over pallets and the rest, thereby enhancing warehouse safety.

3. Theory Overview

In this section, we define the theoretical aspects used throughout the implementation process of the proposed method. First, we present a detailed description of neural networks and how they developed from perceptron to convolutional networks. Afterward, we outline the semantic segmentation process, focusing on the UPerNet model that represents the foundation of our proposed algorithm.

3.1. Neural Network Architectures

The architecture of an artificial neural network (ANN) [10] defines how the neurons are arranged and interconnected. Generally, a neural network has three types of layers in its composition. The first one is the input layer. It receives information from the external environment. Network inputs are usually scaled to maximum values. The normalization step increases the numerical precision of the mathematical operations performed by the network. After the input layer, a network can have one or more hidden layers. Most of the neural network's internal processing occurs at the level of these layers. At the end of the network, it is an output layer responsible for producing and presenting the final outputs of the network obtained due to the processing performed by the previous layers.

Furthermore, we outline several main architectures depending on how the neurons are interconnected.

The most straightforward layout is the perceptron [11]. It represents an element with a certain number of inputs, which calculates their weighted sum. For each entry, the weight can be either 1 or -1. Finally, this amount is compared with a threshold, and the output y is obtained according to Equation (1).

$$y = \begin{cases} 1 & \rightarrow \sum_{i=1}^{N} (\beta_i \cdot x_i) \geq \theta \\ \\ 0 & \rightarrow \sum_{i=1}^{N} (\beta_i \cdot x_i) < \theta \end{cases} \tag{1}$$

where N symbolizes the total number of inputs, x_i is the value of each input, β_i represents the value of the weight associated with input i, and θ is the decision threshold. The two output values are used to distinguish between two different classes. For a perceptron to classify as well as possible, the weights must be changed, and the threshold must be set to an appropriate value.

The perceptron is not a complete decision model, being able to distinguish only between two different classes. For this reason, the need for a complex network of perceptrons has occurred. These networks, called multilevel perceptron (MLP) networks [12], can solve classification problems. The perceptrons in the input layer make simple decisions by applying weights to the input data. In contrast, the ones in the intermediate layers apply weights to the results generated by the previous layer. In this way, the perceptrons in the intermediate layers make decisions more complex and abstract than those in the first layer. As the number of layers increases, the MLP network can make increasingly sophisticated decisions.

The purpose of MLPs is to approximate a mathematical function of the form

$$y = f(x; \theta), \tag{2}$$

and the goal of the network is to learn the parameter θ which leads to the best approximation of the desired function.

The network needs to use a cost function in the training stage [13], which depends on the values of the weights. The goal of any neural network is to minimize this cost function

using various optimization methods. The most used optimization methods are described in the following subsection.

One of the cost functions used is mean squared error (MSE), from (3).

$$C(w,b) = \frac{1}{2n}\sum_{x}||y(x) - a||^2 \tag{3}$$

where w represents the weights in the network, n is the number of inputs trained, and the vector a represents the vector of outputs when the vector x is at the network's input.

3.2. Optimizations Methods

Solving the training problem generally involves a lot of time and resources. Since this is a fundamental and costly problem, several optimization techniques have been developed [14]. Mainly, the optimization boils down to finding the parameter θ of the neural network so that the cost function $J(\theta)$ is minimal.

3.2.1. Stochastic Gradient Descent (SGD)

The classic gradient descent (GD) method [15] uses the entire training set to update the parameters, which involves high costs in terms of time and required computing power. In contrast, using the SGD algorithm [16] involves using only a few training examples or even a single example. Thus, this method leads to a smaller required memory and a high convergence speed.

The GD algorithm updates the θ parameter according to

$$\theta = \theta - \alpha \cdot \nabla_\theta \cdot E[J(\theta)] \tag{4}$$

In the case of SGD, the parameters' gradient is calculated using only some of the training examples, thus obtaining an update of the form

$$\theta = \theta - \alpha \cdot \nabla_\theta \cdot J(\theta; x^{(i)}, y^{(i)}) \tag{5}$$

where α represents the learning rate, and $x^{(i)}, y^{(i)}$ is a pair from the training set. The learning rate decreases linearly until iteration τ, then it remains constant, according to Equation (6).

$$\alpha_k = (1 - \epsilon) \cdot \alpha_0 + \epsilon\alpha_\tau \tag{6}$$

$$\epsilon = \frac{k}{\tau} \tag{7}$$

where α_k represents the value of the learning rate at iteration k, and α_0 is the initial value of the learning rate.

3.2.2. Momentum Method

While the SGD method is popular, it can lead to relatively slow learning. The momentum method was created to speed up the learning process. The parameter update is computed according to

$$v = \gamma \cdot v + \nabla_\theta \cdot J(\theta; x^{(i)}, y^{(i)}) \tag{8}$$

$$\theta = \theta - v \tag{9}$$

The momentum method introduces variable v, which represents the velocity vector that has the same size as the θ vector. The parameter γ determines the rate at which the contributions of the previous gradients decay exponentially. This parameter belongs to the range $(0, 1]$.

3.2.3. Adaptive Gradient (AdaGrad)

The AdaGrad (adaptive gradient) method [17] is an extension of the SGD algorithm. It adapts the learning rates of all model parameters individually. Unlike the SGD method, the AdaGrad method scales the learning rates to the root of the sum of all previous gradients. Let N be the number of examples from the training set used. In the first step, this method computes the size

$$g = \nabla_\theta \cdot J(\theta; x^{(i)}, y^{(i)}) \tag{10}$$

and updates the sum

$$r = r + g \odot g \tag{11}$$

where $\odot$ represents the element-by-element multiplication of the two vectors.

Finally, AdaGrad uses an update relation for the parameter θ similar to the one used by the SGD method in Equation (5):

$$\theta = \theta - \frac{\alpha}{\delta + \sqrt{r}} \cdot \nabla_\theta \cdot J(\theta; x^{(i)}, y^{(i)}) \tag{12}$$

where δ is a small constant that aims to avoid division by zero of the learning rate.

The AdaGrad method performs updates with a more significant step for the less frequent parameters and a minor step for the more frequent ones.

3.3. Convolutional Neural Networks

Convolutional neural networks (CNN) [18] are similar to the previously presented neural networks. They are composed of several layers of neurons that have various associated weights. These networks receive input images, leading to a three-dimensional network architecture. The main difference between fully connected networks and CNNs is the type of input data they accept.

The input layer receives the pixel values from the image for the three color channels: R—red, G—green, and B—blue, respectively. The intermediate layers can be of several types, among which the most important ones, convolutional layers and pooling layers, are described below. Finally, the last layer is a fully connected type layer. Its purpose is to calculate the results for each class. Thus, a CNN-type network transforms the pixel values from the input image into probabilities belonging to all classes. A loss function (for example, SVM or SoftMax) is applied to the last neural layer and measures the network's performances and the outputs' correctness.

3.3.1. Convolutional Layers

The primary process that takes place in a neural network is affine transformations [19]. The input receives a vector, which is then multiplied by a matrix to produce the output. This transformation can be applied to any input data. Regardless of their size, data can be put into a vector before the transformation occurs.

Discrete convolution is a linear transformation that preserves the input data's structure and considers how those data are ordered. Only a few units from the input data structure are used to calculate a unit from the output data structure. In addition, discrete convolution reuses parameters, with the same weights being applied to multiple units in the input data structure. The kernels or filters used in the convolution operation are spatially small (along the width and height of the image) but extend through the entire depth of the input data volume. At each location, the product of each kernel element and the input element it overlaps is calculated. All the products are summed, resulting in the output value at the current location.

The convolution made at the level of these layers can have N dimensions. The collection of kernels defining a discrete convolution has a form corresponding to one of

the permutations $(n, m, k_1..., k_N)$, where n is the number of output feature maps, m is the number of input feature maps, and k_j is the size of the kernel along the j-axis.

A convolution layer results in a whole set of filters, each producing a two-dimensional feature map. All these maps are stored along the third dimension, depth, and thus produce the output of the convolutional layer. Neuron connections are local in space but complete over the entire depth of the input volume. Since the images have large sizes, each neuron favors being connected to a specific region in the input data volume, thus reducing the spatial dimensions—width and height. The regions are chosen to have the exact same dimensions as those of the filter used and are called the neuron's receptive field. In contrast, the third dimension remains unchanged. The data's dimensions at the convolutional layer's output are calculated according to (13).

$$O = \frac{W - F + 2P}{S} + 1 \tag{13}$$

where W represents the size of the input data, F is the size of the receptive field of the neurons, which is equivalent to the size of the filters used in the convolution operation, S represents the step used, and P is the number of padding zeros used.

According to (13), as the step S used has a bigger value, the output produces a smaller quantity of data. Furthermore, the parameter P allows one to control the quantity of output data too. In general, a number of zeros is used so that the output data are the same size as the input data. The values of the parameters S and P should be chosen such that applying Equation (13) yields an integer value for the size of the output data.

3.3.2. Pooling Layers

In addition to convolution operations, pooling operations form another essential building block in CNNs. These operations reduce the dimensions of feature maps by using certain operations to summarize each subregion, such as averaging or the maximum value in each subregion. These operations work similarly to the convolution relation. A window of various sizes is hovered over the input data, selecting one subregion at a time. The content of this region is processed according to a pooling function. In a neural network, the role of pooling layers is to ensure the invariance to small translations of the input. The most common pooling operation is to choose the maximum value of each subregion in the input map.

In general, it is typical to introduce a pooling layer between several consecutive convolutional layers in a CNN network to reduce the number of required parameters and the amount of computation in the network. The pooling layer acts independently on each region in the input data and resizes it spatially. The depth does not change. The most common form is a set of filters of dimensions 2×2, applied with a step equal to two. In this case, the pooling operation is applied to four numbers in each region. Thus, the volume is reduced by 75%.

At the output of this layer, a volume of data is obtained with the following dimensions:

$$W = \frac{W_i - F}{S} + 1 \tag{14}$$

$$H = \frac{H_i - F}{S} + 1 \tag{15}$$

$$D = D_i \tag{16}$$

where W_i, H_i, and D_i are the input data sizes, S is the step at which the filters are applied, and F is the filter size. Equations (14)–(16) show that the third dimension remains unchanged while the width and height shrink.

Some architectures have no such layers but only use successive convolutional layers. To reduce the data size, a network can use convolutional layers with larger steps instead of pooling layers.

3.4. Residual Neural Networks

As stated in this section, as the number of layers in a network increases, any function can be approximated, regardless of its complexity. For this reason, vast neural networks with hundreds of layers have been implemented. One of the main problems of these networks is the vanishing or exploding gradient.

In the first case, the gradient can decrease exponentially towards zero. In this circumstance, optimization methods such as gradient descent evolve slowly toward a solution. In the second case, the gradient grows exponentially, taking very high values. To address this problem, residual neural networks were implemented, formed by a series of residual blocks. The concept of residual networks was first presented in [20] and was later augmented in [21].

The residual blocks are based on a method called skip-connections. This method assumes that the input data in the block are not passed through all the layers and are added directly to the block output. Figure 3 presents different architectures of a residual block.

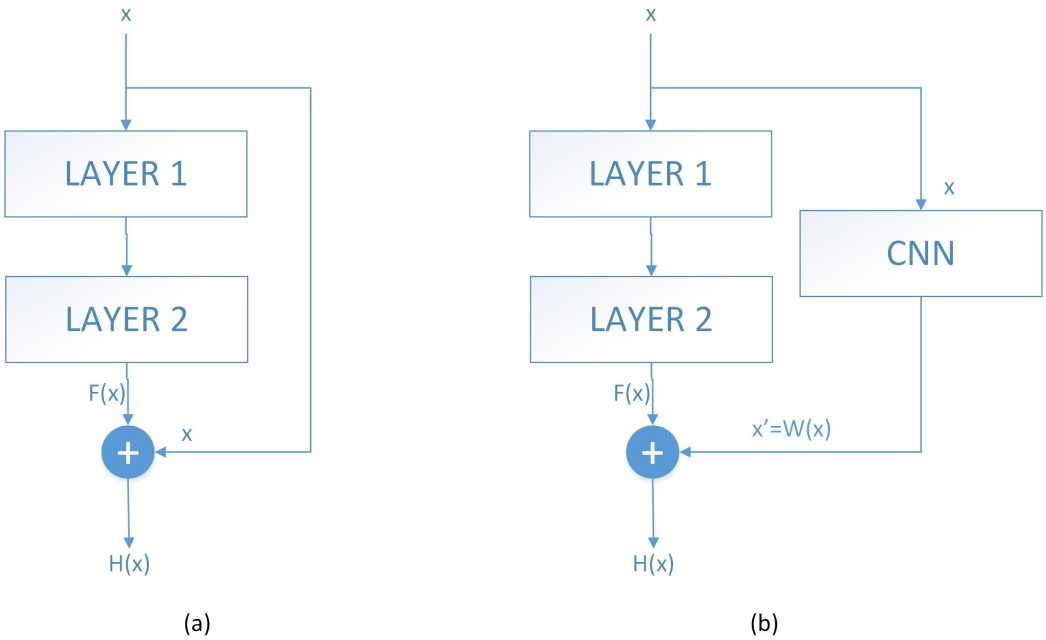

Figure 3. Architectures of residual blocks.

Figure 3a illustrates the classic architecture of a block, where the input x and the output $F(x)$ have the same dimensions.

Let x be the input to the residual block and $H(x)$ be the desired output. $F(x)$ represents the function learned by the neural network and represents the output of the block when the input is x. In the case of the residual block, the new output is a sum between the input to the block and the output of the layers, as can be seen from Equations (17)–(20). The function that the network has to learn is, this time, a residual function computed as the difference between the desired output and input (Equation (21)).

$$x \rightarrow \text{Input} \tag{17}$$

$$H(x) \rightarrow \text{Correct Output} \tag{18}$$

$$F(x) \rightarrow \text{Network Output} \tag{19}$$

$$H(x) = F(x) + x \tag{20}$$

$$F(x) = H(x) - x \tag{21}$$

The second architecture shown in Figure 3b is used when the output $F(x)$ and the input x have different sizes and cannot be summed. To solve the dimensions issue, the input has to go through one or more layers to reach the required size. In this way, a linear transformation is applied to the input, which can be achieved by using convolutional layers or just by filling the input with zeros until it reaches the size of the output $F(x)$. This time, the new output of the residual block is of the form

$$H(x) = F(x) + x' \tag{22}$$

where $x' = W(x)$ is the linear transform.

Finally, several residual blocks are combined to obtain a residual neural network.

3.5. Semantic Segmentation

Semantic segmentation is a computer vision task that involves labeling each pixel in an image with a semantic label. Semantic segmentation aims to comprehend the pixel-level content of an image, providing a more comprehensive understanding of the scene than traditional object detection techniques. Deep neural networks, such as convolutional neural networks (CNNs) that are trained to predict the class of each pixel in an image, are typically used to perform semantic segmentation. Semantic segmentation results in a dense label map that assigns a class label to each pixel in the image, providing a comprehensive understanding of the scene's objects and their boundaries. Semantic segmentation has numerous applications, including autonomous driving, scene comprehension, and medical imaging.

UPerNet [22] is a deep learning model for semantic segmentation that fuses multiscale contextual information using a top-down and bottom-up mechanism. By employing a hierarchical feature fusion mechanism that effectively combines the strengths of both top-down and bottom-up pathways, UPerNet is an improvement over existing models such as fully convolutional network (FCN) [23]. To generate the final semantic segmentation map, the UPerNet model employs a deep neural network architecture that combines a ResNet-style network [20] with an upsampling mechanism. It has been demonstrated that the UPerNet model produces high-quality results for semantic segmentation, making it a suitable option for detecting unrecognized objects in images.

For a better understanding of the UPerNet model, we define the following terms:

- **Top-down and bottom-up mechanisms**: UPerNet captures high-level semantic information and low-level details using a top-down and bottom-up mechanism. The top-down pathway employs a pyramid pooling module to extract multiscale context information, whereas the bottom-up pathway employs dilated convolutions to maintain spatial resolution.
- **ResNet-style network**: UPerNet's backbone is a ResNet-style network. Pretrained on the ImageNet dataset, the ResNet-style network provides the model with rich feature representations that can be tuned for semantic segmentation.
- **Upsampling mechanism**: To generate the final semantic segmentation map, UPerNet employs an upsampling mechanism. The upsampling mechanism combines the top-down and bottom-up features and employs transposed convolutions to improve the spatial resolution of the features.
- **Hierarchical feature fusion**: the hierarchical feature fusion mechanism enables UPerNet to capture high-level and low-level semantic information, yielding high-quality semantic segmentation maps.
- **Performance**: UPerNet has demonstrated state-of-the-art performance on multiple benchmark datasets for semantic segmentation, including PASCAL VOC [24] and Cityscapes [25]. This makes UPerNet a suitable option for detecting unidentified image objects.

The following can be seen in Figure 4.

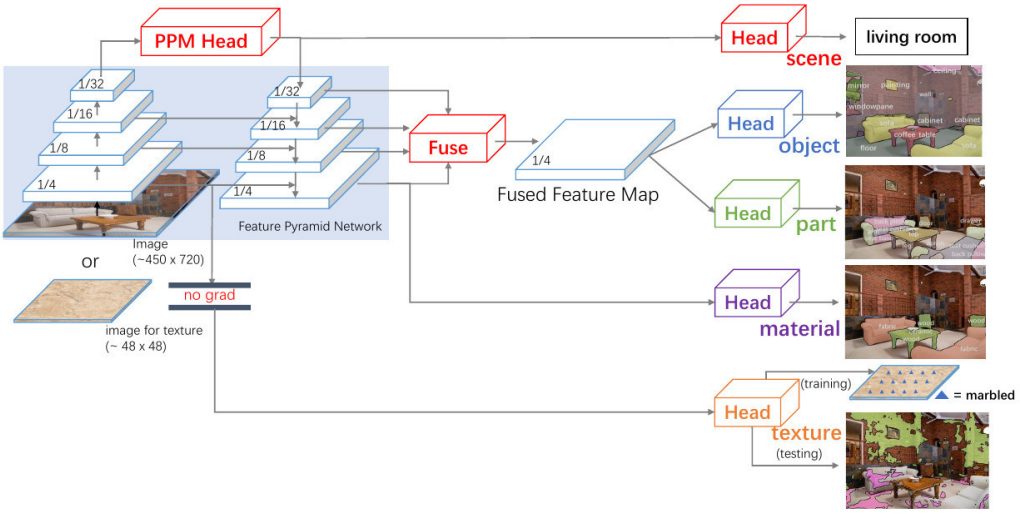

Figure 4. UPerNet framework for Unified Perceptual Parsing [22].

Top-left: feature pyramid network (FPN) [26] is a common architecture for object detection and semantic segmentation. FPN is designed to extract multiscale contextual information from an image, which is essential for accurately detecting objects of various sizes.

In the FPN architecture, the pyramid pooling module (PPM) [27] is a component that is added to the final layer of the backbone network. The PPM module effectively captures both high-level semantic information and low-level details by combining features from different scales. The PPM module is incorporated into the top-down branch of the FPN architecture, where it contributes to the final feature maps used for object detection or semantic segmentation.

The combination of the FPN architecture and the PPM module enables the model to extract multiscale information from an image, which is essential for accurately detecting objects of varying sizes. By adding the PPM module to the final layer of the FPN architecture's backbone network, the FPN + PPM model is able to effectively capture both high-level semantic information and low-level details, resulting in enhanced performance for object detection and semantic segmentation tasks.

This depicts a multihead architecture for semantic segmentation, with each head designed to extract specific semantic information from an image.

The scene head is attached to the feature map immediately after the pyramid pooling module (PPM), as information at the image level is more suitable for scene classification. This head is in charge of recognizing the overall scene and classifying it into various categories, such as indoor or outdoor scenes.

The object and part heads are affixed to the feature map in which all the layers generated by the feature pyramid network have been combined (FPN). These heads are responsible for detecting objects and their parts in the image, respectively. The object head provides coarse-grained information about the location of an object, whereas the part head provides fine-grained information about object parts.

The material head is attached to the highest-resolution feature map in the FPN. This head is responsible for identifying various material properties, including metal, glass, and cloth.

The texture head is connected to the Res-2 block in the ResNet [20] architecture and is fine-tuned after the network has completed training for other tasks. This head is responsible for capturing texture data, such as the roughness, smoothness, and patterns of image objects.

The use of multiple heads in this architecture enables the model to capture multiple levels of semantic information, resulting in a more in-depth and thorough comprehension of the image content.

4. Proposed Method

4.1. Proposed Algorithm

Using a block diagram, the following section illustrates the proposed algorithm. The block diagram provides a clear and concise illustration of the algorithm's various components and their interactions. In addition, the section contains a pseudocode implementation of the algorithm, which describes the algorithm's operations and decision-making procedure in detail. The block diagram and pseudocode together provide a comprehensive understanding of the proposed method and its operation. These visual aids clarify the algorithm's complex operations and demonstrate its functionality in a concise and clear manner.

As input data, the proposed algorithm works only on videos. While the first part of the algorithm is applied to the static images extracted from the footage, the following stages such as tracking are dependent on the temporal component.

In the first step of the proposed algorithm, the input image is preprocessed. This is achieved by running an image through a background subtraction algorithm [28]. The mathematical model of this method and its detailed description are presented in the following subsection. In addition to producing a clean image as input for the semantic segmentation algorithm, this step extracts the bounding boxes of all moving objects within the scene. By eliminating the static background, the algorithm is able to concentrate on the static objects, making them easier to identify and segment. This preliminary step is essential for ensuring the consistency of the subsequent semantic segmentation process over time and enhancing the algorithm's overall performance.

In the second step of the proposed algorithm, it is determined whether or not the semantic segmentation map needs to be updated. If an update is deemed necessary, the current image is subjected to the semantic segmentation algorithm, and the mean segmentation map is updated using a majority vote approach [29]. In this method, the segmentation result that occurs most frequently for each pixel is selected as the updated map. If no update is necessary, the algorithm continues without performing the semantic segmentation. Periodically, the map is revised to ensure that it remains accurate and reflects the current landscape, by adding a new vote to the majority vote algorithm. The updated map provides a comprehensive depiction of the floor, as it is routinely revised to maintain its accuracy and keep up with environmental changes.

In the third step of the proposed algorithm, motion areas detected in the first step are analyzed using a tracking algorithm. The tracker is responsible for maintaining the paths of moving objects and identifying which of these paths have remained stationary and inactive for a predetermined amount of time, such as one minute. Once a stationary track is identified, it is considered a potential object of interest and sent to the next stage of the algorithm. This step is essential for reducing false-positive detections by eliminating tracks that are merely noise or transient motions. Instead, the algorithm focuses on objects that are potentially significant and remain in the scene for an extended amount of time. The tracking algorithm is essential for ensuring the accuracy and efficacy of the overall algorithm, as it identifies objects that require additional analysis and consideration. In this algorithm, we use the centroid tracking method, which is presented in detail later in this section.

In the next phase of the proposed algorithm, a potential object of interest is analyzed further to determine if it is a class that is unknown. On the current image, the semantic segmentation algorithm is applied, and its output map is compared to the historical map. This comparison, along with the location of the potential object of interest, aids in determining whether the area of interest belongs to an unknown class or is expected to be present. If an unknown or unexpected class is detected at the location of the tracked object, it is safe to assume the presence of the target class.

The comparison of the maps, in conjunction with the location of the potential object of interest, provides a reliable method for detecting unidentified objects. The detection is based on the likelihood that unknown objects will be of a different class than the expected

objects in the scene, and that their position will differ from that of the expected objects on the historical map.

The proposed algorithm can be implemented through pseudocode (Algorithm 1) and represented through a block diagram as in Figure 5. The pseudocode provides a clear and concise description of each step in the algorithm, making it easy to understand and implement. On the other hand, the block diagram provides a visual representation of the flow of the algorithm, helping to simplify and clarify the overall process.

The most significant contribution of this paper is the novel application of semantic segmentation techniques to detect objects that were not present during the training phase of the algorithm. This renders the proposed algorithm extremely resistant to novel circumstances and environments.

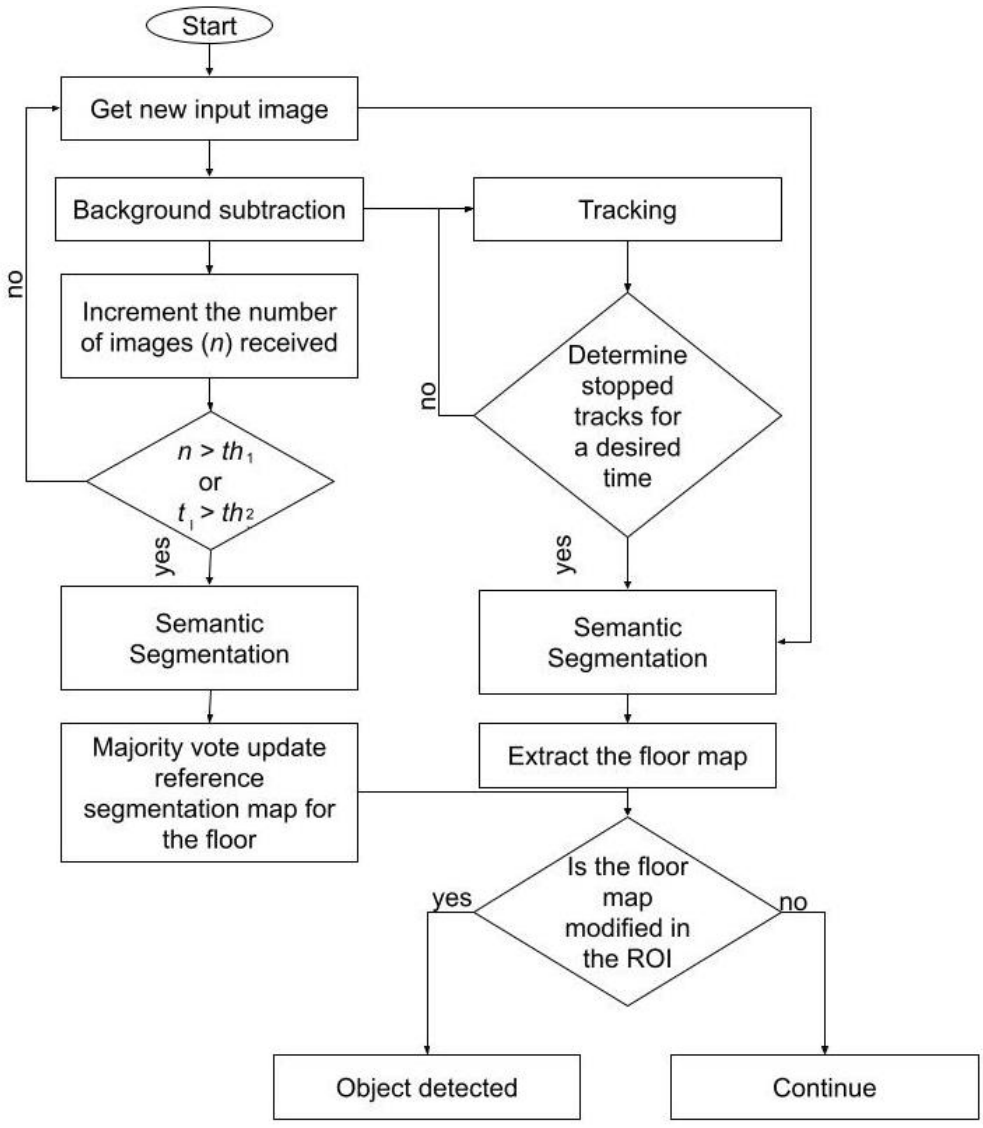

Figure 5. Proposed method.

Algorithm 1 Proposed method

$I \leftarrow InputImage$
$BG, MM \leftarrow BackgroundSubtraction(I)$
$V_M \leftarrow GetBoundingBoxes(MM)$
if $i \leq th_1 | now - tm < th_2$ **then**
  $Tmp_{Map} \leftarrow SemanticSegmentation(BG)$
  $Map \leftarrow MajorityVote(Tmp_{Map})$
  $FlorMap \leftarrow LabelFilter(Map)$
end if
$Tracks \leftarrow ObjectTracking(V_M, now)$
$StopedTracks \leftarrow GetStopedTraks(Tracks, now)$
if $len(StopedTracks) > 0$ **then**
  $Map \leftarrow SemanticSegmentation(I)$
  $FlorMap_{tmp} \leftarrow LabelFilter(Map)$
  $FlorChange = FlorMap - FlorMao_{tmp}$
  $n \leftarrow 0$
  while $n < len(StopedTracks)$ **do**
    $Nz = CountNonZeros(StopTracks(n), FlorChange)$
    if (**then** $Nz > 0$)
      $ObjectOfInterest \leftarrow StopTracks(n)$
    end if
    $n \leftarrow n + 1$
  end while
end if

4.2. Background Subtraction

The background subtraction method (BSM) represents one of the most used algorithms for detecting moving objects in a video stream. After applying this method, a mask of the foreground objects is obtained. This mask has the exact dimensions as the input image, with the foreground objects being white and the background black. The mathematical formulation of background subtraction can be described as follows:

Let $I(x, y)$ be the current frame and $I_{bg}(x, y)$ be the estimated background model. The goal of background subtraction is to find the pixels that correspond to the foreground object. This can be achieved by subtracting the background model from the current frame:

$$F(x, y) = |I(x, y) - I_{bg}(x, y)| \tag{23}$$

where $F(x, y)$ is the difference between the current frame and the background model.

Finally, a thresholding function can be applied to $F(x, y)$ to segment the foreground object:

$$B(x, y) = \begin{cases} 255 & \text{if } F(x, y) > T \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

where $B(x, y)$ is the binary segmentation, with the foreground object in white and the background in black, and T is the threshold value. The choice of threshold value affects the performance of the background subtraction algorithm, with higher values leading to fewer false positives and lower values leading to fewer false negatives (Figure 6).

There are several ways to perform background subtraction. All the methods start from an estimated background model that is updated over time. Each method updates the background model differently.

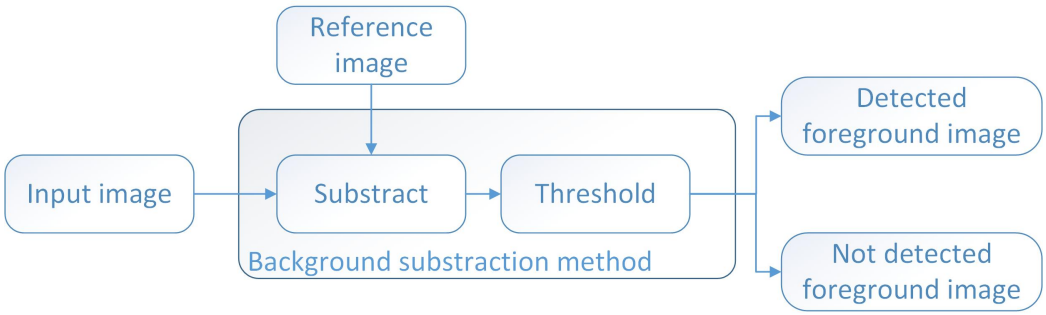

Figure 6. Background subtraction method.

One of the simplest and most frequently used techniques is *frame differencing*. In this case, the absolute difference of two consecutive frames is used to detect moving objects. Initially, the estimated background model is the first frame of the input video, and then, it is considered to be the previous frame. Mathematically it can be written as

$$F(x,y) = |I_t(x,y) - I_{t-1}(x,y)| \tag{25}$$

Frame difference is a relatively simple technique, sensitive to its threshold value. Although it is a technique that does not involve high costs in terms of computing power, other more complex methods of updating the background model were necessary.

One of the most well-known techniques is the *mixture of Gaussians* (MoG) technique. This background subtraction method was also used in this proposed method. The Gaussian model is a probabilistic model that starts from the hypothesis that a mixture of Gaussian distributions with unknown parameters can generate all pixel values. In practice, between three and five Gaussian distributions are used. The mathematical model is described as follows:

The multivariate Gaussian distribution is:

$$N(\mathbf{x}|\mu,\Sigma) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)} \tag{26}$$

At any time t, we know the history of each pixel (x_0, y_0):

$$X_1, \ldots, X_t = \{I(x_0, y_0, i) : 1 \leq i \leq t\} \tag{27}$$

The history of each pixel is considered to be a mixture of k Gaussian distributions, according to

$$P(X_t) = \sum_{i=1}^{K} \omega_{i,t} * N(\mathbf{X}_t|\mu_{i,t}, \Sigma_{i,t}) \tag{28}$$

where $\omega_{i,t}$ is the weight at time t, corresponding to the ith distribution, while $\mu i, t$ and $\Sigma_{i,t}$ are the mean and, respectively, the standard deviation of the ith distribution, at time t.

When a new frame appears, at time $t+1$, each pixel in the frame is compared with the Gaussian distributions by calculating the Mahalanobis distance:

$$((X_{t+1} - \mu_{i,t})^T \sigma_{i,t}^{-1} (X_{t+1} - \mu_{i,t}))^{0.5} < 2.5 \cdot \sigma_{i,t} \tag{29}$$

Two situations are possible:

1. If the pixel value X_{t+1} matches one of the distributions, that distribution is updated according to the relations (30) and (31).

$$\mu_{i,t+1} = (1-\rho)\mu_{i,t} + \rho X_{t+1} \tag{30}$$

$$\sigma_{i,t+1}^2 = (1-\rho)\sigma_{i,t}^2 + \rho(X_{t+1} - \mu_{i,t+1})^2 \tag{31}$$

where

$$\rho = \alpha N(t+1|\mu_{i,t}, \sigma^2_{i,t}) \tag{32}$$

and α represents the learning rate.

Finally, the weights are updated for all the distributions according to:

$$\omega_{i,t+1} = (1-\alpha)\omega_{i,t} + \alpha(M_{i,t+1}) \tag{33}$$

where $M_{i,t+1} = 1$ only for the matching distribution, and zero in all other cases.

2. If the pixel value does not match any of the distributions, the least likely Gaussian distribution is replaced with a new one with high variance, low weight, and mean $\mu_{t+1} = X_{t+1}$.

Finally, all distributions are ordered by the ω/σ value. The first B distributions are chosen as the background models, while the rest are foreground models, where

$$B = \text{argmin}_b(\sum_{i=1}^{b} \omega_i > T) \tag{34}$$

and T represents the threshold.

4.3. Tracking

Centroid tracking [30] is a prevalent object tracking algorithm that uses the center of an object's bounding box as a representative feature for tracking its position over time. Given an initial set of bounding boxes, the algorithm modifies the position of each bounding box in subsequent frames based on the centroid position. The centroid of a bounding box is the average position of all pixels contained within the box. In other words, the centroid of a bounding box is the average of the x and y coordinates of all pixels within the bounding box.

Mathematically, the centroid of a bounding box can be expressed as:

$$(x_c, y_c) = \frac{1}{N}\sum_{i=1}^{N}(x_i, y_i) \tag{35}$$

where N is the number of pixels within the bounding box, (x_i, y_i) are the x and y coordinates of pixel i, and (x_c, y_c) is the centroid position. The position of the centroid in subsequent frames is updated by using the same formula, with the updated set of pixels within the bounding box. This information can be used to update the position of the bounding box and track the object over time.

4.4. Majority Vote

The majority vote method is a straightforward and effective method for combining multiple results into a single, more precise result. It is commonly employed in computer vision and image processing for semantic segmentation tasks. The basic idea behind this method is to use the result that occurs most frequently for each pixel as the updated result.

Mathematically, let $S_{i,j}$ be the segmentation results for each pixel (i,j), and k be the number of segmentation results being combined. The majority vote method can be expressed as:

$$\hat{S}_{i,j} = \text{argmax}_{c=1}^{k} N_{i,j}(c) \tag{36}$$

where $\hat{S}_{i,j}$ is the final segmentation result for pixel (i,j), and $N_{i,j}(c)$ is the number of times class c was assigned to the pixel (i,j) across the k segmentation results.

4.5. Parameters Used in the Proposed Method

Several parameters were used to implement the proposed method. These parameters directly affected the final performance of the algorithm, both in terms of detection accuracy and the number of false alarms it generated. These parameters are described below.

The diagram from Figure 5 presents two parameters marked th_1 and th_2, respectively. Parameter th_1 is a preset number of frames after which the floor mask is updated; similarly, th_2 is a preset time after which the same action is taken. One of the two parameters must have a set value for the method to work. Their purpose is to decide when the reference map of the floor is updated. Both parameters can take values in the range $[0, \infty)$. If values are set for both parameters, the algorithm updates the map according to the smallest parameter. In the proposed method, we determined that triple the time to trigger an alarm was a good value for these thresholds. We used the following th_2 parameter:

$$th_2 = 3 \cdot 60 \text{ s} \tag{37}$$

Depending on the value of these parameters, false alarms can be prevented if a pallet is moved to a shelf and it changes the floor map. As the value of the parameters decreases toward 0, the map is continuously updated, and the algorithm does not generate any alarm. Otherwise, if the value of the parameters is too high, the map is no longer updated, and the system generates an increasing number of false alarms.

Another important parameter is the time to trigger an alarm. This parameter was set, in our case, to

$$\Delta t = 60 \text{ s}; \tag{38}$$

this parameter represents the time interval from when a track is considered stationary until an alarm is generated. If movement is detected, the timer resets.

In order to eliminate small objects that can generate false alarms, the proposed method filters the detections according to their size. Therefore, the algorithm eliminates all the objects with a size smaller than

$$\Delta a = 0.005 \cdot w \cdot h \tag{39}$$

where w represents the width of the input frame, while h is the height of the frame.

The last parameter used is the threshold from the background subtraction phase. This parameter can have values in the interval $[0, 255]$. The choice of this threshold affects the performance of the background subtraction stage and, implicitly, the overall performance of the algorithm. As the threshold value decreases, the method generates more false alarms. Contrarily, as the value increases, the algorithm starts to miss moving objects, which may lead to a reduced performance of the method. As a compromise, we determined that the most optimal value for this threshold was

$$th_{bg} = 16 \tag{40}$$

5. Evaluation and Results

In order to test the final performances of the proposed algorithm, we designed several testing stages. This section presents the metrics used in the testing phase and describes the scenarios and the results obtained. The first step was to test several neural models to identify the most suitable neural network for the presented scenario. In this sense, we compared the existing methods from the point of view of efficiency and cost in terms of the time and computing power required.

Later, after we chose the most optimal neural network, a series of tests were implemented to determine the performance of the proposed method.

5.1. Performance Evaluation Metrics

In order to correctly determine the performance of the proposed algorithm, we used several evaluation metrics in addition to the method's accuracy [31]. Computed individually, the accuracy is vague for determining a method's capabilities [32]. Different performance metrics are employed to quantify an algorithm's precision and efficacy during its evaluation. We computed the evaluation metrics characteristic of classification methods, such as mAP [33], recall [34], or F-score [35]. The metrics specific to segmentation methods, such as mIoU, were also used in the testing stage.

In order to be able to calculate the presented evaluation metrics, it is necessary to define four elementary theoretical concepts used in any classification system. These are presented in Table 1.

Table 1. Elementary concepts used in performance evaluation.

Elementary Concept	Abbreviation	Description
True positives	TP	It occurs when a detection annotated as belonging to class *C* is classified correctly by the system
True negatives	TN	It occurs when a detection annotated as belonging to a class other than class *C* is also classified by the system as not belonging to class *C*
False positives	FP	It occurs when a detection annotated as belonging to a class other than class *C* is classified incorrectly by the system as belonging to class *C*
False negatives	FN	It occurs when a detection annotated as belonging to class *C* is classified incorrectly by the system as belonging to a class other than class *C*

- **Accuracy.** Accuracy is the most frequently used metric in determining the performance of a detection and classification system. It provides a measure of the correctness of the classification, representing the ratio between valid detections and the total number of detections that the system generates. The accuracy is calculated according to

$$acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{41}$$

  This metric can take values in the range $[0, 1]$. The purpose of a classification system is to maximize the value of accuracy. As false positive and false negative detections decrease, the accuracy value approaches one, and the system is considered more efficient.
- **Mean average precision (mAP).** The mAP metric is considered a direct measure of the accuracy of a classifier. Its value is directly proportional to the accuracy value of the algorithm. It is a subunit value in the range $[0, 1]$, computed according to

$$mAP = \frac{TP}{TP + FP} \tag{42}$$

  As can be seen, the mAP value is directly impacted by the number of true positives and false positives detections. The higher number of false positive detections the algorithm generates, the lower the mAP value is, which leads to a system with low accuracy.
- **Recall or true positive rate (TPR).** Recall is the proportion of true positive detections to the total number of ground truth objects $(TP + FN)$. It measures the percentage of correctly detected objects relative to the total number of ground truth objects. Recall can be expressed mathematically as

$$recall = \frac{TP}{TP + FN} \tag{43}$$

Also named the TPR metric, the recall quantifies the number of predictions the system makes correctly, representing a measure of the positive predictions it misses. The recall value decreases when the system generates more false negative detections.

- **F1-score.** The F1-score is calculated according to Equation (44) and represents the weighted average between the previously presented evaluation metrics, mAP and recall. Like the other two metrics, it can take values in the range $[0, 1]$. Its value increases as the system generates fewer false positive and false negative detections.

$$F - score = 2 \cdot \frac{mAP \cdot recall}{mAP + recall} \tag{44}$$

Using the F1 score metric for evaluating the model's performance leads to a compromise between the system's precision and recall. It is a suitable metric for systems where it is not expected to maximize only one of the metrics.

- **Mean intersection over union (mIoU).** The global IoU score, namely, the mIoU, represents an average of the IoU score for the entire segmentation model, and it is computed in two steps. First, we calculate the IoU value associated with each class separately, according to (45). Afterward, we compute an average of the previously calculated scores for all the classes that can be classified using the model.

$$IoU = \frac{A_T \cap A_P}{A_T \cup A_P} \tag{45}$$

where A_T represents the area of ground truth, A_P represents the area of predicted objects, while $\cap$ refers to the intersection operation, and $\cup$ refers to the union operation. The intersection between two areas is formed by all the pixels belonging to both the predicted and annotated objects. On the other hand, the union of the two areas represents the set of pixels that belong to the predicted object or the ground truth. The value of each IoU practically measures the number of common pixels between the two areas of interest, divided by the total number of pixels present in the two objects.

By calculating these performance metrics, we can gain a better understanding of the algorithm's strengths and weaknesses and identify areas for future improvement.

5.2. Evaluating Semantic Segmentation Models

A semantic segmentation network is a crucial component of the proposed approach. In order to select the optimal one for our needs, we evaluated several of them to establish which had the optimal balance of accuracy, frames per second, and model size.

Due to the fact that anything could be present in a logistics warehouse, for validation, we chose a dataset that contained a large number of classes in a variety of scenarios: the ADE20K dataset [1].

In Table 2, we give the results obtained by each model. In the following subsection, we compare these methods and select the most appropriate model.

We represented the data from Table 2 in a scatter plot, indicating the method, the model's size in gigabytes, the inference time, and the mIoU, for easy visualization. Figure 7 demonstrates that none of the models are clearly superior to the others; however, when real-world constraints are considered, one emerges.

In order for our proposed algorithm to be applicable in the real world, it must utilize as few computing resources as possible while maintaining a high FPS and a good mIoU. Currently, it is reasonable to expect a standard graphics card to have at least 10 GB of RAM, so this was our first limitation. For the FPS, we determined that a minimum of 20 was acceptable, and a minimum of 40 was also acceptable for the mIoU.

Table 2. Tested semantic segmentation methods done on the ADE20K dataset [1].

Method	Backbone	Crop Size	Lr schd	Mem (GB)	Inf Time (fps)	mIoU	mIoU (ms + flip)
FCN [36]	R-50-D8	512 × 512	80,000	8.5	23.49	35.94	37.94
FCN	R-101-D8	512 × 512	80,000	12	14.78	39.61	40.83
FCN	R-50-D8	512 × 512	160,000	-	-	36.1	38.08
FCN	R-101-D8	512 × 512	160,000	-	-	39.91	41.4
PSPNet [37]	R-50-D8	512 × 512	80,000	8.5	23.53	41.13	41.94
PSPNet	R-101-D8	512 × 512	80,000	12	15.3	43.57	44.35
PSPNet	R-50-D8	512 × 512	160,000	-	-	42.48	43.44
PSPNet	R-101-D8	512 × 512	160,000	-	-	44.39	45.35
DeepLabV3 [38]	R-50-D8	512 × 512	80,000	8.9	14.76	42.42	43.28
DeepLabV3	R-101-D8	512 × 512	80,000	12.4	10.14	44.08	45.19
DeepLabV3	R-50-D8	512 × 512	160,000	-	-	42.66	44.09
DeepLabV3	R-101-D8	512 × 512	160,000	-	-	45	46.66
PSANet [39]	R-50-D8	512 × 512	80,000	9	18.91	41.14	41.91
PSANet	R-101-D8	512 × 512	80,000	12.5	13.13	43.8	44.75
PSANet	R-50-D8	512 × 512	160,000	-	-	41.67	42.95
PSANet	R-101-D8	512 × 512	160,000	-	-	43.74	45.38
DeepLabV3+ [40]	R-50-D8	512 × 512	80,000	10.6	21.01	42.72	43.75
DeepLabV3+	R-101-D8	512 × 512	80,000	14.1	14.16	44.6	46.06
DeepLabV3+	R-50-D8	512 × 512	160,000	-	-	43.95	44.93
DeepLabV3+	R-101-D8	512 × 512	160,000	-	-	45.47	46.35
UPerNet [22]	R-18	512 × 512	80,000	6.6	24.76	38.76	39.81
UPerNet	R-50	512 × 512	80,000	8.1	23.4	40.7	41.81
UPerNet	R-101	512 × 512	80,000	9.1	20.34	42.91	43.96
UPerNet	R-18	512 × 512	160,000	-	-	39.23	39.97
UPerNet	R-50	512 × 512	160,000	-	-	42.05	42.78
UPerNet	R-101	512 × 512	160,000	-	-	43.82	44.85
NonLocalNet [41]	R-50-D8	512 × 512	80,000	9.1	21.37	40.75	42.05
NonLocalNet	R-101-D8	512 × 512	80,000	12.6	13.97	42.9	44.27
NonLocalNet	R-50-D8	512 × 512	160,000	-	-	42.03	43.04
NonLocalNet	R-101-D8	512 × 512	160,000	-	-	44.63	45.79
EncNet [42]	R-50-D8	512 × 512	80,000	10.1	22.81	39.53	41.17
EncNet	R-101-D8	512 × 512	80,000	13.6	14.87	42.11	43.61
EncNet	R-50-D8	512 × 512	160,000	-	-	40.1	41.71
EncNet	R-101-D8	512 × 512	160,000	-	-	42.61	44.01
DANet [43]	R-50-D8	512 × 512	80,000	11.5	21.2	41.66	42.9
DANet	R-101-D8	512 × 512	80,000	15	14.18	43.64	45.19
DANet	R-50-D8	512 × 512	160,000	-	-	42.45	43.25
DANet	R-101-D8	512 × 512	160,000	-	-	44.17	45.02
APCNet [44]	R-50-D8	512 × 512	80,000	10.1	19.61	42.2	43.3
APCNet	R-101-D8	512 × 512	80,000	13.6	13.1	45.54	46.65

Table 2. *Cont.*

Method	Backbone	Crop Size	Lr schd	Mem (GB)	Inf Time (fps)	mIoU	mIoU (ms + flip)
APCNet	R-50-D8	512 × 512	160,000	-	-	43.4	43.94
APCNet	R-101-D8	512 × 512	160,000	-	-	45.41	46.63
CCNet [45]	R-50-D8	512 × 512	80,000	8.8	20.89	41.78	42.98
CCNet	R-101-D8	512 × 512	80,000	12.2	14.11	43.97	45.13
CCNet	R-50-D8	512 × 512	160,000	-	-	42.08	43.13
CCNet	R-101-D8	512 × 512	160,000	-	-	43.71	45.04
DMNet [46]	R-50-D8	512 × 512	80,000	9.4	20.95	42.37	43.62
DMNet	R-101-D8	512 × 512	80,000	13	13.88	45.34	46.13
DMNet	R-50-D8	512 × 512	160,000	-	-	43.15	44.17
DMNet	R-101-D8	512 × 512	160,000	-	-	45.42	46.76
ANN [47]	R-50-D8	512 × 512	80,000	9.1	21.01	41.01	42.3
ANN	R-101-D8	512 × 512	80,000	12.5	14.12	42.94	44.18
ANN	R-50-D8	512 × 512	160,000	-	-	41.74	42.62
ANN	R-101-D8	512 × 512	160,000	-	-	42.94	44.06
GCNet [48]	R-50-D8	512 × 512	80,000	8.5	23.38	41.47	42.85
GCNet	R-101-D8	512 × 512	80,000	12	15.2	42.82	44.54
GCNet	R-50-D8	512 × 512	160,000	-	-	42.37	43.52
GCNet	R-101-D8	512 × 512	160,000	-	-	43.69	45.21
FastFCN [49] + DeepLabV3	R-50-D32	512 × 512	80,000	8.46	12.06	41.88	42.91
FastFCN + DeepLabV3	R-50-D32	512 × 512	160,000	-	-	43.58	44.92
FastFCN + PSPNet	R-50-D32	512 × 512	80,000	8.02	19.21	41.4	42.12
FastFCN + PSPNet	R-50-D32	512 × 512	160,000	-	-	42.63	43.71
FastFCN + EncNet	R-50-D32	512 × 512	80,000	9.67	17.23	40.88	42.36
FastFCN + EncNet	R-50-D32	512 × 512	160,000	-	-	42.5	44.21
ISANet [50]	R-50-D8	512 × 512	80,000	9	22.55	41.12	42.35
ISANet	R-50-D8	512 × 512	160,000	9	22.55	42.59	43.07
ISANet	R-101-D8	512 × 512	80,000	12.562	10.56	43.51	44.38
ISANet	R-101-D8	512 × 512	160,000	12.562	10.56	43.8	45.4
OCRNet [51,52]	HRNetV2p-W18-Small	512 × 512	80,000	6.7	28.98	35.06	35.8
OCRNet	HRNetV2p-W18	512 × 512	80,000	7.9	18.93	37.79	39.16
OCRNet	HRNetV2p-W48	512 × 512	80,000	11.2	16.99	43	44.3
OCRNet	HRNetV2p-W18-Small	512 × 512	160,000	-	-	37.19	38.4
OCRNet	HRNetV2p-W18	512 × 512	160,000	-	-	39.32	40.8
OCRNet	HRNetV2p-W48	512 × 512	160,000	-	-	43.25	44.88
DNLNet [53]	R-50-D8	512 × 512	80,000	8.8	20.66	41.76	42.99

Table 2. *Cont.*

Method	Backbone	Crop Size	Lr schd	Mem (GB)	Inf Time (fps)	mIoU	mIoU (ms + flip)
DNLNet	R-101-D8	512 × 512	80,000	12.8	12.54	43.76	44.91
DNLNet	R-50-D8	512 × 512	160,000	-	-	41.87	43.01
DNLNet	R-101-D8	512 × 512	160,000	-	-	44.25	45.78

After applying these constraints to Table 2 and selecting the model with the highest mIoU, we concluded that the UPerNet model with the R-101 backbone was the optimal model, with an mIoU of 43.85.

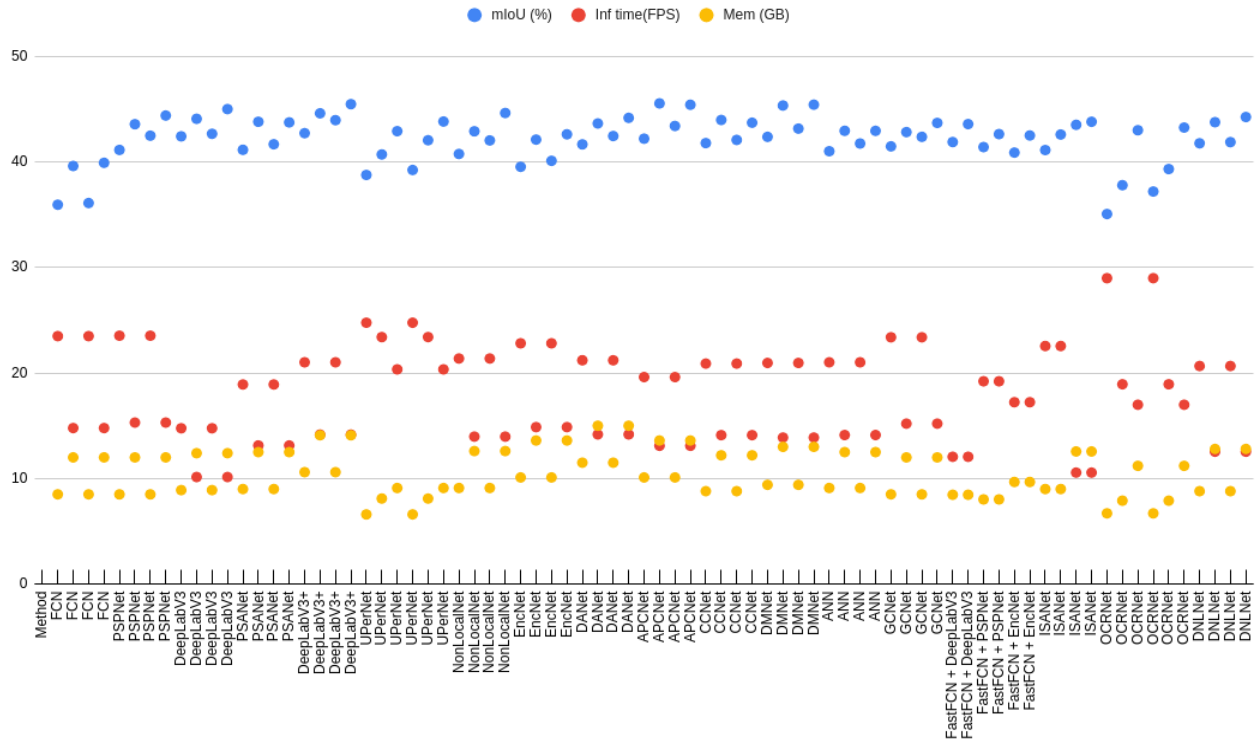

Figure 7. The method, the model's size in gigabytes, the inference time, and the mIoU.

5.3. Examples from Datasets

A dataset containing 120 h of video footage (at 24 fps, 10,368,000 images) from three industrial logistics warehouses was used to rigorously test the proposed algorithm. The dataset included a wide variety of scenarios, such as varying lighting conditions, different types of objects on the pallets, and various camera angles. This exhaustive testing allowed us to evaluate the algorithm's performance and identify areas for enhancement. Training was done on 75%, testing on 20%, and the validation on 5%.

In order to evaluate the performance of the proposed method, the dataset included three kinds of events: "Pallet," "Forklift," and "Fallen Material"—referring to any substantial object that falls from a shelf or from a moving pallet. These events were marked as present if any of the specified objects remained stationary in the video sequence.

In Figure 8, two events are depicted. If we examine Figure 8b, we can see that when an object is removed from the scene, the algorithm does not generate an alarm, but it does when something new and improperly placed is removed. The image demonstrates the system's ability to accurately identify and distinguish objects of interest from their surroundings. Combining the outputs of multiple algorithms, including the background subtraction, tracking, and semantic segmentation algorithms, enabled the detection. The

outcome is a clear and accurate representation of the target object, demonstrating the efficacy of the proposed method in detecting unidentified or misplaced objects within an industrial logistics warehouse.

Figure 8. Events from the system: (**a**) image before the event in (**b**); (**b**) event; (**c**) image before the event in (**d**); (**d**) event.

In the image depicting a system detection, the event is displayed as the last track position from the object tracking, as opposed to the actual detected region. Displaying the last track position is more helpful in a production environment, so this design decision was made for practical reasons. The image was chosen to illustrate the varied contents of a pallet and the potential for a forklift to be abandoned in an unclear location.

In addition to these events, in Figure 9, we illustrate the fact that the algorithm is robust and capable of running from different angles. In subimage b from Figure 9, the small size of the detected object is notable.

The delicate nature of the images in the dataset necessitates the strictest secrecy and discretion. In order to prevent these photographs from falling into the wrong hands, it is vital that we take the required safeguards. As a result, a tiny, separate room was made available for the presentation of events from the dataset. This area was outfitted with all the tools and resources necessary for the secure handling and exhibition of photographs. By showing the happenings from this room, we can reassure the participating logistics businesses that their data are being treated with the highest care. It is critical that we maintain the highest levels of security and confidentiality when dealing with sensitive material, and the availability of a separate space for the presentation of such information is a crucial step in attaining this goal.

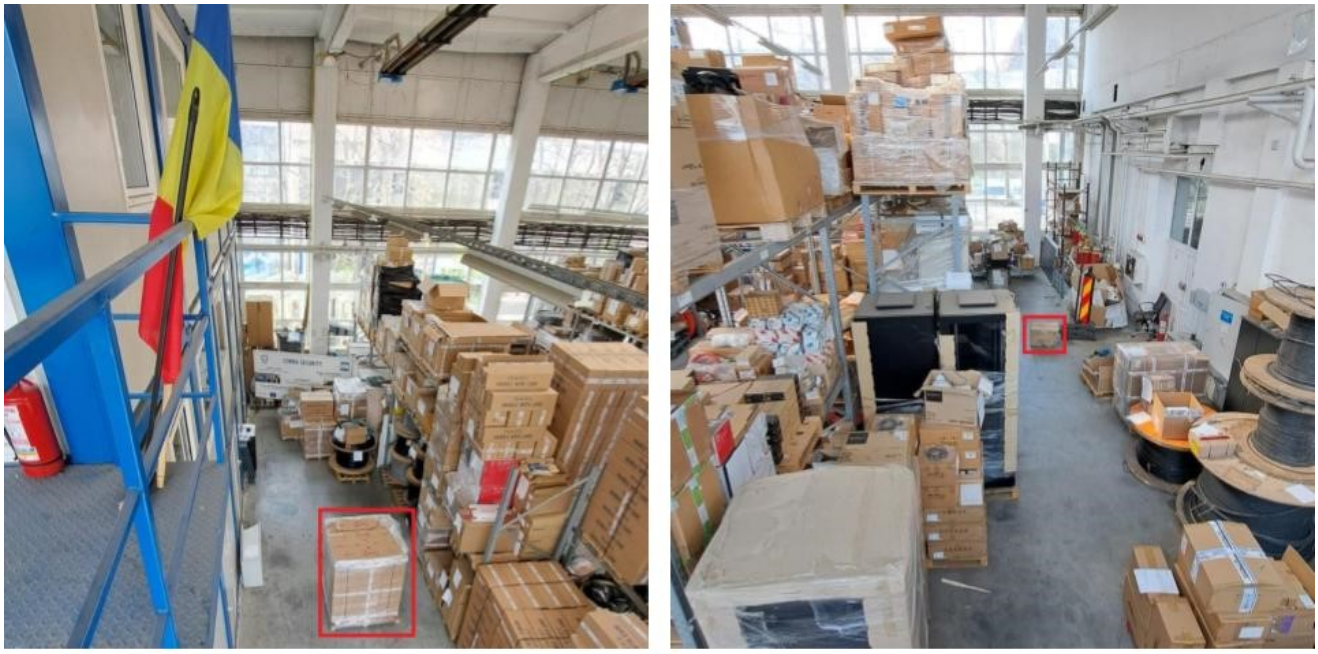

Figure 9. Alarms from different testing angles.

5.4. Experimental Results

As stated in Section 2, the existing methods do not apply to our scenario, so it is hard to compare those approaches and our proposed method directly. In order to present the novelty brought by the proposed method and to outline its benefits, we designed a two-part testing scenario.

The first testing phase was implemented to compare the existing semantic segmentation model and was described in the previous section. Those tests demonstrated that the UPerNet model was the most suitable in terms of semantic segmentation (Table 2).

In the second part of the testing stage, our primary goal was to outline the benefits of each new component added according to the diagram presented in Figure 5. In order to evaluate the effectiveness of our suggested strategy and its components, we report the outcomes of each iteration. First, we tested the proposed method only using the semantic segmentation block, which consisted of the UPerNet model. Second, we added a tracker to see how it affected the overall performance of the proposed algorithm. Finally, we added the last processing block, the majority vote. Each partial architecture was evaluated using the same dataset to compare their performance in the same circumstances. The obtained experimental results are presented in this section.

The initial version of the system relied solely on semantic segmentation and triggered an event if an object obscured the floor. As seen in Table 3, the outcomes of this method were not optimal, as false alarms were produced by each moving object traversing the image.

The tracker was introduced to reduce the high number of false alarms. This prevented alerts from being activated by moving objects. As shown in Table 4, this significantly improved the outcomes, yet they were still unacceptable.

At this point, the fact that the reference floor map did not adapt over time was the cause of most of the issues. By adding a majority vote update system to the reference map, the results were greatly improved, as seen in Table 5.

Table 3. Results after testing the proposed method: semantic segmentation.

Site	Human Label	TP	TN	FP	FN	mAP	Recall	F-Score	Accuracy
1	Pallet	57	228	250	0	0.18	1	0.31	0.53
1	Forklift	3	423	73	0	0.03	1	0.07	0.85
1	Fallen material	6	38	88	0	0.06	1	0.12	0.33
2	Pallet	31	233	156	0	0.16	1	0.28	0.62
2	Forklift	0	368	61	0	0	-	-	0.85
2	Fallen material	0	51	129	0	0	-	-	0.28
3	Pallet	30	354	91	0	0.24	1	0.39	0.80
3	Forklift	0	226	88	0	0	-	-	0.71
3	Fallen material	17	1	110	0	0.13	1	0.23	0.14
Total	Pallet	118	1112	497	0	0.19	1	0.32	0.71
Total	Forklift	3	652	222	0	0.01	1	0.02	0.74
Total	Fallen material	23	581	327	0	0.06	1	0.12	0.64
Overall	-	**144**	**2345**	**1046**	**0**	**0.12**	**0.95**	**0.21**	**0.70**

Table 4. Results after testing the proposed method: semantic segmentation + tracking.

Site	Human Label	TP	TN	FP	FN	mAP	Recall	F-Score	Accuracy
1	Pallet	57	404	72	0	0.44	1	0.61	0.86
1	Forklift	3	437	59	0	0.48	1	0.09	0.88
1	Fallen material	6	94	32	0	0.15	1	0.27	0.75
2	Pallet	31	338	51	0	0.37	1	0.54	0.87
2	Forklift	0	383	46	0	0	-	-	0.89
2	Fallen material	0	123	57	0	0	-	-	0.68
3	Pallet	30	376	69	0	0.30	1	0.46	0.85
3	Forklift	0	281	33	0	0	-	-	0.89
3	Fallen material	16	20	91	1	0.14	0.94	0.25	0.28
Total	Pallet	114	1118	192	0	0.37	1	0.54	0.86
Total	Forklift	3	1101	138	0	0.02	1	0.04	0.88
Total	Fallen material	20	237	179	1	0.10	0.95	0.18	0.58
Overall	-	**143**	**2456**	**509**	**1**	**0.21**	**0.99**	**0.35**	**0.83**

Due to the minuscule size of fallen objects, the proposed algorithm exhibited limitations in detecting them. In some instances, dropped objects were overlooked because they were obscured by the shelves' pallets. The missed pallets nearly completely overlapped with the pallets stored on the shelves, making it difficult to distinguish and detect them with the current method.

This algorithm is notable due to the fact that it operates in real time with a mean of 45 frames per second, although this is dependent on the hardware used (in our case, NVIDIA's 1080 GPU). This indicates that it is capable of producing results quickly and without significant delay. This application requires the ability to process data in real time because it contains time-sensitive information.

Table 5. Results after testing the proposed method: semantic segmentation + tracking + majority vote.

Site	Human Label	TP	TN	FP	FN	mAP	Recall	F-Score	Accuracy
1	Pallet	55	474	4	2	0.93	0.96	0.94	0.98
1	Forklift	3	496	2	0	0.6	1	0.75	0.99
1	Fallen material	5	125	0	1	1	0.83	0.90	0.99
2	Pallet	31	389	3	0	0.93	1	0.96	0.99
2	Forklift	0	429	1	0	-	-	-	0.99
2	Fallen material	0	180	0	0	-	-	-	1
3	Pallet	28	443	3	2	0.93	0.93	0.93	0.98
3	Forklift	0	314	0	0	-	-	-	1
3	Fallen material	15	109	0	2	0.88	0.88	0.88	0.98
Total	Pallet	114	1306	10	4	0.93	0.96	0.94	0.99
Total	Forklift	3	1239	3	0	1	1	1	0.99
Total	Fallen material	20	414	0	3	1	0.86	0.92	0.99
Overall	-	**137**	**2959**	**13**	**7**	**0.91**	**0.95**	**0.92**	**0.99**

6. Conclusions

The proposed algorithm for detecting misplaced or fallen pallets within a logistics warehouse demonstrated promising results. Utilizing a semantic segmentation and a majority vote approach, a map of the warehouse floor was created to accurately detect and track objects. On 120 h of video from an industrial logistics warehouse, the algorithm was evaluated using evaluation metrics such as true positive, false negative, and recall.

While the system was able to detect the majority of pallets within the warehouse, some smaller fallen objects and overlapping pallets were missed. However, these limitations were outweighed by the algorithm's success in detecting the vast majority of pallets and improving the overall warehouse operations' efficiency.

In conclusion, the proposed algorithm significantly enhances the capability of detecting and tracking pallets within a logistics warehouse. Its ability to accurately detect misplaced or fallen pallets provides logistics companies with valuable information, enhancing workplace safety and efficiency. Future success will be even greater as a result of the algorithm's continued development and refinement, which will enhance its performance.

Author Contributions: Conceptualization, S.V.C.; Writing—original draft, S.V.C. and R.M.; Writing—review & editing, M.G.; Supervision, S.V.C.; Funding acquisition, M.G. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by the Romanian UEFISCDI Agency under projects PN3-P2-1166 / 29Sol and PN3-P2-1167 / 28Sol.

Data Availability Statement: Not applicable.

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

References

1. Zhou, B.; Zhao, H.; Puig, X.; Fidler, S.; Barriuso, A.; Torralba, A. Scene parsing through ade20k dataset. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 633–641.
2. Zaccaria, M.; Monica, R.; Aleotti, J. A comparison of deep learning models for pallet detection in industrial warehouses. In Proceedings of the 2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP), Cluj-Napoca, Romania, 3–5 September 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 417–422.

3. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *Adv. Neural Inf. Process. Syst.* **2015** , *28*. [CrossRef] [PubMed]
4. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Washington, DC, USA, 23–28 June 2014; pp. 580–587.
5. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. Ssd: Single shot multibox detector. In Proceedings of the Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, 11–14 October 2016; Proceedings, Part I 14; Springer: Berlin/Heidelberg, Germany, 2016; pp. 21–37.
6. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
7. Li, Y.Y.; Chen, X.H.; Ding, G.Y.; Wang, S.; Xu, W.C.; Sun, B.B.; Song, Q. Pallet detection and localization with RGB image and depth data using deep learning techniques. In Proceedings of the 2021 6th International Conference on Automation, Control and Robotics Engineering (CACRE), Dalian, China, 15–17 July 20212; IEEE: Piscataway, NJ, USA, 2021; pp. 306–310.
8. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Region-based convolutional networks for accurate object detection and segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2015**, *38*, 142–158. [CrossRef] [PubMed]
9. Varga, R.; Nedevschi, S. Robust Pallet Detection for Automated Logistics Operations. In Proceedings of the VISIGRAPP (4: VISAPP), Rome, Italy, 27 February 2016; pp. 470–477.
10. Jain, A.K.; Mao, J.; Mohiuddin, K.M. Artificial neural networks: A tutorial. *Computer* **1996**, *29*, 31–44. [CrossRef]
11. Kanal, L.N. Perceptron. In *Encyclopedia of Computer Science*; Wiley: New York, NY, USA, 2003; pp. 1383–1385.
12. Ramchoun, H.; Ghanou, Y.; Ettaouil, M.; Janati Idrissi, M.A. Multilayer perceptron: Architecture optimization and training. *Int. J. Interact. Multimed. Artif. Intell.* **2016**, *4*, 26–30. [CrossRef]
13. Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. JMLR Workshop and Conference Proceedings, Sardinia, Italy, 13–15 May 2010; pp. 249–256.
14. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
15. Ruder, S. An overview of gradient descent optimization algorithms. *arXiv* **2016**, arXiv:1609.04747.
16. Bottou, L.; et al. Stochastic gradient learning in neural networks. *Proc. Neuro-Nımes* **1991**, *91*, 12.
17. Duchi, J.; Hazan, E.; Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.* **2011**, *12*, 2121–2159.
18. Alom, M.Z.; Taha, T.M.; Yakopcic, C.; Westberg, S.; Sidike, P.; Nasrin, M.S.; Hasan, M.; Van Essen, B.C.; Awwal, A.A.; Asari, V.K. A state-of-the-art survey on deep learning theory and architectures. *Electronics* **2019**, *8*, 292. [CrossRef]
19. Dumoulin, V.; Visin, F. A guide to convolution arithmetic for deep learning. *arXiv* **2016**, arXiv:1603.07285.
20. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
21. He, K.; Zhang, X.; Ren, S.; Sun, J. Identity mappings in deep residual networks. In Proceedings of the Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, 11–14 October 2016; Proceedings, Part IV 14; Springer: Berlin/Heidelberg, Germany, 2016; pp. 630–645.
22. Xiao, T.; Liu, Y.; Zhou, B.; Jiang, Y.; Sun, J. Unified perceptual parsing for scene understanding. In Proceedings of the European conference on computer vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 418–434.
23. Long, J.; Shelhamer, E.; Darrell, T. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 3431–3440.
24. Everingham, M.; Eslami, S.M.A.; Van Gool, L.; Williams, C.K.I.; Winn, J.; Zisserman, A. The Pascal Visual Object Classes Challenge: A Retrospective. *Int. J. Comput. Vis.* **2015**, *111*, 98–136. [CrossRef]
25. Cordts, M.; Omran, M.; Ramos, S.; Rehfeld, T.; Enzweiler, M.; Benenson, R.; Franke, U.; Roth, S.; Schiele, B. The cityscapes dataset for semantic urban scene understanding. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 3213–3223.
26. Lin, T.Y.; Dollár, P.; Girshick, R.; He, K.; Hariharan, B.; Belongie, S. Feature pyramid networks for object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 2117–2125.
27. Zhao, H.; Shi, J.; Qi, X.; Wang, X.; Jia, J. Pyramid scene parsing network. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 2881–2890.
28. Piccardi, M. Background subtraction techniques: A review. In Proceedings of the 2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583), Hague, The Netherlands, 10–13 October 2004; IEEE: Piscataway, NJ, USA, 2004; Volume 4; pp. 3099–3104.
29. Plott, C.R. A notion of equilibrium and its possibility under majority rule. *Am. Econ. Rev.* **1967**, *57*, 787–806.
30. Nascimento, J.C.; Abrantes, A.J.; Marques, J.S. An algorithm for centroid-based tracking of moving objects. In Proceedings of the 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No. 99CH36258), Phoenix, AZ, USA, 15–19 March 1999; IEEE: Piscataway, NJ, USA, 1999; Volume 6, pp. 3305–3308.
31. Zhou, J.; Gandomi, A.H.; Chen, F.; Holzinger, A. Evaluating the quality of machine learning explanations: A survey on methods and metrics. *Electronics* **2021**, *10*, 593. [CrossRef]

32. Grandini, M.; Bagli, E.; Visani, G. Metrics for multi-class classification: An overview. *arXiv* **2020**, arXiv:2008.05756.
33. Li, K.; Huang, Z.; Cheng, Y.C.; Lee, C.H. A maximal figure-of-merit learning approach to maximizing mean average precision with deep neural network based classifiers. In Proceedings of the 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Florence, Italy, 4–9 May 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 4503–4507.
34. Winkler, J.P.; Grönberg, J.; Vogelsang, A. Optimizing for recall in automatic requirements classification: An empirical study. In Proceedings of the 2019 IEEE 27th International Requirements Engineering Conference (RE), Jeju Island, Republic of Korea, 23–27 September 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 40–50.
35. Sokolova, M.; Japkowicz, N.; Szpakowicz, S. Beyond accuracy, F-score and ROC: A family of discriminant measures for performance evaluation. In Proceedings of the AI 2006: Advances in Artificial Intelligence: 19th Australian Joint Conference on Artificial Intelligence, Hobart, Australia, 4–8 December 2006; Proceedings 19; Springer: Berlin/Heidelberg, Germany, 2006; pp. 1015–1021.
36. Shelhamer, E.; Long, J.; Darrell, T. Fully convolutional networks for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *39*, 640–651. [CrossRef] [PubMed]
37. Wightman, R.; Touvron, H.; Jégou, H. Resnet strikes back: An improved training procedure in timm. *arXiv* **2021**, arXiv:2110.00476.
38. Chen, L.C.; Papandreou, G.; Schroff, F.; Adam, H. Rethinking atrous convolution for semantic image segmentation. *arXiv* **2017**, arXiv:1706.05587.
39. Zhao, H.; Zhang, Y.; Liu, S.; Shi, J.; Change Loy, C.; Lin, D.; Jia, J. Psanet: Point-wise spatial attention network for scene parsing. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 267–283.
40. Chen, L.C.; Zhu, Y.; Papandreou, G.; Schroff, F.; Adam, H. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. In Proceedings of the ECCV, Munich, Germany, 8–14 September 2018.
41. Wang, X.; Girshick, R.; Gupta, A.; He, K. Non-local neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 7794–7803.
42. Zhang, H.; Dana, K.; Shi, J.; Zhang, Z.; Wang, X.; Tyagi, A.; Agrawal, A. Context Encoding for Semantic Segmentation. In Proceedings of the The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–23 June 2018.
43. Fu, J.; Liu, J.; Tian, H.; Li, Y.; Bao, Y.; Fang, Z.; Lu, H. Dual Attention Network for Scene Segmentation. In Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2019.
44. He, J.; Deng, Z.; Zhou, L.; Wang, Y.; Qiao, Y. Adaptive Pyramid Context Network for Semantic Segmentation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2019.
45. Huang, Z.; Wang, X.; Huang, L.; Huang, C.; Wei, Y.; Liu, W. CCNet: Criss-Cross Attention for Semantic Segmentation. In Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Republic of Korea, 27 October–2 November 2019.
46. He, J.; Deng, Z.; Qiao, Y. Dynamic Multi-Scale Filters for Semantic Segmentation. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Long Beach, CA, USA, 15–20 June 2019.
47. Zhu, Z.; Xu, M.; Bai, S.; Huang, T.; Bai, X. Asymmetric non-local neural networks for semantic segmentation. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Long Beach, CA, USA, 15–20 June 2019; pp. 593–602.
48. Cao, Y.; Xu, J.; Lin, S.; Wei, F.; Hu, H. Gcnet: Non-local networks meet squeeze-excitation networks and beyond. In Proceedings of the IEEE International Conference on Computer Vision Workshops, Seoul, Republic of Korea, 27–28 October 2019.
49. Wu, H.; Zhang, J.; Huang, K.; Liang, K.; Yu, Y. Fastfcn: Rethinking dilated convolution in the backbone for semantic segmentation. *arXiv* **2019**, arXiv:1903.11816.
50. Yuan, Y.; Huang, L.; Guo, J.; Zhang, C.; Chen, X.; Wang, J. OCNet: Object Context for Semantic Segmentation. *Int. J. Comput. Vis.* **2021**, *129*, 2375–2398. [CrossRef]
51. Yuan, Y.; Wang, J. Ocnet: Object context network for scene parsing. *arXiv* **2018**, arXiv:1809.00916.
52. Yuan, Y.; Chen, X.; Wang, J. Object-Contextual Representations for Semantic Segmentation. In Proceedings of the Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, 23–28 August 2020.
53. Yin, M.; Yao, Z.; Cao, Y.; Li, X.; Zhang, Z.; Lin, S.; Hu, H. Disentangled Non-Local Neural Networks. In Proceedings of the Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, 23–28 August 2020.